Keysight Infiniium MXR/EXR-Series Oscilloscopes

KEYSIGHT TECHNOLOGIES

Programmer's Guide

Notices

Revision

Version 11.30.00000

Edition

March 2022

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 Infiniium oscilloscopes that have the 11.00 or greater user interface software. Supported models include:

- MXR/EXR-Series oscilloscopes.

In this book, Chapter 1, “What's New,” starting on page 51, describes programming command changes in the latest version of oscilloscope software.

Chapter 2, “Setting Up,” starting on page 83, describes the steps you must take before you can control the oscilloscope with remote programs.

The next several chapters give you an introduction to programming the oscilloscopes, along with necessary conceptual information. These chapters describe basic program communications, interface, syntax, data types, and status reporting:

The next chapters describe the commands used to program the oscilloscopes. Each chapter describes the set of commands that belong to an individual subsystem, and explains the function of each command.

Chapter 49, "Obsolete and Discontinued Commands," starting on page 1821, describes obsolete (deprecated) commands that still work but have been replaced by newer commands, and lists discontinued commands that are no longer supported.

Chapter 51, "Error Messages," starting on page 2013, describes error messages.

Chapter 52, "Example Programs," starting on page 2027, shows example programs in various languages using the VISA COM, VISA, and SICL libraries.

Finally, Chapter 53, "Reference," starting on page 2159, contains file format descriptions.

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 controller PC interface configuration, see the documentation for the interface card used (for example, the Keysight 82350A GPIB interface).
- 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/Infiniium-manuals

Contents

29 :LANE<N> (Equalization) Commands

33 :MARKer Commands

34 :MEASure Commands

46 :WGEN (Waveform Generator) Commands

47 :WMEMory (Waveform Memory) Commands

1 What's New

What's New in Version 11.30

New command descriptions for Version 11.30 of the Infiniium MXR/EXR-Series oscilloscope software appear below.

New Commands

Command	Description
:ANALyze:SNDR:CORRected (see page 357)	Lets you enable or disable the display of the Pulse Corrected waveform.
:ANALyze:SNDR:CORRected:HORizontal (see page 358)	When the Pulse Corrected waveform is displayed, this command selects either automatic or manual horizontal scaling.
:ANALyze:SNDR:CORRected:HORizontal:POSition (see page 359)	When the Pulse Corrected waveform is displayed and manual horizontal scaling is selected, this command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.
:ANALyze:SNDR:CORRected:HORizontal:SCALe (see page 360)	When the Pulse Corrected waveform is displayed and manual horizontal scaling is selected, this command specifies the waveform's horizontal time per division.
:ANALyze:SNDR:CORRected:VERTical (see page 361)	When the Pulse Corrected waveform is displayed, this command selects either automatic or manual vertical scaling.
:ANALyze:SNDR:CORRected:VERTical:OFFSet (see page 362)	When the Pulse Corrected waveform is displayed and manual vertical scaling is selected, this command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.
:ANALyze:SNDR:CORRected:VERTical:SCALe (see page 363)	When the Pulse Corrected waveform is displayed and manual vertical scaling is selected, this command specifies the waveform's vertical volts per division.
:ANALyze:SNDR:ERRor (see page 364)	Lets you enable or disable the display of the linear fit Error e(k) waveform.
:ANALyze:SNDR:ERRor:VERTical (see page 365)	When the linear fit Error e(k) waveform is displayed, this command selects either automatic or manual vertical scaling.
:ANALyze:SNDR:ERRor:VERTical:OFFSet (see page 366)	When the linear fit Error e(k) waveform is displayed and manual vertical scaling is selected, this command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.
:ANALyze:SNDR:ERRor:VERTical:SCALe (see page 367)	When the linear fit Error e(k) waveform is displayed and manual vertical scaling is selected, this command specifies the waveform's vertical volts per division.
:ANALyze:SNDR:LFPR (see page 368)	Lets you enable or disable the display of the Linear Fit Pulse Response p(k) waveform.
:ANALyze:SNDR:LFPR:HORizontal (see page 369)	When the Linear Fit Pulse Response p(k) waveform is displayed, this command selects either automatic or manual horizontal scaling.

Command	Description
:ANALyze:SNDR:LFPR:HORizontal:POSition (see page 370)	When the Linear Fit Pulse Response p(k) waveform is displayed and manual horizontal scaling is selected, this command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.
:ANALyze:SNDR:LFPR:HORizontal:SCALe (see page 371)	When the Linear Fit Pulse Response p(k) waveform is displayed and manual horizontal scaling is selected, this command specifies the waveform's horizontal time per division.
:ANALyze:SNDR:LFPR:VERTical (see page 372)	When the Linear Fit Pulse Response p(k) waveform is displayed, this command selects either automatic or manual vertical scaling.
:ANALyze:SNDR:LFPR:VERTical:OFFSet (see page 373)	When the Linear Fit Pulse Response p(k) waveform is displayed and manual vertical scaling is selected, this command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.
:ANALyze:SNDR:LFPR:VERTical:SCALe (see page 374)	When the Linear Fit Pulse Response p(k) waveform is displayed and manual vertical scaling is selected, this command specifies the waveform's vertical volts per division.
:ANALyze:SNDR:INPut (see page 375)	Lets you enable or disable the display of the SNDR Input waveform.
:ANALyze:SNDR:INPut:HORizontal (see page 376)	When the SNDR Input waveform is displayed, this command selects either automatic or manual horizontal scaling.
:ANALyze:SNDR:INPut:HORizontal:POSition (see page 377)	When the SNDR Input waveform is displayed and manual horizontal scaling is selected, this command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.
:ANALyze:SNDR:INPut:HORizontal:SCALe (see page 378)	When the SNDR Input waveform is displayed and manual horizontal scaling is selected, this command specifies the waveform's horizontal time per division.
:ANALyze:SNDR:INPut:NAVerages (see page 379)	Specifies the number of patterns to average in the linear fit math function computation.
:ANALyze:SNDR:INPut:PPUI (see page 380)	Specifies the number of sample points per UI (symbol) to use in the linear fit math function computation.
:ANALyze:SNDR:INPut:VERTical (see page 381)	When the SNDR Input waveform is displayed, this command selects either automatic or manual vertical scaling.
:ANALyze:SNDR:INPut:VERTical:OFFSet (see page 382)	When the SNDR Input waveform is displayed and manual vertical scaling is selected, this command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.
:ANALyze:SNDR:INPut:VERTical:SCALe (see page 383)	When the SNDR Input waveform is displayed and manual vertical scaling is selected, this command specifies the waveform's vertical volts per division.

Command	Description
:ANALyze:SNDR:LFPDelay (see page 384)	Sets the linear fit math function's pulse delay parameter.
:ANALyze:SNDR:LFPLength (see page 385)	Sets the linear fit math function's pulse length parameter.
:ANALyze:SNDR:SOURce (see page 386)	Specifies the input waveform source for the PAM-4 Signal to Noise and Distortion Ratio (SNDR) measurements.
:DISK:PUTFILE (see page 535)	Writes bytes to a full-path location on the oscilloscope. This is not a new command but its description is now being included.
:DISPlay:PAM:GRAPh (see page 588)	Selects the PAM-4 12-Edge Jitter graphs to display.
:DISPlay:PAM:SOURce (see page 589)	Selects the PAM-4 12-Edge Jitter sources whose tables and graphs will be displayed.
:DISPlay:PAM:TABLe (see page 590)	Selects the PAM-4 12-Edge Jitter tables to display.
:FUNCtion<F>:PSDensity (see page 675)	Computes the Power Spectral Density (PSD) of the specified channel, function, or memory.
:LANE<N>:EQUalizer:DFE:TAP:LFPDelay (see page 788)	Specifies the Linear Fit Pulse Delay parameter when using the PMAX-Normalized DFE tap optimization algorithm.
:LANE<N>:EQUalizer:DFE:TAP:LFPLength (see page 789)	Specifies the Linear Fit Pulse Length parameter when using the PMAX-Normalized DFE tap optimization algorithm.
:LANE<N>:EQUalizer:DFE:TAP:SEED (see page 796)	When you select the PMAX-Normalized DFE tap optimization algorithm (see :LANE<N>:EQUalizer:DFE:TAP:ALGorithm), this command lets you specify whether taps should be seeded and optionally what seed value to use.
:LXI:IDENtify[:STATe] (see page 836)	Lets you identify an oscilloscope.
:MEASure:PAM:EYE:WSDeviation (see page 1025)	When a Gaussian window shape is selected for :MEASure:PAM:EYE:WSHape, this command specifies the standard deviation (σ) of the Gaussian shape as a percent of the UI (Unit Interval).
:MEASure:PAM:EYE:WSHape (see page 1026)	Specifies the window shape of how data is weighted with respect to the recovered clock location when calculating PAM-4 eye height and VEC (Vertical Eye Closure).
:MEASure:SLEVel (see page 1119)	Installs, or returns the value of, a Level Mean measurement on the SNDR Input PAM-4 waveform.
:MEASure:SNDR (see page 1122)	Installs, or returns the value of, a Signal to Noise and Distortion Ratio (SNDR) measurement on a PAM-4 waveform.

Command	Description
:MEASure:UNDershoot (see page 1190)	Installs, or returns the value of, an Undershoot measurement.
:MEASure:VUNDershoot (see page 1210)	Installs, or returns the value of, a V undershoot measurement.

Changed Commands

Some commands that have waveform source parameters can now accept the INPut (SNDR Input), CORRected (linear fit Pulse Corrected), ERRor (linear fit Error e(k)), and LFPR (Linear Fit Pulse Response p(k)) sources that can be enabled by the Signal to Noise and Distortion Ratio (SNDR) measurements.

Command	Description
:FUNCtion<F>:FFT:VUNits (see page 642)	Added the DBMHZ, WATTHZ, and VRTHZ units for the Power Spectral Density math function.
:MARKer:MEASurement:MEASurement (see page 841)	Parameters have been added to support multiple measurement tracking markers.

Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:MEASure:LRLLevel	none	There is now a Level Mean measurement, :MEASure:SLEVel (see page 1119), that is one of the PAM-4 Signal to Noise and Distortion Ratio (SNDR) measurements.

What's New in Version 11.25

New command descriptions for Version 11.25 of the Infiniium MXR/EXR-Series oscilloscope software appear below.

New Commands

Command	Description
:LANE<N>:EQUalizer:DFE:TAP:STATus? (see page 797)	Returns whether the optimization process would choose a tap value that would be clamped by the limits or would be within the limits.
:MEASure:DCAI (see page 915)	Installs, or returns the value of, a duty cycle adjustment measurement for the I phase of a DDR5 DQS (strobe) signal.
:MEASure:DCAQ (see page 916)	Installs, or returns the value of, a duty cycle adjustment measurement for the Q phase of a DDR5 DQS (strobe) signal.
:MEASure:DCIPrime (see page 917)	Installs, or returns the value of, a duty cycle adjustment measurement for the I' (I prime) phase of a DDR5 DQS (strobe) signal.
:MEASure:DCQPrime (see page 918)	Installs, or returns the value of, a duty cycle adjustment measurement for the Q' (Q prime) phase of a DDR5 DQS (strobe) signal.
:MEASure:LRLLevel	Installs, or returns the value of, a Long-Run-Length Level measurement on a PAM-4 waveform.

Changed Commands

Command	Description
:LTESt:MEASurement (see page 830)	The maximum number of measurements has increased from 20 to 40, so wherever <N> is used to define a measurement number (for example, in the :MEASurement<N> commands or in MEAS<N> command parameters), it can now be an integer from 1 to 40.
:LTESt:RESults (see page 831)	
:MARKer:MEASurement:MEASurement (see page 841)	
:MEASure:JITTer:MEASurement (see page 968)	
:MEASure:NSIGma (see page 1004)	Results can now be returned per symbol level.
:MEASure:PAM:PRBS13q:PATTern (see page 1045)	Specifies the edge definition for the 12-edge jitter measurements' data pattern (either PRBS13Q, PCIE6 (52 symbols), or a custom data pattern).
:MEASure:RJDJ:ALL? (see page 1087)	New jitter measurement results for Uncorrelated Total Jitter (UTJ(p-p)) and Uncorrelated Deterministic Jitter Delta-Delta (UDJδ-δ) have been added.

Command	Description
:MEASure:WINDow (see page 1212)	The maximum number of measurements has increased from 20 to 40, so wherever <N> is used to define a measurement number (for example, in the :MEASurement<N> commands or in MEAS<N> command parameters), it can now be an integer from 1 to 40.
:MEASure:ZTMAX (see page 1215)	
:MEASure:ZTMIN (see page 1216)	
:MEASurement<N>:CLEar (see page 1217)	
:MEASurement<N>:NAME (see page 1218)	
:MEASurement<N>:POSition (see page 1219)	
:MEASurement<N>:SOURce (see page 1220)	
:MEASurement<N>:ZTMAX (see page 1221)	
:MEASurement<N>:ZTMIN (see page 1222)	

What's New in Version 11.20

New command descriptions for Version 11.20 of the Infiniium MXR/EXR-Series oscilloscope software appear below.

New Commands

Command	Description
:DIGital<N>:POSition (see page 522)	Sets the vertical position of a digital waveform within its waveform area grid.
:DISPlay:BOOKmark:DELete:ALL (see page 551)	Deletes all bookmarks.
:DISPlay:GRATicule:AREA<N>:HSCale (see page 568)	Specifies whether the horizontal scale is displayed.
:DISPlay:GRATicule:AREA<N>:VSCale (see page 570)	Specifies whether the vertical scale is displayed with automatic sizing (AUTO), displayed with handles for manual sizing (MANual), or not displayed (OFF).
:HOSTed:EOACq (see page 723)	Enables or disables the **Enable Offline Acquisition** user preference.
:MEASure:FFT:THDistortion (see page 945)	Measures the Total Harmonic Distortion (THD) using the magnitude of the fundamental frequency and as many harmonics as are on the screen.
:MEASure:PN:DESKew (see page 1058)	When two channels are used, this command will automatically remove time skew due to different length cables connecting the device under test (DUT) to the oscilloscope.
:MEASure:XCPeriod (see page 1214)	Measures cross-correlated Period using the same *two-channel cross-correlation technique* that is used when measuring phase noise.
:MTESt:CPOLygons (see page 1247)	When hardware-accelerated mask testing is not being used (see :MTESt:HWMask:ACTive?), this command specifies whether polygons should be counted in mask test results.
:MTESt:HWMask:ACTive? (see page 1260)	When using hardware-accelerated mask testing is enabled (:MTESt:UHWMask), this query returns whether hardware-acceleration is actually being used.
:MTESt:STATistics (see page 1267)	Specifies the type(s) of counts that should be included in the mask testing pass/fail statistics results.
:MTESt:UHWMask (see page 1269)	Enables or disables using hardware-accelerated mask testing.
:POWer:SIGNals:OVERshoot:UNITs (see page 1381)	Specifies the units of the :POWer:SIGNals:OVERshoot value.

Command	Description
:REPository Commands (see page 1403)	Commands for uploading measurement data results to Keysight data analytics web service repositories (either the N8844A or the KS6800A Series).
:TRIGger:IFMagn Commands (see page 1599)	Available on MXR-Series oscilloscopes, commands for the IF Magnitude trigger mode.

Changed Commands

Command	Description
:ANALyze:SIGNal:TYPE (see page 404)	Added the PAM6 and PAM8 signal types.
:CHANnel<N>:PROBe:HEAD:VTERm (see page 487)	Supports the MX0020A/21A/22A/24A/25A probe amplifiers.
:DISPlay:JITTer:THReshold (see page 583)	Threshold levels for PAM-6 and PAM-8 signal types are now supported.
:DISPlay:NOISe:LEVel (see page 587)	Levels for PAM-6 and PAM-8 signal types are now supported.
:LANE<N>:EQUalizer:CTLE:NUMPoles (see page 768)	Added the P6Z2 (6 Pole, 2 Zeros) option for the "number of poles" selection.
:LISTer:DATA? (see page 822)	Added the BUS parameter for returning Digital Listing data.
:MEASure:CGRade:EHEight (see page 889)	The <threshold> parameter now supports PAM-6 and PAM-8 eyes.
:MEASure:CGRade:EWIDth (see page 892)	The <threshold> parameter now supports PAM-6 and PAM-8 eyes.
:MEASure:FALLtime (see page 932)	With PAM6 and PAM8 signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.
:MEASure:NOISe:ALL? (see page 991)	Parameters are added for levels in the PAM-6 and PAM-8 signal types.
:MEASure:NOISe:RN (see page 997)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:NOISe:SCOPe:RN (see page 999)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:PAM:ELEVel (see page 1014)	Eyes for PAM-6 and PAM-8 are now supported.
:MEASure:PAM:ESKew (see page 1016)	Eyes for PAM-6 and PAM-8 are now supported.
:MEASure:PAM:LEVel (see page 1027)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:PAM:LRMS (see page 1029)	Levels for PAM-6 and PAM-8 signal types are now supported.

Command	Description
:MEASure:PAM:LTHickness (see page 1031)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:PAM:PRBS13q:EDGE:J6U? (see page 1040)	If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned (instead of just one result state).
:MEASure:RISetime (see page 1085)	With PAM6 and PAM8 signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.
:MEASure:RJDJ:ALL? (see page 1087)	Jitter analysis is now supported on PAM-6 and PAM-8 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.
:MEASure:RJDJ:PAMThreshold (see page 1101)	Threshold levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:RJDJ:TJRJDJ? (see page 1109)	Jitter analysis is now supported on PAM-6 and PAM-8 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.
:MEASure:THResholds:GENeral:PAMAutomatic (see page 1139)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:THResholds:GENeral:PAMCustom (see page 1137)	Threshold levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:THResholds:RFALl:PAMAutomatic (see page 1154)	Levels for PAM-6 and PAM-8 signal types are now supported.
:MEASure:TIEData2 (see page 1176)	The <threshold> parameter now supports PAM-6 and PAM-8 eyes.
:MTESt:FOLDing:BITS (see page 1251)	The <pattern> string can contain characters "2", "3", "4", or "5" when specified for PAM-6 signals or characters "2", "3", "4", "5", "6", or "7" when specified for PAM-8 signals.
:MTESt<N>:SOURce (see page 1300)	NONE can now be selected as the mask test source.
:TRIGger:MODE (see page 1572)	Available on MXR-Series oscilloscopes, the IFMagn option is added.

What's New in Version 11.15

New command descriptions for Version 11.15 of the Infiniium MXR/EXR-Series oscilloscope software appear below.

New Commands

Command	Description
:ANALyze:SIGNal:PATTern:INVert (see page 396)	Enables or disables inverting a PRBS or loaded pattern.
:ANALyze:SIGNal:PATTern:REVerse (see page 400)	Enables or disables reversing a PRBS or loaded pattern.
:FUNCtion:FFT:PEAK:SORT (see page 630)	Specifies the peak annotation sort order.
:FUNCtion<F>:FFT:PEAK:COUNt (see page 631)	Specifies the maximum number of peaks in the FFT to annotate.
:FUNCtion<F>:FFT:PEAK:FREQuency? (see page 632)	Returns a comma-separated string of annotated peak frequency values.
:FUNCtion<F>:FFT:PEAK:LEVel (see page 633)	Specifies the level above which peaks are identified.
:FUNCtion<F>:FFT:PEAK:MAGNitude? (see page 634)	Returns a comma-separated string of annotated peak magnitude values.
:FUNCtion<F>:FFT:PEAK:STATe (see page 635)	Enables or disables FFT peak annotations.
:FUNCtion<F>:LABel (see page 659)	Sets the math function waveform label to the quoted string.
:LANE<N>:EQUalizer:CTLE:P5 (see page 774)	Sets the Pole 5 frequency for the CTLE.
:LANE<N>:EQUalizer:CTLE:P6 (see page 775)	Sets the Pole 6 frequency for the CTLE.
:MEASure:NSIGma (see page 1004)	Installs, or returns the value of, a Sigma-n measurement on a PAM-4 waveform.
:MEASure:PAM:PRBS13q:EDGE:J6U? (see page 1040)	When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled, this query returns the measured PRBS13Q J6u values.
:MEASure:PAM:PRBS13q:PATTern (see page 1045)	Specifies the edge definition for the 12-edge jitter measurements' data pattern (either PRBS13Q, PCIE6 (52 symbols), or a custom data pattern).
:MEASure:PAM:PRBS13q:PFILe (see page 1046)	When a custom data pattern is being used, this command specifies the full-path location of the edge definition file for that pattern.

Command	Description
:MEASure:PN:INFO? (see page 1062)	Returns information about the number of averages and number of correlations that have occurred in the phase noise analysis.
:WMEMory<R>:LABel (see page 1772)	Sets the waveform memory label to the quoted string.

Changed Commands

Command	Description
:ANALyze:CLOCk:METHod (see page 334)	Added the "PCIE6,PCIE8", "PCIE6,PCIE16", and "PCIE6,PCIE32" PCIe 6 CXL Behavioral SRIS CC clock recovery methods for 8 Gb/s, 16 Gb/s, and 32 Gb/s data rates, respectively.
:BUS:B<N>:TYPE (see page 410)	Added support for new E1000BASET1 (Automotive Ethernet 1000BaseT1) and DPAUX (DisplayPort Aux) protocols.
:WGEN:PRBS:LENGth (see page 1761)	The options for this command have changed to P7M1, P15M1, P23M1, and P31M1 for PRBS pattern lengths 2^7-1, 2^15-1, 2^23-1, and 2^31-1, respectively.

What's New in Version 11.10

New command descriptions for Version 11.10 of the Infiniium MXR/EXR-Series oscilloscope software appear below.

New Commands

Command	Description
:ANALyze:CLOCk:METHod:PLLadvanced (see page 347)	Enables or disables the "Advanced PLL for closed eyes" option.
:CHANnel<N>:SPECtral:CFRequency:TESTLIMITS? (see page 505)	Returns the center frequency minimum and maximum limits.
:CHANnel<N>:SPECtral:SPAN:TESTLIMITS? (see page 507)	Returns the frequency span minimum and maximum limits.
:DISK:SAVE:MREPort (see page 542)	Lets you save information about the oscilloscope setup, a waveform screen capture, and measurement results to a PDF or MHTML (*.mht) format file.
:DISK:SAVE:NOISe (see page 543)	Saves the noise measurements, shown in the Noise Results tab at the bottom of the oscilloscope screen, along with the Noise graph data in a comma separated variables (CSV) file format
:DISPlay:CGRade:LEGend (see page 559)	Lets you enable or disable the Color Grade legend in the graphical user interface's Results pane.
:DISPlay:GRATicule:GLAYout (see page 571)	Specifies the layout format to be used in organizing the grids in a waveform area.
:DISPlay:THEMe (see page 602)	Lets you select the graphical user interface's "Midnight" or "Platinum" theme.
:FUNCtion<F>:FFT:IMPedance (see page 628)	When the FFT vertical units are displayed (and measured) as power (that is, dBm or Watt), this command lets you specify the reference impedance of the waveform source so that power is calculated correctly.
:LANE<N>:EQUalizer:CTLE:DBACgain (see page 762)	Sets the AC Gain parameter in dB for the CTLE when USB31 is selected for the "# of Poles" option
:LANE<N>:EQUalizer:CTLE:DBDCG2 (see page 763)	Sets the DC Gain 2 parameter in dB when "2 Pole, 2 Gain" is selected for the "# of Poles" option.
:LANE<N>:EQUalizer:CTLE:DBDCgain (see page 764)	Sets the DC Gain parameter in dB for the CTLE.
:LANE<N>:EQUalizer:CTLE:DCGain2 (see page 766)	Sets the DC Gain 2 parameter when "2 Pole, 2 Gain" is selected for the "# of Poles" option.
:LANE<N>:EQUalizer:CTLE:LF (see page 767)	Sets the LF Frequency parameter when "2 Pole, 2 Gain" is selected for the "# of Poles" option.
:LANE<N>:EQUalizer:DFE:TAP:ALGorithm (see page 783)	Sets the DFE tap optimization algorithm.

Command	Description
:MARKer<K>:CMODe (see page 856)	Specifies a particular marker's color mode.
:MARKer<K>:COLor (see page 857)	Gives the marker a custom color when in CUSTom color mode.
:MEASure:HISTogram:MM3S (see page 958)	Installs a histogram measurement, or returns the measurement value, of the mean minus three standard deviations.
:MEASure:HISTogram:MP3S (see page 960)	Installs a histogram measurement, or returns the measurement value, of the mean plus three standard deviations.
:MEASure:OERatio (see page 1008)	Installs, or returns the value of, an outer Extinction Ratio measurement on a PAM-4 signal, through an optical probe, that has a pattern with at least seven consecutive 3s and six consecutive 0s.
:MEASure:OOMA (see page 1010)	Installs, or returns the value of, an outer OMA (Optical Modulation Amplitude) measurement on a PAM-4 signal, through an optical probe, that has a pattern with at least seven consecutive 3s and six consecutive 0s.
:MEASure:PAM:EYE:VEC (see page 1023)	Installs, or returns the value of, a VEC (Vertical Eye Closure) measurement on a PAM-4 eye.
:MEASure:THResholds:SERauto (see page 1161)	For protocol decodes that do not use clock recovery, this command automatically sets the general "Custom: thresholds (low, mid, up)" or "Custom: thresholds +/- hysteresis" when thresholds apply to individual waveforms.
:MEASure:TIEFilter:DAMPing (see page 1178)	Specifies the damping factory for a second order low-pass TIE filter.
:MEASurement<N>:POSition (see page 1219)	Lets you reorder measurements within the Measurements window in the Results pane
:SBUS<N>:SEARch:ENABle (see page 1429)	Enables or disables protocol search.
:SBUS<N>:SEARch:TRIGger (see page 1430)	Enables or disables protocol search being used as a software trigger.
:SBUS<N>:UART:BAUDrate (see page 1504)	Selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode.
:SBUS<N>:UART:BITorder (see page 1505)	Selects whether the most significant bit (MSB) or least significant bit (LSB) is presented after the start bit in the signal from your device under test.
:SBUS<N>:UART:DIRection (see page 1506)	Specifies whether you are decoding Rx only (RX), Tx only (TX), or Rx and Tx (RXTX).
:SBUS<N>:UART:EOF:HEX (see page 1507)	Specifies the End-Of-Frame hexadecimal byte value.

Command	Description
:SBUS<N>:UART:IDLE (see page 1508)	Selects LOW or HIGH to match the device under test's state when at idle.
:SBUS<N>:UART:PARity (see page 1509)	Specifies the type of parity being used.
:SBUS<N>:UART:SOURce:RX (see page 1510)	Specifies the receiver signal source.
:SBUS<N>:UART:SOURce:TX (see page 1511)	Specifies the transmitter signal source.
:SBUS<N>:UART:WIDTh (see page 1512)	Sets the number of bits in the RS-232/UART words to match your device under test.
:SYSTem:DIMPedance (see page 1524)	Sets the **Default Impedance to 1M Ohm** user preference.
:WAVeform:PNOise:FREQuency? (see page 1701)	Returns the horizontal frequency axis values for the phase noise analysis results waveform.
:WAVeform:SEGMented:POINts? (see page 1709)	Returns the number of points in the segmented memory data.
:WGEN:PRBS:LENGth (see page 1761)	When the PRBS waveform is selected, this command specifies the bit length of the PRBS pattern (2^7, 2^15, 2^23, or 2^31 bits).

Changed Commands

Command	Description
:ANALyze:CLOCk:METHod (see page 334)	Added the "PCIE5,PCIE8", "PCIE5,PCIE16", and "PCIE5,PCIE32" PCIe 5 CXL Behavioral SRIS CC clock recovery methods for 8 Gb/s, 16 Gb/s, and 32 Gb/s data rates, respectively.
:ANALyze:SIGNal:PATTern:PLENgth (see page 398)	There are now additional P5M1, P6M1, P7M1, P8M1, P9M1, P10M1, P11M1, P12M1, P13M1, P14M1, and P15M1 options for specifying PRBS pattern lengths.
:CHANnel<N>:ISIM:BWLimit:TYPE (see page 457)	The BUTTerworth option has been added.
:CHANnel<N>:PROBe:INFO? (see page 489)	Added a field at the end of the query results for probe bandwidth.
:LANE<N>:EQUalizer:CTLE:NUMPoles (see page 768)	Added the LFG2 (2 Pole, 2 Gain) option for the "number of poles" selection.

Command	Description
:LANE<N>:EQUalizer:DFE:TAP:MAX (see page 791)	The max or min tap value can now be specified for individual taps, and the query can return individual tap values.
:LANE<N>:EQUalizer:DFE:TAP:MAXV (see page 792)	
:LANE<N>:EQUalizer:DFE:TAP:MIN (see page 793)	
:LANE<N>:EQUalizer:DFE:TAP:MINV (see page 794)	
:BUS:B<N>:TYPE (see page 410)	Added support for new PCI5 (PCI Express Gen 1-5), USB4, USB4LS (USB4 Low Speed), USB4TUSB32 (USB4 Tunnel USB 3.2), and USB4TPCI31 (USB4 Tunnel PCI Express 3.1) protocols.
:MEASure:CGRade:EWIDth (see page 892)	Added the ability to select units in UNITinterval or SECond (which was the default before).
:MEASure:CGRade:JITTer (see page 897)	Added the ability to select units in UNITinterval or SECond (which was the default before).
:MEASure:PAM:ESKew (see page 1016)	Added the ability to select units in UNITinterval or SECond. This control has been in the graphical user interface and is now available in the remote SCPI command.
:MEASure:THResholds:RFALl:METHod (see page 1152)	The T1090 (10% and 90% of levels) and T2080 (20% and 80% of levels) settings for rise/fall measurement thresholds now work for NRZ and UNSPecified signal types as well as PAM signal types.
:MEASure:TIEFilter:SHAPe (see page 1179)	Added the FIRSt and SECond options for the new First Order and Second Order TIE filter shapes. Added the DB60 option for the new 60dB/Decade TIE filter shape.
:WAVeform:DATA (see page 1675)	With Infiniium Offline only, this command copies the waveform points in an IEEE data block to the channel source specified by the :WAVeform:SOURce command.
:WAVeform:SEGMented:XLISt? (see page 1711)	Added the OFFSet option for getting the offset for each segment (within the returned data) when :WAVeform:SEGMented:ALL is ON.

Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:MTESt:FOLDing:FAST	none	This option is no longer available.
:MTESt:TRIGger:SOURce	none	Normal :TRIGger commands should be used to set up triggers when mask testing.

What's New in Version 11.06

The 11.06 software release supports the new Infiniium EXR-Series oscilloscopes. The EXR-Series oscilloscopes are similar to the MXR-Series oscilloscopes except that:

- There is no RTSA (Real-Time Spectrum Analysis) mode. (The **[RTSA]** front panel key is replaced by the **[Fault Hunter]** key for the Fault Hunter analysis feature.)
- There is no Spectral Analysis (DDC) signal type.
- There is no Frequency Extension signal type.
- The highest bandwidth models are 2.5 GHz.
- Standard memory depth is 100 Mpts per channel.

What's New in Version 11.05

New command descriptions for Version 11.05 of the Infiniium MXR-Series oscilloscope software appear below.

New Commands

Command	Description
:SYSTem:MODE (see page 1534)	Selects between the OSCilloscope or RTSA (Real-Time Spectrum Analysis) mode or returns the current selection.
:WGEN:ARBitrary:IMPort (see page 1733)	Imports arbitrary waveform points from a previously-saved comma-separated value (CSV) arbitrary waveform points file.
:WGEN:ARBitrary:INTerpolate (see page 1734)	Specifies how lines are drawn between arbitrary waveform points.

Changed Commands

Command	Description
:ANALyze:SIGNal:TYPE (see page 404)	SPECtral and FEXTension have been added as signal types.
:CHANnel<N>:SPECtral:CFRequency (see page 504)	Sets the center frequency with the Frequency Extension, Spectral Analysis (DDC), and Real-Time Spectrum Analysis (RTSA) features.
:CHANnel<N>:SPECtral:SPAN (see page 506)	Sets the frequency span with the Frequency Extension, Spectral Analysis (DDC), and Real-Time Spectrum Analysis (RTSA) features.
:WAVeform:DATA? (see page 1675)	When SPECtral is selected as the signal type, this query returns ASCII format data in complex IQ pairs in the following order: <Real>,<Imaginary>,<Real>,<Imaginary>,...
:WGEN:FUNCtion (see page 1736)	ARBitrary has been added as a waveform generator function.

Version 11.00 at Introduction

The Keysight Infiniium MXR-Series oscilloscopes were introduced with version 11.00 of oscilloscope operating software.

The command set is most closely related to version 10.20 of the Infiniium UXR-Series oscilloscope operating software and, with respect to digital channels, version 6.50 of the Infiniium S-Series oscilloscope operating software.

For more information, see "Command Differences From Previous Infiniium Oscilloscopes" on page 70.

Command Differences From Previous Infiniium Oscilloscopes

The Keysight MXR-Series oscilloscopes command set is most closely related to version 10.20 of the Infiniium UXR-Series oscilloscope operating software and, with respect to digital channels, version 6.50 of the Infiniium S-Series oscilloscope operating software.

The main differences between the version 10.20 programming command set for the Infiniium UXR-Series oscilloscopes and the 11.00 programming command set for the Infiniium MXR-Series oscilloscopes are related to:

- Up to 8 analog input channels (with triggering limitations on channels 5 through 8).
- New digital voltmeter (DVM), counter, and waveform generator features.
- New High Resolution Oscilloscope (HRO) selections.
- New history of acquisitions after running acquisitions are stopped.
- New Power Analysis feature.
- Roll mode is supported on the MXR-Series oscilloscope models.
- Up to eight analog channel waveform memories are now available.
- Digital channels are supported on the MXR-Series oscilloscope models.
- Mask testing on up to 8 signals at the same time is supported.

With respect to digital channels, the main differences between the version 6.50 programming command set for the Infiniium S-Series oscilloscopes and the 11.00 programming command set for the Infiniium MXR-Series oscilloscopes are related to:

- The ability to set digital channel thresholds per nibble (4 bits) instead of byte-sized pods (8 bits).

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

New Commands

Command	Description
:ACQuire:ADCRes (see page 292)	The MXR-Series oscilloscopes provide High Resolution Oscilloscope (HRO) selections that work differently than the High Resolution acquisition mode in earlier Infiniium oscilloscopes.
:ACQuire:HISTory:COUNt? (see page 304)	Returns the number of acquisitions in the history (after running acquisitions are stopped).
:ACQuire:HISTory:INDex (see page 305)	Navigates to the specified acquisition in the history or returns the currently displayed acquisition history.
:ACQuire:HISTory:PLAY (see page 306)	Turns the acquisition history play setting on or off, or returns the acquisition history play status.

Command	Description
:ACQuire:MODE (see page 308)	In the MXR-Series oscilloscopes, ROLL mode is available and SEGHres is no longer available.
:ACQuire:POINts:DIGital? (see page 313)	Returns the current memory depth for the digital channels.
:ACQuire:SRATe:DIGital (see page 325)	Sets the digital acquisition sampling rate.
:ACQuire:SRATe:DIGital:AUTO (see page 326)	Enables (ON) or disables (OFF) the automatic digital channel sampling rate selection control.
:BUS<B> commands (see page 409)	Commands for setting up digital channel buses.
:COUNter:RATio (see page 510)	Enables or disables the "show Counter A to Counter B ratio" feature.
:COUNter:RATio:CURRent? (see page 511)	Returns the current Counter A to Counter B ratio value.
:COUNter<N>:CURRent? (see page 512)	Returns the current counter value.
:COUNter<N>:ENABle (see page 513)	Enables or disables the counter feature.
:COUNter<N>:MODE (see page 514)	Sets the counter mode.
:COUNter<N>:NDIGits (see page 515)	Sets the number of digits of resolution used for the frequency or period counter.
:COUNter<N>:SOURce (see page 516)	Selects the waveform source that the counter measures.
:COUNter<N>:TOTalize:CLEar (see page 517)	Zeros the edge event counter.
:COUNter<N>:TOTalize:SLOPe (see page 518)	Specifies whether positive or negative edges are counted.
:DIGital<N> commands (see page 519)	Commands for digital input channels.
:DISable DIGital (see page 266)	Disables the digital channels 0-15.
:DVM:ARANge (see page 606)	If the selected digital voltmeter (DVM) source channel is not used in oscilloscope triggering, this command turns the digital voltmeter's Auto Range capability on or off.
:DVM:CURRent? (see page 607)	Returns the displayed DVM value.
:DVM:ENABle (see page 608)	Turns the digital voltmeter (DVM) analysis feature on or off.

Command	Description
:DVM:MODE (see page 609)	Sets the digital voltmeter (DVM) mode.
:DVM:SOURce (see page 610)	Selects the analog channel on which digital voltmeter (DVM) measurements are made.
:ENABle DIGital (see page 267)	Enables the digital channels 0-15.
:MEASure:ANGLe (see page 1224)	Installs a power phase angle measurement on screen or returns the measured power phase angle in degrees.
:MEASure:APParent (see page 1225)	Installs an apparent power measurement on screen or returns the measured apparent power.
:MEASure:CPLoss (see page 1226)	Installs a power loss per cycle measurement on screen or returns the measured value.
:MEASure:CRESt (see page 1227)	Installs a crest factor measurement on screen or returns the measured crest factor.
:MEASure:EFFiciency (see page 1228)	Installs an efficiency (output power / input power) measurement on screen or returns the measured efficiency as a percent value.
:MEASure:ELOSs (see page 1229)	Installs an energy loss measurement on screen or returns the switching loss in joules.
:MEASure:FACTor (see page 1230)	Installs a power factor measurement on screen or returns the measured power factor.
:MEASure:IPOWer (see page 1231)	Installs an input power measurement on screen or returns the measured input power.
:MEASure:OFFTime (see page 1232)	Installs an "off time" measurement on screen or returns the measured turn off time.
:MEASure:ONTime (see page 1233)	Installs an "on time" measurement on screen or returns the measured turn off time.
:MEASure:OUTPower (see page 1234)	Installs an output power measurement on screen or returns the measured output power.
:MEASure:PCURrent (see page 1235)	Installs a peak current measurement on screen or returns the measured peak current.
:MEASure:PLOSs (see page 1236)	Installs a power loss measurement on screen or returns the switching loss in watts.
:MEASure:RDSon (see page 1237)	Installs an RDS(on) power measurement on screen or returns the measured value.
:MEASure:REACtive (see page 1238)	Installs a reactive power measurement on screen or returns the measured reactive power.
:MEASure:REAL (see page 1240)	Installs a real power measurement on screen or returns the measured real power.

Command	Description
:MEASure:RIPPle (see page 1241)	Installs an output ripple measurement on screen or returns the measured output ripple.
:MEASure:TRESponse (see page 1242)	Installs a transient response time measurement on screen or returns the measured transient response time.
:MEASure:VCESat (see page 1243)	Installs a VCE(sat) power measurement on screen or returns the measured value.
:MTESt:FENable (see page 1248)	Enables or disables the "Stop on Failure" option.
:MTESt:RUNNing? (see page 1265)	Returns whether the mask test is running.
:MTESt:RUMode:MOFailure (see page 1263)	Enables or disables the "Stop on Failure" option. Enables or disables the "Perform Multipurpose on Failure" run until option.
:POD<N> commands (see page 1303)	Commands for digital channel pods.
:POD<N>:NIBBle<N>:THReshold (see page 1305)	Sets the logic threshold value for a nibble within a pod.
:POWer:ANALysis (see page 1314)	Selects the type of power analysis.
:POWer:CLResponse:APPLy (see page 1315)	Performs the control loop response (Bode) analysis.
:POWer:CLResponse:DATA? (see page 1316)	Returns data from the Control Loop Response (Bode) power analysis.
:POWer:CLResponse:DATA:GMARgin? (see page 1317)	Returns the gain margin in dB.
:POWer:CLResponse:DATA:GMARgin:FREQuency? (see page 1318)	Returns the 0° phase crossover frequency in Hz.
:POWer:CLResponse:DATA:PMARgin? (see page 1319)	Returns the phase margin in degrees.
:POWer:CLResponse:DATA:PMARgin:FREQuency? (see page 1320)	Returns the 0 dB gain crossover frequency in Hz.
:POWer:CLResponse:FREQuency:MODE (see page 1321)	Specifies whether the analysis should be performed by sweeping through a range of frequencies (SWEep) or at a single frequency (SINGle).
:POWer:CLResponse:FREQuency:SINGle (see page 1322)	Sets the single frequency value.

Command	Description
:POWer:CLResponse:FREQuency:STARt (see page 1323)	Sets the frequency sweep start value.
:POWer:CLResponse:FREQuency:STOP (see page 1324)	Sets the frequency sweep stop value.
:POWer:CLResponse:SOURce:INPut (see page 1325)	Selects the oscilloscope channel that is probing the power supply input.
:POWer:CLResponse:SOURce:OUTPut (see page 1326)	Selects the oscilloscope channel that is probing the power supply output.
:POWer:CLResponse:SWEep:POINts (see page 1327)	Specifies the total number of points in the frequency response analysis.
:POWer:CLResponse:WGEN:LOAD (see page 1328)	Sets the waveform generator expected output load impedance.
:POWer:CLResponse:WGEN:VOLTage (see page 1329)	Sets the waveform generator output amplitude(s).
:POWer:CLResponse:WGEN:VOLTage:PROFile (see page 1330)	Enables or disables the ability to set initial waveform generator ramp amplitudes for each frequency range.
:POWer:EFFiciency:APPLy (see page 1331)	Applies the efficiency power analysis.
:POWer:EFFiciency:TYPE (see page 1332)	Specifies the type of power that is being converted from the input to the output.
:POWer:ENABle (see page 1333)	Enables or disables power analysis.
:POWer:HARMonics:APPLy (see page 1334)	Applies the current harmonics analysis.
:POWer:HARMonics:DATA? (see page 1335)	Returns the power harmonics results table data.
:POWer:HARMonics:DISPlay (see page 1336)	Specifies how to display the current harmonics analysis results.
:POWer:HARMonics:FAILcount? (see page 1337)	Returns the current harmonics analysis' fail count.
:POWer:HARMonics:LINE (see page 1338)	Specifies the line frequency setting for the current harmonics analysis.
:POWer:HARMonics:POWerfactor? (see page 1339)	Returns the power factor for IEC 61000-3-2 Standard Class C power factor value.
:POWer:HARMonics:RPOWer (see page 1340)	When Class D is selected as the current harmonics analysis standard, this command specifies whether the Real Power value used for the current-per-watt measurement is measured by the oscilloscope or is defined by the user.

Command	Description
:POWer:HARMonics:RPOWer:USER (see page 1341)	When Class D is selected as the current harmonics analysis standard and you have chosen to use a user-defined Real Power value, this command specifies the Real Power value used in the current-per-watt measurement.
:POWer:HARMonics:RUNCount? (see page 1342)	Returns the current harmonics analysis' run iteration count.
:POWer:HARMonics:STANdard (see page 1343)	Selects the standard to perform current harmonics compliance testing on.
:POWer:HARMonics:STATus? (see page 1344)	Returns the overall pass/fail status of the current harmonics analysis.
:POWer:HARMonics:THD? (see page 1345)	Returns the Total Harmonics Distortion (THD) results of the current harmonics analysis.
:POWer:INRush:APPLy (see page 1346)	Applies the inrush current analysis.
:POWer:INRush:NEXT (see page 1347)	Goes to the next step of the inrush current analysis.
:POWer:ITYPe (see page 1348)	Specifies the type of power that is being converted from the input (DC or AC).
:POWer:MODulation:APPLy (see page 1349)	Applies the selected modulation analysis type.
:POWer:MODulation:SOURce (see page 1350)	Selects either the voltage source or the current source as the source for the modulation analysis.
:POWer:MODulation:TYPE (see page 1351)	Selects the type of measurement to make in the modulation analysis.
:POWer:ONOFf:APPLy (see page 1352)	Applies the selected turn on/off analysis test.
:POWer:ONOFf:NEXT (see page 1353)	Goes to the next step of the turn on/turn off analysis.
:POWer:ONOFf:TEST (see page 1354)	Selects whether turn on or turn off analysis is performed.
:POWer:ONOFf:THResholds (see page 1355)	Specifies the input and output thresholds used in the Turn On/Turn Off analysis.
:POWer:PSRR:APPLy (see page 1356)	Applies the power supply rejection ratio (PSRR) analysis.
:POWer:PSRR:DATA? (see page 1357)	Returns data from the Power Supply Rejection Ratio (PSRR) power analysis.
:POWer:PSRR:FREQuency:MAXimum (see page 1358)	Sets the end sweep frequency value.

Command	Description
:POWer:PSRR:FREQuency:MINimum (see page 1359)	Sets the start sweep frequency value.
:POWer:PSRR:FREQuency:MODE (see page 1360)	specifies whether the analysis should be performed by sweeping through a range of frequencies (SWEep) or at a single frequency (SINGle).
:POWer:PSRR:FREQuency:SINGle (see page 1361)	Sets the single frequency value.
:POWer:PSRR:SOURce:INPut (see page 1362)	Selects the oscilloscope channel that is probing the power supply input.
:POWer:PSRR:SOURce:OUTPut (see page 1363)	Selects the oscilloscope channel that is probing the power supply output.
:POWer:PSRR:SWEep:POINts (see page 1364)	Specifies the total number of points in the frequency response analysis.
:POWer:PSRR:WGEN:LOAD (see page 1365)	Sets the waveform generator expected output load impedance.
:POWer:PSRR:WGEN:VOLTage (see page 1366)	Sets the waveform generator output amplitude(s).
:POWer:PSRR:WGEN:VOLTage:PROFile (see page 1367)	Enables or disables the ability to set initial waveform generator ramp amplitudes for each frequency range.
:POWer:QUALity:APPLy (see page 1368)	Applies the selected power quality analysis type.
:POWer:RIPPle:APPLy (see page 1369)	Applies the output ripple analysis.
:POWer:SIGNals:AUTosetup (see page 1370)	Performs automated oscilloscope setup for the signals in the specified type of power analysis.
:POWer:SIGNals:CYCLes:HARMonics (see page 1371)	Specifies the number of cycles to include in the current harmonics analysis.
:POWer:SIGNals:CYCLes:QUALity (see page 1372)	Specifies the number of cycles to include in the power quality analysis.
:POWer:SIGNals:DURation:EFFiciency (see page 1373)	Specifies the duration of the efficiency analysis.
:POWer:SIGNals:DURation:MODulation (see page 1374)	Specifies the duration of the modulation analysis.
:POWer:SIGNals:DURation:ONOFf:OFF (see page 1375)	Specifies the duration of the turn off analysis.
:POWer:SIGNals:DURation:ONOFf:ON (see page 1376)	Specifies the duration of the turn on analysis.
:POWer:SIGNals:DURation:RIPPle (see page 1377)	Specifies the duration of the output ripple analysis.

Command	Description
:POWer:SIGNals:DURation:TRANsient (see page 1378)	Specifies the duration of the transient response analysis.
:POWer:SIGNals:IEXPected (see page 1379)	Specifies the expected inrush current amplitude.
:POWer:SIGNals:OVERshoot (see page 1380)	Specifies the percent of overshoot of the output voltage.
:POWer:SIGNals:VMAXimum:INRush (see page 1382)	Specifies the maximum expected input voltage.
:POWer:SIGNals:VMAXimum:ONOFf:OFF (see page 1383)	Specifies the maximum expected input voltage.
:POWer:SIGNals:VMAXimum:ONOFf:ON (see page 1384)	Specifies the maximum expected input voltage.
:POWer:SIGNals:VSTeady:ONOFf:OFF (see page 1385)	Specifies the expected steady state output DC voltage of the power supply for turn off analysis.
:POWer:SIGNals:VSTeady:ONOFf:ON (see page 1386)	Specifies the expected steady state output DC voltage of the power supply for turn on analysis.
:POWer:SIGNals:VSTeady:TRANsient (see page 1387)	Specifies the expected steady state output DC voltage of the power supply for transient response analysis.
:POWer:SIGNals:SOURce:CURRent<I> (see page 1388)	Specifies the first, and perhaps second, current source channel to be used in the power analysis.
:POWer:SIGNals:SOURce:VOLTage<I> (see page 1389)	Specifies the first, and perhaps second, voltage source channel to be used in the power analysis.
:POWer:SLEW:APPLy (see page 1390)	Applies the slew rate analysis.
:POWer:SLEW:SOURce (see page 1391)	Selects either the voltage source or the current source as the source for the slew rate analysis.
:POWer:SWITch:APPLy (see page 1392)	Applies the switching loss analysis using the conduction calculation method, V reference, and I reference settings.
:POWer:SWITch:CONDuction (see page 1393)	Specifies the conduction calculation method.
:POWer:SWITch:IREFerence (see page 1394)	Specifies the current switching level for the start of switching edges.
:POWer:SWITch:RDS (see page 1395)	Specifies the Rds(on) value when the RDS conduction calculation method is chosen (by :POWer:SWITch:CONDuction).
:POWer:SWITch:VCE (see page 1396)	Specifies the Vce(sat) value when the VCE conduction calculation method is chosen (by :POWer:SWITch:CONDuction).
:POWer:SWITch:VREFerence (see page 1397)	Specifies the voltage switching level for the switching edges.

Command	Description
:POWer:TRANsient:APPLy (see page 1398)	Applies the transient analysis using the initial current and new current settings.
:POWer:TRANsient:IINitial (see page 1399)	Specifies the initial load current value.
:POWer:TRANsient:INEW (see page 1400)	Specifies the new load current value.
:POWer:TRANsient:NEXT (see page 1401)	Goes to the next step of the transient analysis.
:WGEN:FREQuency (see page 1735)	For all waveforms except Noise and DC, this command specifies the frequency of the waveform.
:WGEN:FUNCtion (see page 1736)	Selects the type of waveform.
:WGEN:FUNCtion:PULSe:WIDTh (see page 1740)	For Pulse waveforms, this command specifies the width of the pulse.
:WGEN:FUNCtion:RAMP:SYMMetry (see page 1741)	For Ramp waveforms, this command specifies the symmetry of the waveform.
:WGEN:FUNCtion:SQUare:DCYCle (see page 1742)	For Square waveforms, this command specifies the square wave duty cycle.
:WGEN:MODulation:AM:DEPTh (see page 1743)	Sets the portion of the amplitude range that will be used by the modulation.
:WGEN:MODulation:AM:FREQuency (see page 1744)	Specifies the frequency of the modulating signal.
:WGEN:MODulation:FM:DEViation (see page 1745)	Specifies the frequency deviation from the original carrier signal frequency.
:WGEN:MODulation:FM:FREQuency (see page 1746)	Specifies the frequency of the modulating signal.
:WGEN:MODulation:FSKey:FREQuency (see page 1747)	Specifies the "hop frequency".
:WGEN:MODulation:FSKey:RATE (see page 1748)	Specifies the rate at which the output frequency "shifts".
:WGEN:MODulation:FUNCtion (see page 1749)	Specifies the shape of the modulating signal.
:WGEN:MODulation:FUNCtion:RAMP:SYMMetry (see page 1750)	Specifies the amount of time per cycle that the ramp waveform is rising.
:WGEN:MODulation:NOISe (see page 1751)	Adds noise to the currently selected signal.
:WGEN:MODulation:STATe (see page 1752)	Enables or disables modulated waveform generator output.

Command	Description
:WGEN:MODulation:TYPE (see page 1753)	Selects the modulation type.
:WGEN:OUTPut (see page 1755)	Specifies whether the waveform generator signal output is ON (1) or OFF (0).
:WGEN:OUTPut:LOAD (see page 1756)	Selects the expected output load impedance.
:WGEN:OUTPut:MODE (see page 1757)	Specifies whether the defined waveform is output continuously or as a single cycle (single-shot).
:WGEN:OUTPut:POLarity (see page 1758)	Specifies whether the waveform generator output is inverted.
:WGEN:OUTPut:SINGle (see page 1759)	When the single-shot output mode is selected, this command causes a single cycle of the defined waveform to be output.
:WGEN:PERiod (see page 1760)	For all waveforms except Noise and DC, this command specifies the period of the waveform.
:WGEN:RST (see page 1762)	Restores the waveform generator factory default settings.
:WGEN:VOLTage (see page 1763)	Specifies the waveform's amplitude.
:WGEN:VOLTage:HIGH (see page 1764)	Specifies the waveform's high-level voltage.
:WGEN:VOLTage:LOW (see page 1765)	Specifies the waveform's low-level voltage.
:WGEN:VOLTage:OFFSet (see page 1766)	Specifies the waveform's offset voltage or the DC level.

Changed Commands

Command	Differences From Version 10.20 and 6.50 Infiniium Oscilloscopes
:ACQuire:HRESolution (see page 1835)	The MXR-Series oscilloscopes provide High Resolution Oscilloscope (HRO) selections that work differently than the High Resolution acquisition mode in earlier Infiniium oscilloscopes.
:ACQuire:MODE (see page 308)	In the MXR-Series oscilloscopes, ROLL mode is available and SEGHres is no longer available since acquisition modes and high-resolution oscilloscope modes are now separate.
:BLANk (see page 262)	The BUS<B>, DIGital<M>, and POD<P> parameters are available because digital channels are supported on the MXR-Series oscilloscopes.
:CALibrate:OUTPut (see page 425)	The MXR-Series oscilloscopes have updated output options.
:DIGitize (see page 264)	The DIGital<M> and POD<P> parameters are available because digital channels are supported on the MXR-Series oscilloscopes.

Command	Differences From Version 10.20 and 6.50 Infiniium Oscilloscopes
:MTESt<N>:AMASk:CREate (see page 1270)	Now applies to one of 8 masks. MTESt is equivalent to MTESt1.
:MTESt<N>:AMASk:SAVE (see page 1271)	
:MTESt<N>:AMASk:SOURce (see page 1272)	
:MTESt<N>:AMASk:UNITs (see page 1273)	
:MTESt<N>:AMASk:XDELta (see page 1274)	
:MTESt<N>:AMASk:YDELta (see page 1275)	
:MTESt<N>:COUNt:FAILures? (see page 1276)	
:MTESt<N>:COUNt:FUI? (see page 1277)	
:MTESt<N>:COUNt:FWAVeforms? (see page 1278)	
:MTESt<N>:COUNt:MARGin:FAILures? (see page 1279)	
:MTESt<N>:COUNt:SUI? (see page 1280)	
:MTESt<N>:COUNt:UI? (see page 1281)	
:MTESt<N>:COUNt:WAVeforms? (see page 1282)	
:MTESt<N>:DELete (see page 1283)	
:MTESt<N>:ENABle (see page 1284)	
:MTESt<N>:INVert (see page 1285)	

Command	Differences From Version 10.20 and 6.50 Infiniium Oscilloscopes
:MTESt<N>:LOAD (see page 1286)	Now applies to one of 8 masks. MTESt is equivalent to MTESt1.
:MTESt<N>:MARGin:AUTO:HITS (see page 1287)	
:MTESt<N>:MARGin:AUTO:HRATio (see page 1288)	
:MTESt<N>:MARGin:AUTO:METHod (see page 1289)	
:MTESt<N>:MARGin:METHod (see page 1290)	
:MTESt<N>:MARGin:PERCent (see page 1291)	
:MTESt<N>:MARGin:STATe (see page 1292)	
:MTESt<N>:NREGions? (see page 1293)	
:MTESt<N>:SCALe:BIND (see page 1294)	
:MTESt<N>:SCALe:DRAW (see page 1295)	
:MTESt<N>:SCALe:X1 (see page 1296)	
:MTESt<N>:SCALe:XDELta (see page 1297)	
:MTESt<N>:SCALe:Y1 (see page 1298)	
:MTESt<N>:SCALe:Y2 (see page 1299)	
:MTESt<N>:SOURce (see page 1300)	
:MTESt<N>:TITLe? (see page 1301)	
:OVLRegister (see page 273)	The Overload Event Register is widened to 16 bits, making room for up to 8 analog input channels and one additional Probe Power Fault bit.
:STATus? (see page 281)	The BUS<B>, DIGital<M>, and POD<P> parameters are available because digital channels are supported on the MXR-Series oscilloscopes.

Command	Differences From Version 10.20 and 6.50 Infiniium Oscilloscopes
:TIMebase:REFClock (see page 1544)	The HFRequency option is not supported on Keysight MXR-Series oscilloscopes.
:TIMebase:ROLL:ENABle (see page 1548)	The ON (1) option is available (in addition to OFF) because the roll mode sampling mode is available in the Keysight MXR-Series oscilloscopes.
:TRIGger:EDGE:SOURce (see page 1594)	The DIGital<M> and LINE parameters are supported on the MXR-Series oscilloscopes.
:TRIGger:HIGH (see page 1562)	OFF (0) is the only option available because high-bandwidth trigger is not needed with the Keysight MXR-Series oscilloscopes.
:VIEW (see page 287)	The BUS<B>, DIGital<M>, and POD<P> parameters are available because digital channels are supported on the MXR-Series oscilloscope models.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:ACQuire:HRESolution (see page 1835)	:ACQuire:ADCRes (see page 292)	The :ACQuire:HRESolution is for backwards compatibility. It does not provide access to all the High Resolution Oscilloscope (HRO) settings.
:MTESt:AVERage (see page 1854)		
:MTESt:AVERage:COUNt (see page 1855)		
:MTESt:STIMe (see page 1858)		
:MTESt<N>:ALIGn (see page 1859)		
:MTESt<N>:AUTO (see page 1860)		

Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:MTESt:IMPedance	none	
:MTESt:PROBe:IMPedance?	none	

2 Setting Up

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

Note that Keysight IO Libraries Suite software comes installed on Infiniium oscilloscopes, and it is possible to control the oscilloscope from programs running on the instrument.

NOTE

Secure Instrument Communication, where participants are authenticated and communication is encrypted, is now supported by the Infiniium oscilloscope software. A recent version of Keysight IO Libraries Suite is required. For more information, see: www.keysight.com/find/secureiousersguide

Also required is the Keysight Secure Instrument Communication (SIC) Expert software running on a central server. You can download the SIC Expert installation file from: www.keysight.com/find/iosuite

For information on setting up the Keysight Secure Instrument Communication (SIC) Expert configuration tool, click the Keysight "IO" Control icon in the taskbar and choose **Documentation > Secure Instrument Communication Expert Setup Guide.**

KEYSIGHT TECHNOLOGIES

Step 1. Install Keysight IO Libraries Suite software

To install the Keysight IO Libraries Suite software on a controller PC:

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.

Note that Keysight IO Libraries Suite software comes installed on Infiniium oscilloscopes.

Step 2. Connect and set up the oscilloscope

Infiniium oscilloscopes can have these interfaces for programming the oscilloscope:

- USB (device port, square connector).
- LAN. To configure the LAN interface, set up the Infiniium oscilloscope on the network as you would any other computer with the Windows operating system.
- GPIB, when the instrument has a GPIB interface connector or when the N4865A GPIB-to-LAN adapter is used.

When installed, these interfaces are always active.

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.

   Some oscilloscopes have a USB 2.0 high-speed port; other more recent models have a USB 3.0 super-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.

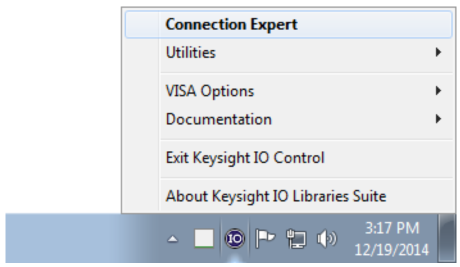
   Setting up an Infiniium oscilloscope on a network is the same as setting up any other computer with the Windows 10 operating system.

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

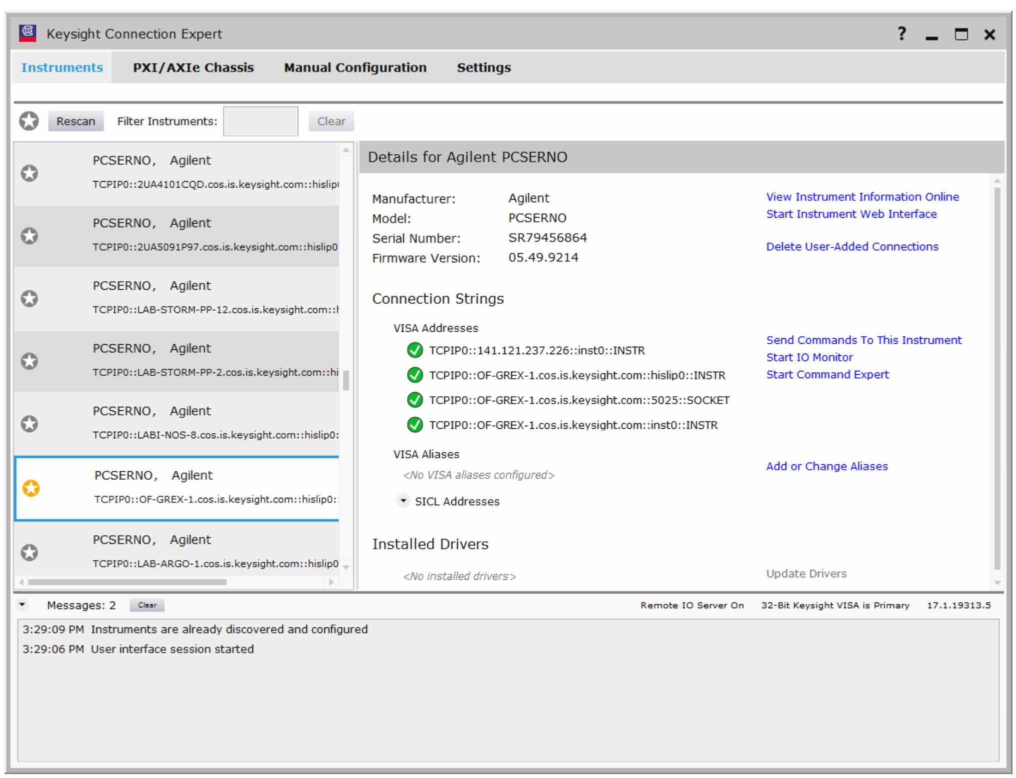
Step 3. Verify the oscilloscope connection

NOTE Make sure the Keysight Infiniium software is running on the oscilloscope. It must be running before you can make a connection.

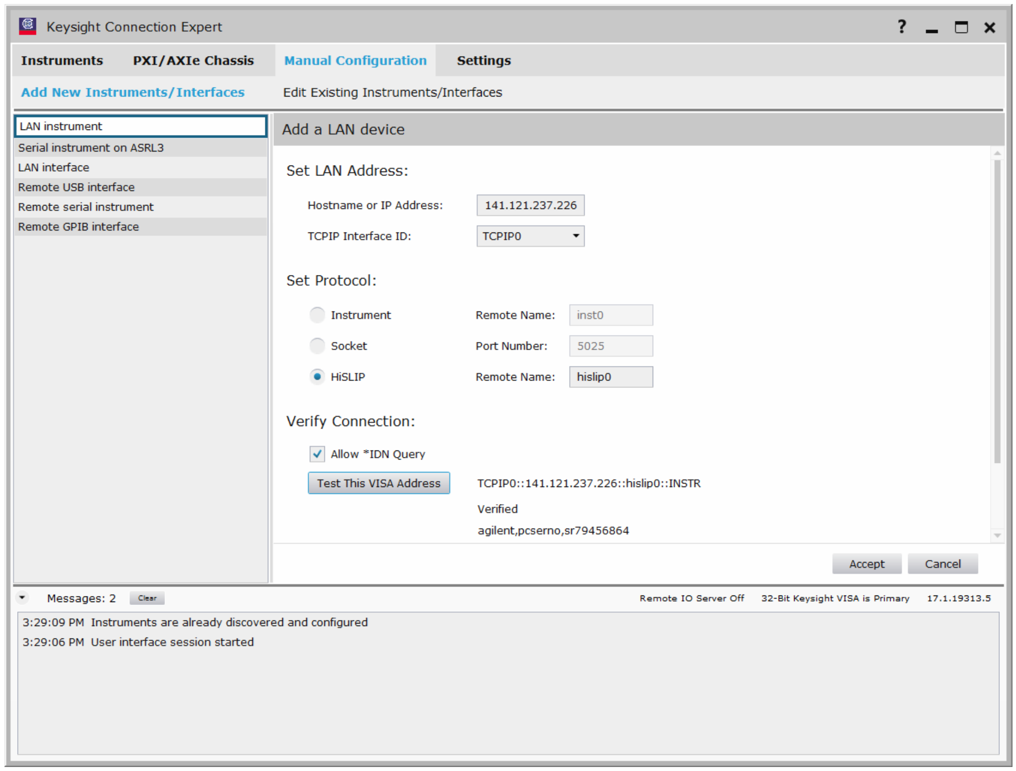
1 On the controller PC, click on the Keysight IO Control icon in the taskbar and choose **Connection Expert** from the popup menu.



2 In the Keysight Connection Expert application, instruments connected to the controller's USB and GPIB interfaces 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 **HiSLIP** under Set Protocol.

HiSLIP (High-Speed LAN Instrument Protocol) is a protocol for TCP-based instrument control that provides the instrument-like capabilities of conventional test and measurement protocols with minimal impact to performance.

For more information on the HiSLIP protocol, see:

- The Keysight IO Libraries Suite documentation.
- http://www.lxistandard.org/About/LXI-Device-Support-HiSLIP.aspx
- http://www.ivifoundation.org/specifications/

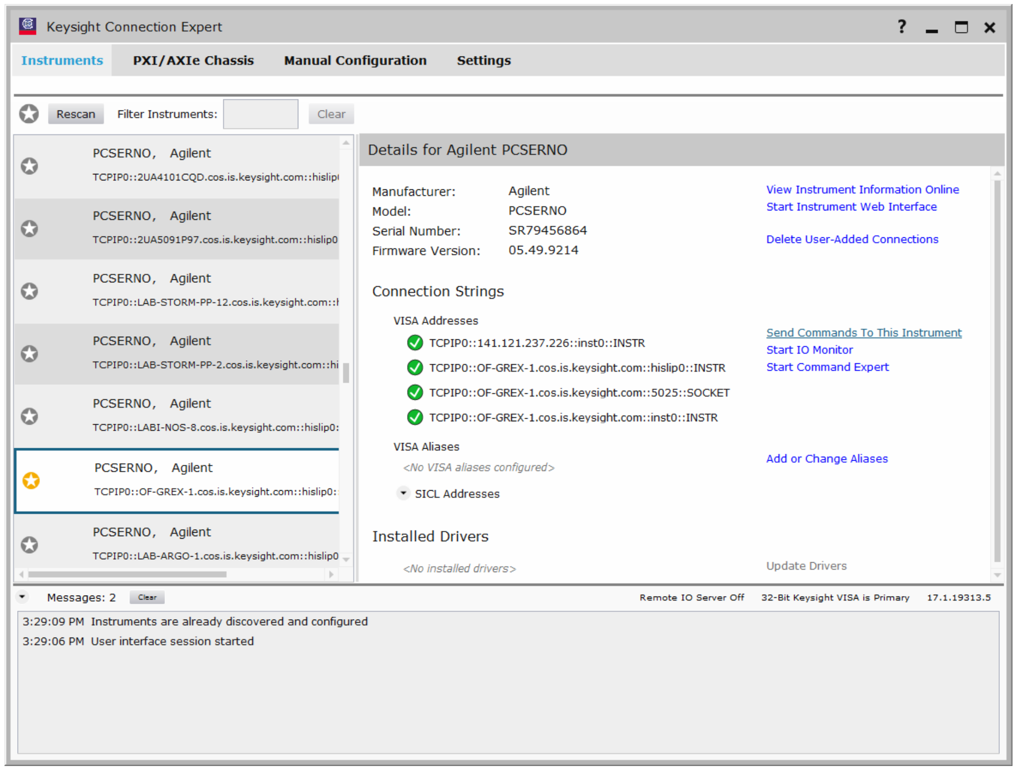
e Click **Test This VISA Address** to verify the connection.

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

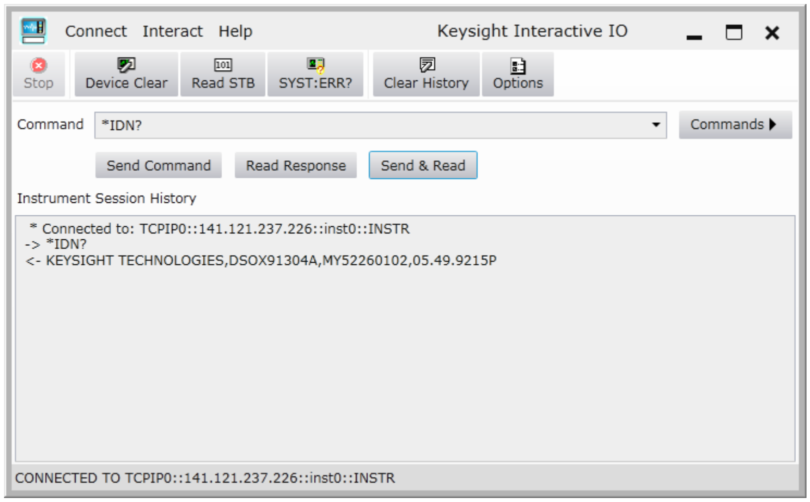
If the connection test is not successful, go back and verify the LAN connections and the oscilloscope setup.

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 Introduction to Programming

KEYSIGHT
TECHNOLOGIES

This chapter introduces the basics for remote programming of an oscilloscope. The programming commands in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming commands provide the means of remote control.

Basic operations that you can do with a computer and an oscilloscope include:

- Set up the oscilloscope.
- Make measurements.
- Get data (waveform, measurements, and configuration) from the oscilloscope.
- Send information, such as waveforms and configurations, to the oscilloscope.

You can accomplish other tasks by combining these functions.

NOTE

Example Programs are Written in Visual Basic for Applications (VBA) and C

The programming examples for individual commands in this manual are written in Visual Basic for Applications (VBA) and C.

Communicating with the Oscilloscope

Computers communicate with the oscilloscope by sending and receiving messages over a remote interface, such as a GPIB card (must order the N4865A GPIB-to-LAN adapter) or a Local Area Network (LAN) card. Commands for programming normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your computer. The input commands of the host language are used to read responses from the oscilloscope.

For example, the VISA COM library provides the WriteString() method for sending commands and queries. After a query is sent, the response can be read using the ReadString() method. The ReadString() method passes the value across the bus to the computer and places it in the designated variable.

For the GPIB interface, messages are placed on the bus using an output command and passing the device address, program message, and a terminator. Passing the device address ensures that the program message is sent to the correct GPIB interface and GPIB device.

The following WriteString() method sends a command that sets the channel 1 scale value to 500 mV:

```
myScope.WriteString ":CHANnel1:SCALe 500E-3"
```

The VISA COM library setup is explained on the following pages.

NOTE

Use the Suffix Multiplier Instead

Using "mV" or "V" following the numeric voltage value in some commands will cause Error 138 - Suffix not allowed. Instead, use the convention for the suffix multiplier as described in Chapter 6, “Message Communication and System Functions,” starting on page 147.

Instructions

Instructions, both commands and queries, normally appear as strings embedded in a statement of your host language, such as Visual Basic for Applications (VBA), Visual Basic .NET, C#, C, etc.

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 with the :SYSTem:SETup command. There are only a few instructions that use block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provides additional information to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more command mnemonics separated by colons (:). They represent the operation to be performed by the oscilloscope. See Chapter 4, “Programming Conventions,” starting on page 127 for more information.

Queries are formed by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you include the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

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. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 in decimal.

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.

Ellipsis

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

Square Brackets

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

Command and Query Sources

Many commands and queries require that a source be specified. Depending on the command or query and the model number of Infiniium oscilloscope being used, some of the sources are not available. The following is a list of sources:

CHANnel1	FUNCtion1	WMEMory1	COMMonmode{3|4}
CHANnel2	FUNCtion2	WMEMory2	DIFFerential{1|2}
CHANnel3	FUNCtion3	WMEMory3	EQUalized{1|2|3|4}
CHANnel4	FUNCtion4	WMEMory4	DIGital0 - DIGital15
CLOCk	MTRend	MSPectrum	HISTogram

Program Data

Program data is used to clarify the meaning of the command or query. It provides 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 and the values they accept.

When there is more than one data parameter, they are separated by commas (,). You can add spaces around the commas to improve readability.

Header Types

There are three types of headers:

- "Simple Command Header" on page 102
- "Compound Command Header" on page 102
- "Common Command Header" on page 103

See Also

- "Combining Commands in the Same Subsystem" on page 103
- "Duplicate Mnemonics" on page 103

Simple Command Header

Simple command headers contain a single mnemonic. AUToscale and DIGitize are examples of simple command headers typically used in this oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

For example:

```
":AUToscale"
```

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>
```

For example:

```
":DIGitize CHANnel1,FUNCtion2"
```

Compound Command Header

Compound command headers are a combination of two 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"
```

Combining Commands in the Same Subsystem

To execute more than one command within the same subsystem, use a semi-colon (;) to separate the commands:

```
:<subsystem>:<command><separator><data>;<command><separator>
<data><terminator>
```

For example:

```
:CHANnel1:INPut DC;BWLimit ON
```

Common Command Header

Common command headers, such as clear status, control the IEEE 488.2 functions within the oscilloscope. The 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.

Duplicate Mnemonics

Identical function mnemonics can be used for more than one subsystem. For example, you can use the function mnemonic RANGE to change both the vertical range and horizontal range:

To set the vertical range of channel 1 to 0.4 volts full scale:

```
:CHANnel1:RANGe .4
```

To set the horizontal time base to 1 second full scale:

```
:TIMebase:RANGe  1
```

In these examples, CHANnel1 and TIMebase are subsystem selectors, and determine the range type being modified.

Query Headers

A command header immediately followed by a question mark (?) is a query. After receiving a query, the oscilloscope interrogates the requested subsystem and places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a computer).

For example, with VISA COM library and Visual Basic for Applications (VBA) language, the query:

```
myScope.WriteString ":TIMebase:RANGe?"
```

places the current time base setting in the output queue.

The computer input statement:

```
varRange = myScope.ReadNumber
```

passes the value across the bus to the computer and places it in the variable varRange.

You can use queries to find out how the oscilloscope is currently configured and to get results of measurements made by the oscilloscope. For example, the query:

```
:MEASure:RISetime?
```

tells the oscilloscope to measure the rise time of your waveform and place the result in the output queue.

The output queue must be read before the next program message is sent. For example, when you send the query :MEASure:RISetime?, you must follow it with an input statement.

With the VISA COM library and Visual Basic for Applications (VBA) language, this is usually done with a ReadString() or ReadNumber() method. These methods read the result of the query and place the result in a specified variable.

NOTE

Handle Queries Properly

If you send another command or query before reading the result of a query, the output buffer is cleared and the current response is lost. This also generates a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

Program Header Options

You can send program headers using any combination of uppercase or lowercaseASCII characters. Oscilloscope responses, however, are always returned in uppercase.

You may send program command and query headers in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form. For example:

":TIMebase:DELay 1E-6" is the long form.

":TIM:DEL 1E-6" is the short form.

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.

NOTE

Using Long Form or Short Form

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of computer memory needed for program storage and reduces I/O activity.

The rules for the short form syntax are described in Chapter 4, “Programming Conventions,” starting on page 127.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMebase:REFerence command can be set to left, center, or right. The character program data in this case may be LEFT, CENTer, or RIGHt. The command :TIMebase:REFerence RIGHt sets the time base reference to right.

The available mnemonics for character program data are always included with the instruction's syntax definition. You may send either the long form of commands, or the short form (if one exists). You may mix uppercase and lowercase letters freely. When receiving 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 can use exponential notation or 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, it means that the number should be whole. Any fractional part is ignored and truncated. Numeric data parameters that accept fractional values are called real numbers.

All numbers are expected to be strings of ASCII characters.

- 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). The number of bytes is figured automatically when you include the entire instruction in a string.

Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the oscilloscope. An example of this is the line of text written to the advisory line of the oscilloscope with the :SYSTem:DSP command:

```
:SYSTem:DSP ""This is a message.""
```

You may delimit embedded strings with either single (') or double (") quotation marks. These strings are case-sensitive, and spaces are also legal characters.

Program Message Terminator

The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the GPIB interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

NOTE

New Line Terminator Functions Like EOS and EOT

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Common Commands within a Subsystem

Common commands can be received and processed by the oscilloscope whether they are sent over the bus as separate program messages or within other program messages. If you have selected a subsystem, and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
":ACQuire:AVERage ON;*CLS;COUNt 1024"
```

is received by the oscilloscope, the oscilloscope turns averaging on, then clears the status information without leaving the selected subsystem.

If some other type of command is received within a program message, you must re-enter the original subsystem after the command. For example, the program message

```
":ACQuire:AVERage ON;:AUToscale;:ACQuire:AVERage:COUNt 1024"
```

turns averaging on, completes the autoscale operation, then sets the acquire average count. Here, :ACQuire must be sent again after AUToscale to re-enter the ACQuire subsystem and set the count.

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 lets you enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

:CHANnel1:RANGe 0.4;:TIMebase:RANGe 1

NOTE

You can Combine Compound and Simple Commands

Multiple program commands may be any combination of compound and simple commands.

Programming Getting Started

The remainder of this chapter explains how to set up the oscilloscope, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the computer. Chapter 34, “:MEASure Commands,” starting on page 869 describes getting measurement data from the oscilloscope.

Referencing the IO Library

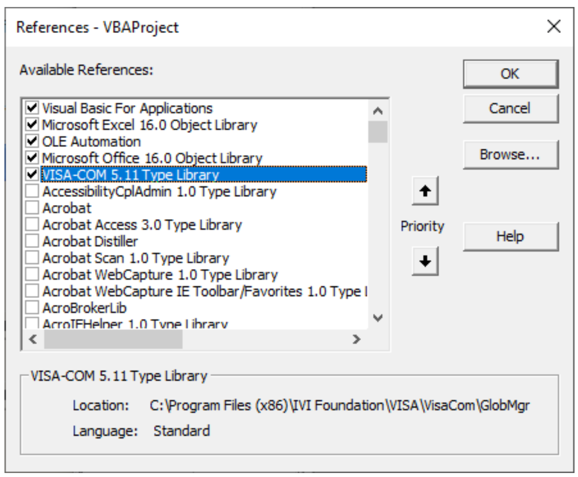
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

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 5.11 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 5.11 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 "Instructions" on page 94.

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 10 seconds
.
myScope.IO.Clear
myScope.IO.Timeout = 10000
```

When you are using GPIB, CLEAR also resets the oscilloscope's parser. The parser is the program which reads in the instructions which you send it.

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 12, "* (Common) Commands," starting on page 221.

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

See Also

- "Autoscale" on page 115
- "Setting Up the Oscilloscope" on page 116

Autoscale

The AUToscale feature of Keysight Technologies digitizing oscilloscopes performs a very useful function on unknown waveforms by automatically setting up the vertical channel, time base, and trigger level of the oscilloscope.

The syntax for the autoscale function is:

```
:AUToscale<terminator>
```

Setting Up the Oscilloscope

A typical oscilloscope setup configures the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope.

A typical example of the commands sent to the oscilloscope are:

```
:CHANnel1:PROBe 10; RANGe 16;OFFSet 1.00<terminator>

:SYSTem:HEADer OFF<terminator>

:TIMebase:RANGe 1E-3;DELay 100E-6<terminator>
```

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

Example Program

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 = 15000   ' Set interface timeout to 15 seconds.

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

' Set the time base mode to normal with the horizontal time at
' 50 us/div with 0 s of delay referenced at the center of the
' graticule.
myScope.WriteString ":TIMebase:RANGe 500E-6"   ' 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 1.0"    ' Probe attenuation
                                             ' to 1:1.
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:INPut 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 CHAN1,-0.4"   ' Trigger level to -0.
4.
myScope.WriteString ":TRIGger:MODE EDGE"      ' Edge triggering
myScope.WriteString ":TRIGger:EDGE:SLOPe POSitive" ' Trigger on pos. slo
pe.

' Configure the instrument for normal acquisition.
myScope.WriteString ":ACQuire:MODE RTIMe"   ' Normal acquisition.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Turn system headers off.
myScope.WriteString ":DISPlay:GRATicule FRAMe"   ' Grid off.
```

Using the DIGitize Command

The DIGitize command is a macro that captures data using the acquisition (ACQuire) subsystem. When the digitize process is complete, the acquisition is stopped. You can measure the captured data by using the oscilloscope or by transferring the data to a computer for further analysis. The captured data consists of two parts: the preamble and the waveform data record.

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

You can send the DIGitize command with no parameters for a higher throughput. Refer to the DIGitize command in Chapter 13, ": (Root Level) Commands," starting on page 251 for details.

When the DIGitize command is sent to an oscilloscope, the specified channel's waveform is digitized using the current ACQuire parameters. Before sending the :WAVeform:DATA? query to download waveform data to your computer, you should specify the WAVeform parameters.

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 lets you specify exactly what the digitized information contains. The following program example shows a typical setup:

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:MODE RTIME"
myScope.WriteString ":ACQuire:COMPlete 100"
myScope.WriteString ":WAVeform:SOURce CHANnel1"
myScope.WriteString ":WAVeform:FORMat BYTE"
myScope.WriteString ":ACQuire:COUNt 8"
myScope.WriteString ":ACQuire:POINts 500"
myScope.WriteString ":DIGitize CHANnel1"
myScope.WriteString ":WAVeform:DATA?"
```

This setup places the oscilloscope into the real time sampling mode using eight averages. This means that when the DIGitize command is received, the command will execute until the waveform has been averaged at least eight times.

After receiving the :WAVeform:DATA? query, the oscilloscope will start downloading the waveform information.

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

The easiest method of receiving a digitized waveform depends on data structures, available formatting, and I/O capabilities. You must convert the data values to determine the voltage value of each point. These data values are passed starting with the left most point on the oscilloscope's display. For more information, refer to the chapter, "Waveform Commands."

When using GPIB, you may abort a digitize operation by sending a Device Clear over the bus.

Receiving Information from the Oscilloscope

After receiving a query (a command header followed by a question mark), the oscilloscope places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the computer.

The input statement for receiving a response message from an oscilloscope's output queue typically has two parameters; the device address and a format specification for handling the response message. For example, with the VISA COM library, to read the result of the query command :CHANnel1:INPut? you would use the ReadString() method:

```
Dim strSetting As String
myScope.WriteString ":CHANnel1:INPut?"
strSetting = myScope.ReadString
```

This would enter the current setting for the channel 1 coupling in the string variable strSetting.

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MEASure:RISETIME?, you must follow that query with an input statement.

NOTE

Handle Queries Properly

If you send another command or query before reading the result of a query, the output buffer will be cleared and the current response will be lost. This will also generate a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

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

String Variable Example

The output of the oscilloscope may be numeric or character data depending on what is queried. Refer to the specific commands 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.

This example shows the data being returned to a string variable:

```
Dim strRang As String
myScope.WriteString ":CHANnel1:RANGe?"
strRang = myScope.ReadString
Debug.Print strRang
```

After running this program, the computer displays:

+8.00000E-01

Numeric Variable Example

This example shows the data being returned to a numeric variable:

```
Dim varRang As Variant
myScope.WriteString ":CHANnel1:RANGe?"
varRang = myScope.ReadNumber
Debug.Print "Channel 1 range: " + FormatNumber(varRang, 0)
```

After running this program, the computer displays:

.8

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. 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 4000 bytes of data, the syntax would be:

```
#44000 <4000 bytes of data> <terminator>
```

The "4" following the pound sign represents the number of digits in the number of bytes, and "4000" represents the number of bytes to be transmitted.

Multiple Queries

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"
```

Oscilloscope Status

Status registers track the current status of the oscilloscope. By checking the oscilloscope status, you can find out whether an operation has completed and is receiving triggers. Chapter 7, "Status Reporting," starting on page 151 explains how to check the status of the oscilloscope.

4 Programming Conventions

This chapter describes conventions used to program the Infiniium-Series Oscilloscopes, and conventions used throughout this manual. A description of the command tree and command tree traversal is also included.

Truncation Rule

The truncation rule is used to produce the short form (abbreviated spelling) for the mnemonics used in the programming headers and parameter arguments.

NOTE

Command Truncation Rule

The mnemonic is the first four characters of the keyword, unless the fourth character is a vowel. Then the mnemonic is the first three characters of the keyword. If the length of the keyword is four characters or less, this rule does not apply, and the short form is the same as the long form.

This document's command descriptions shows how the truncation rule is applied to commands.

Table 1 Mnemonic Truncation

Long Form	Short Form	How the Rule is Applied
RANGe	RANG	Short form is the first four characters of the keyword.
PATTern	PATT	Short form is the first four characters of the keyword.
DISK	DISK	Short form is the same as the long form.
DELay	DEL	Fourth character is a vowel; short form is the first three characters.

The Command Tree

The command tree in this document's table of contents shows all of the commands in the Infiniium-Series Oscilloscopes and the relationship of the commands to each other. The IEEE 488.2 common commands are not part of the command tree because they do not affect the position of the parser within the tree.

When a program message terminator (<NL>, linefeed - ASCII decimal 10) or a leading colon (:) is sent to the oscilloscope, the parser is set to the "root" of the command tree.

- "Command Types" on page 129
- "Tree Traversal Rules" on page 129
- "Tree Traversal Examples" on page 130

Command Types

The commands in this oscilloscope can be viewed as three types: common commands, root level commands, and subsystem commands.

- Common commands are commands defined by IEEE 488.2 and control some functions that are common to all IEEE 488.2 instruments. These commands are independent of the tree and do not affect the position of the parser within the tree. *RST is an example of a common command.
- Root level commands control many of the basic functions of the oscilloscope. These commands reside at the root of the command tree. They can always be parsed if they occur at the beginning of a program message or are preceded by a colon. Unlike common commands, root level commands place the parser back at the root of the command tree. AUToscale is an example of a root level command.
- Subsystem commands are grouped together under a common node of the command tree, such as the TIMebase commands. You may select only one subsystem at a given time. When you turn on the oscilloscope initially, the command parser is set to the root of the command tree and no subsystem is selected.

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree would be :TIMebase:RANGe. This is referred to as a compound header. A compound header is a header made up of two or more mnemonics separated by colons. The compound header contains no spaces. The following rules apply to traversing the tree.

NOTE

Tree Traversal Rules

A leading colon or a program message terminator (<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 places the oscilloscope in that subsystem until a leading colon or a program message terminator is found.

In the command tree, use the last mnemonic in the compound header as a reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. You can send any command below this point within the current program message without sending the mnemonics which appear above them (for example, REFerence).

Tree Traversal Examples

The WriteString() methods in the following examples are written using Visual Basic for Application (VBA) with the VISA COM library.

Example 1 Consider the following command:

```
myScope.WriteString ":CHANnel1:RANGe 0.5;OFFSet 0"
```

The colon between CHANnel1 and RANGe is necessary because :CHANnel1:RANGe is a compound command. The semicolon between the RANGe command and the OFFSet command is required to separate the two commands or operations. The OFFSet command does not need :CHANnel1 preceding it because the :CHANnel1:RANGe command sets the parser to the CHANnel1 node in the tree.

Example 2 Consider the following commands:

```
myScope.WriteString ":TIMebase:REFerence CENTer;POSition 0.00001"
```

or

```
myScope.WriteString ":TIMebase:REFerence CENTer"
myScope.WriteString ":TIMebase:POSition 0.00001"
```

In the first line of example 2, the "subsystem selector" is implied for the POSITION command in the compound command.

A second way to send these commands is shown in the second part of the example. Because the program message terminator places the parser back at the root of the command tree, you must reselect TIMEBASE to re-enter the TIMEBASE node before sending the POSITION command.

Example 3 Consider the following command:

```
myScope.WriteString ":TIMebase:REFerence CENTer;:CHANnel1:OFFSet 0"
```

In this example, the leading colon before CHANnel1 tells the parser to go back to the root of the command tree. The parser can then recognize the :CHANnel1:OFFSet command and enter the correct node.

Infinity Representation

The representation for infinity for this oscilloscope is 9.99999E+37. This is also the value returned when a measurement cannot be made.

Response Generation

As defined by IEEE 488.2, query responses may be buffered for these reasons:

- When the query is parsed by the oscilloscope.
- When the computer addresses the oscilloscope to talk so that it may read the response.

This oscilloscope buffers responses to a query when the query is parsed.

EOI

The EOI bus control line follows the IEEE 488.2 standard without exception.

5 LAN, USB, and GPIB Interfaces

There are several types of interfaces that can be used to remotely program the Infiniium oscilloscope including Local Area Network (LAN) interface and GPIB interface. Telnet and sockets can also be used to connect to the oscilloscope.

LAN Interface Connector

The oscilloscope is equipped with a LAN interface RJ-45 connector on the rear panel. This allows direct connect to your network. However, before you can use the LAN interface to program the oscilloscope, the network properties must be configured. Unless you are a Network Administrator, you should contact your Network Administrator to add the appropriate client, protocols, and configuration information for your LAN. This information is different for every company.

GPIB Interface Connector

The oscilloscope is not equipped with a GPIB interface connector. You can, however, order the N4865A GPIB-to-LAN adapter.

Default Startup Conditions

The following default conditions are established during power-up:

- The Request Service (RQS) bit in the status byte register is set to zero.
- All of the event registers are cleared.
- The Standard Event Status Enable Register is set to 0xFF hex.
- Service Request Enable Register is set to 0x80 hex.
- The Operation Status Enable Register is set to 0xFFFF hex.
- The Overload Event Enable Register is set to 0xFFFF hex.
- The Mask Test Event Enable Register is set to 0xFF hex.

You can change the default conditions using the *PSC command with a parameter of 1 (one). When set to 1, the Standard Event Status Enable Register is set 0x00 hex and the Service Request Enable Register is set to 0x00 hex. This prevents the Power On (PON) event from setting the SRQ interrupt when the oscilloscope is ready to receive commands.

Interface Capabilities

The interface capabilities of this oscilloscope, as defined by IEEE 488.1 and IEEE 488.2, are listed in the following table.

Table 2 Interface Capabilities

Code	Interface Function	Capability
SH1	Source Handshake	Full Capability
AH1	Acceptor Handshake	Full Capability
T5	Talker	Basic Talker/Serial Poll/Talk Only Mode/ Unaddress if Listen Address (MLA)
L4	Listener	Basic Listener/ Unaddresses if Talk Address (MTA)
SR1	Service Request	Full Capability
RL1	Remote Local	Complete Capability
PP0	Parallel Poll	No Capability
DC1	Device Clear	Full Capability
DT1	Device Trigger	Full Capability
C0	Computer	No Capability
E2	Driver Electronics	Tri State (1 MB/SEC MAX)

GPIB Command and Data Concepts

The GPIB interface has two modes of operation: command mode and data mode. The interface is in the command mode when the Attention (ATN) control line is true. The command mode is used to send talk and listen addresses and various interface commands such as group execute trigger (GET).

The interface is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. The device-dependent messages include all of the oscilloscope-specific commands, queries, and responses found in this manual, including oscilloscope status information.

Communicating Over the GPIB Interface

Device addresses are sent by the computer in the command mode to specify who talks and who listens. Because GPIB can address multiple devices through the same interface card, the device address passed with the program message must include the correct interface select code and the correct oscilloscope address.

Device Address = (Interface Select Code * 100) + Oscilloscope Address

See Also

- "Interface Select Code" on page 141
- "Oscilloscope Address" on page 141

Interface Select Code

Each interface card has a unique interface select code. This code is used by the computer to direct commands and communications to the proper interface. The default is typically "7" for the GPIB interface cards.

Oscilloscope Address

Each oscilloscope on the GPIB must have a unique oscilloscope address between decimal 0 and 30. This oscilloscope address is used by the computer to direct commands and communications to the proper oscilloscope on an interface. The default is typically "7" for this oscilloscope. You can change the oscilloscope address in the Utilities, Remote Interface dialog box.

NOTE

Do Not Use Address 21 for an Oscilloscope Address

Address 21 is usually reserved for the Computer interface Talk/Listen address, and should not be used as an oscilloscope address.

Communicating Over the LAN Interface

The device address used to send commands and receive data using the LAN interface is located in the Remote Setup dialog box (**Utilities > Remote Setup**).

The following C example program shows how to communicate with the oscilloscope using the LAN interface and the Keysight Standard Instrument Control Library (SICL).

```
#include <sicl.h>

#define BUFFER_SIZE 1024

main()
{
INST Bus;
int reason;
unsigned long actualcnt;
char buffer[ BUFFER_SIZE ];

   /* Open the LAN interface */
   Bus = iopen( "lan[130.29.71.143]:hpib7,7" );
   if( Bus != 0 ) {
      /* Bus timeout set to 20 seconds */
      itimeout( Bus, 20000 );

      /* Clear the interface */
      iclear( Bus );
      /* Query and print the oscilloscope's Id */
      iwrite( Bus, "*IDN?", 5, 1, &actualcnt );
      iread( Bus, buffer, BUFFER_SIZE, &reason, &actualcnt );
      buffer[ actualcnt - 1 ] = 0;

      printf( "%s\\n", buffer );
      iclose( Bus );
   }
}
```

Communicating via Telnet and Sockets

- "Telnet" on page 143
- "Sockets" on page 143

Telnet

To open a connection to the oscilloscope via a telnet connection, use the following syntax in a command prompt:

```
telnet Oscilloscope_IP_Address 5024
```

5024 is the port number and the name of the oscilloscope can be used in place of the IP address if desired.

After typing the above command line, press enter and a SCPI command line interface will open. You can then use this as you typically would use a command line.

Sockets

Sockets can be used to connect to your oscilloscope on either a Windows or Unix machine.

The sockets are located on port 5025 on your oscilloscope. Between ports 5024 and 5025, only six socket ports can be opened simultaneously. It is, therefore, important that you use a proper close routine to close the connection to the oscilloscope. If you forget this, the connection will remain open and you may end up exceeding the limit of six socket ports.

Some basic commands used in communicating to your oscilloscope include:

- The receive command is: recv
- The send command is: send

Below is a programming example (for a Windows-based machine) for opening and closing a connection to your oscilloscope via sockets.

```
#include <winsock2.h>

void main ()
{
   WSADATA wsaData;
   SOCKET mysocket = NULL;
   char* ipAddress = "130.29.70.70";
   const int ipPort = 5025;

   //Initialize Winsock
   int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
   if(iResult != NO_ERROR)
   {
      printf("Error at WSAStartup()\\n");
```

```
      return NULL;
   }

   //Create the socket
   mySocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCOP);
   if(mySocket == INVALID_SOCKET)
   {
      printf("Error at socket(): %ld\\n", WSAGetLastError());
      WSACleanup();
      return NULL;
   }

   sockaddr_in clientService;
   clientService.sin_family = AF_INET;
   clientService.sin.addr.s_addr = inet_addr(ipAddress);
   clientService.sin_port = htons(ipPort);

   if(connect(mySocket, (SOCKADDR*) &clientService, sizeof(clientService
)))
   {
      printf("Failed to connect.\\n");
      WSACleanup();
      return NULL;
   }

   //Do some work here

   //Close socket when finished
   closesocket(mySocket);
}
```

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions that are taken when these commands are received by the oscilloscope.

Device Clear

The device clear (DCL) and selected device clear (SDC) commands clear the input buffer and output queue, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

Group Execute Trigger

The group execute trigger (GET) command arms the trigger. This is the same action produced by sending the RUN command.

Interface Clear

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system computer.

6 Message Communication and System Functions

This chapter describes the operation of oscilloscopes that operate in compliance with the IEEE 488.2 (syntax) standard. It is intended to give you enough basic information about the IEEE 488.2 standard to successfully program the oscilloscope. You can find additional detailed information about the IEEE 488.2 standard in ANSI/IEEE Std 488.2-1987, "*IEEE Standard Codes, Formats, Protocols, and Common Commands*."

This oscilloscope series is designed to be compatible with other Keysight Technologies IEEE 488.2 compatible instruments. Oscilloscopes that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (GPIB bus standard); however, IEEE 488.1 compatible oscilloscopes may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the oscilloscope and the computer will communicate. It also defines some common capabilities that are found in all IEEE 488.2 oscilloscopes. This chapter also contains some information about the message communication and system functions not specifically defined by IEEE 488.2.

Protocols

The message exchange protocols of IEEE 488.2 define the overall scheme used by the computer and the oscilloscope to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

- "Functional Elements" on page 148
- "Protocol Overview" on page 148
- "Protocol Operation" on page 149
- "Protocol Exceptions" on page 149
- "Suffix Multiplier" on page 149
- "Suffix Unit" on page 150

Functional Elements

Before proceeding with the description of the protocol, you should understand a few system components, as described here.

Input Buffer The input buffer of the oscilloscope is the memory area where commands and queries are stored prior to being parsed and executed. It allows a computer to send a string of commands, which could take some time to execute, to the oscilloscope, then proceed to talk to another oscilloscope while the first oscilloscope is parsing and executing commands.

Output Queue The output queue of the oscilloscope is the memory area where all output data or response messages are stored until read by the computer.

Parser The oscilloscope's parser is the component that interprets the commands sent to the oscilloscope and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and execution of commands begins when either the oscilloscope recognizes a program message terminator, or the input buffer becomes full. If you want to send a long sequence of commands to be executed, then talk to another oscilloscope while they are executing, you should send all of the commands before sending the program message terminator.

Protocol Overview

The oscilloscope and computer communicate using program messages and response messages. These messages serve as the containers into which sets of program commands or oscilloscope responses are placed.

A program message is sent by the computer to the oscilloscope, and a response message is sent from the oscilloscope to the computer in response to a query message. A query message is defined as being a program message that contains one or more queries. The oscilloscope will only talk when it has received a valid query message, and therefore has something to say. The computer should only attempt to read a response after sending a complete query message, but before sending another program message.

NOTE

Remember this Rule of Oscilloscope Communication

The basic rule to remember is that the oscilloscope will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When you turn the oscilloscope on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The oscilloscope and the computer communicate by exchanging complete program messages and response messages. This means that the computer should always terminate a program message before attempting to read a response. The oscilloscope will terminate response messages except during a hard copy output.

After you send a query message, the next message should be the response message. The computer should always read the complete response message associated with a query message before sending another program message to the same oscilloscope.

The oscilloscope allows the computer to send multiple queries in one query message. This is called sending a "compound query". Multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner.

Suffix Multiplier

The suffix multipliers that the oscilloscope will accept are shown in the following table.

Table 3 <suffix mult>

Value	Mnemonic	Value	Mnemonic
1E18	EX	1E-3	M
1E15	PE	1E-6	U
1E12	T	1E-9	N

Table 3 <suffix mult> (continued)

Value	Mnemonic	Value	Mnemonic
1E9	G	1E-12	P
1E6	MA	1E-15	F
1E3	K	1E-18	A

Suffix Unit

The suffix units that the oscilloscope will accept are shown in the following table.

Table 4 <suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second

7 Status Reporting

An overview of the oscilloscope's status reporting structure is shown in Figure 1. The status reporting structure shows you how to monitor specific events in the oscilloscope. Monitoring these events lets you determine 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 oscilloscope.
- To generate a service request (SRQ) interrupt to an external computer, enable at least one bit in the Status Byte Register.

KEYSIGHT TECHNOLOGIES

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. IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also oscilloscope-defined structures and bits.

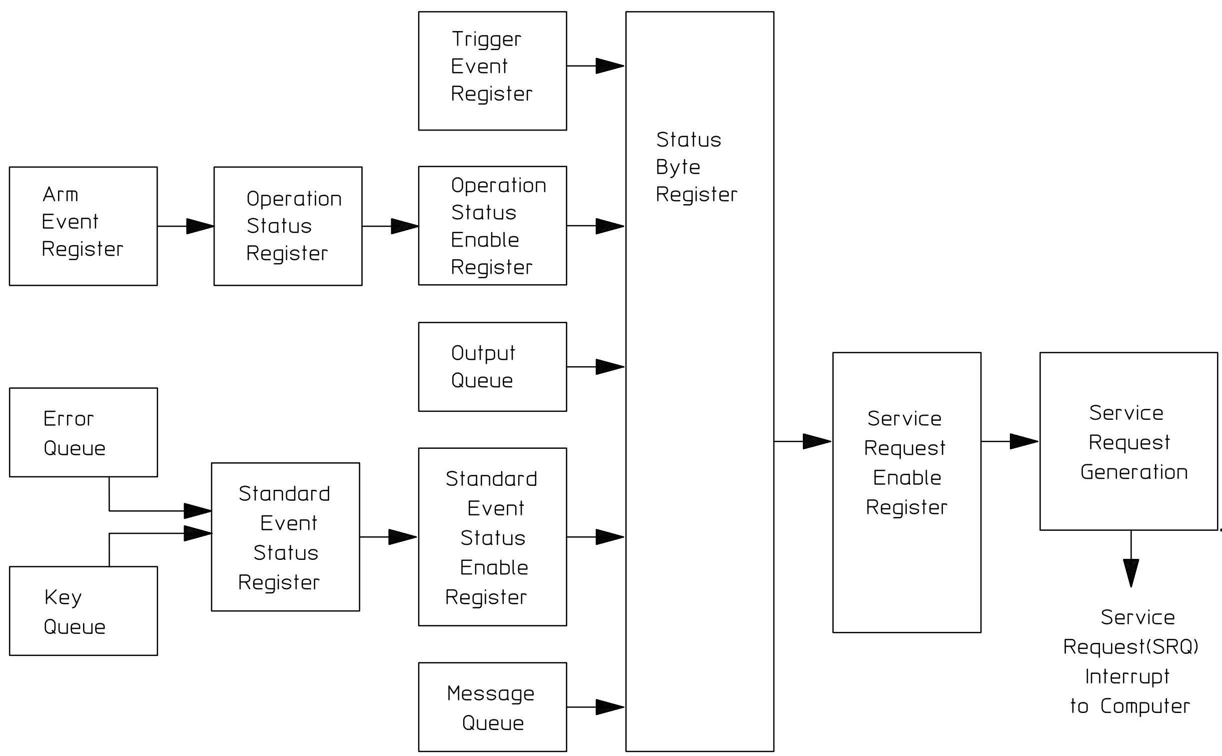


Figure 1 Status Reporting Overview Block Diagram

The status reporting structure consists of the registers shown here.

The definitions for each bit in the status reporting data structure are listed in the following table.

Table 5 Status Reporting Bit Definition

Bit	Description	Definition
PON	Power On	Indicates power is turned on.
URQ	User Request	Not Used. Permanently set to zero.
CME	Command Error	Indicates if the parser detected an error.
EXE	Execution Error	Indicates if a parameter was out of range or was inconsistent with the current settings.

Table 5 Status Reporting Bit Definition (continued)

Bit	Description	Definition
DDE	Device Dependent Error	Indicates if the device was unable to complete an operation for device-dependent reasons.
QYE	Query Error	Indicates if the protocol for queries has been violated.
RQL	Request Control	Indicates if the device is requesting control.
OPC	Operation Complete	Indicates if the device has completed all pending operations.
OPER	Operation Status Register	Indicates if any of the enabled conditions in the Operation Status Register have occurred.
RQS	Request Service	Indicates that the device is requesting service.
MSS	Master Summary Status	Indicates if a device has a reason for requesting service.
ESB	Event Status Bit	Indicates if any of the enabled conditions in the Standard Event Status Register have occurred.
MAV	Message Available	Indicates if there is a response in the output queue.
MSG	Message	Indicates if an advisory has been displayed.
USR	User Event Register	Indicates if any of the enabled conditions have occurred in the User Event Register.
TRG	Trigger	Indicates if a trigger has been received.
WAIT TRIG	Wait for Trigger	Indicates the oscilloscope is armed and ready for trigger.

Status Reporting Data Structures

The different status reporting data structures, descriptions, and interactions are shown in Figure 2. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, you must enable the corresponding bits. 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 the computer, you must enable at least one bit in the Status Byte Register. 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.

For more information about common commands, see the "Common Commands" chapter.

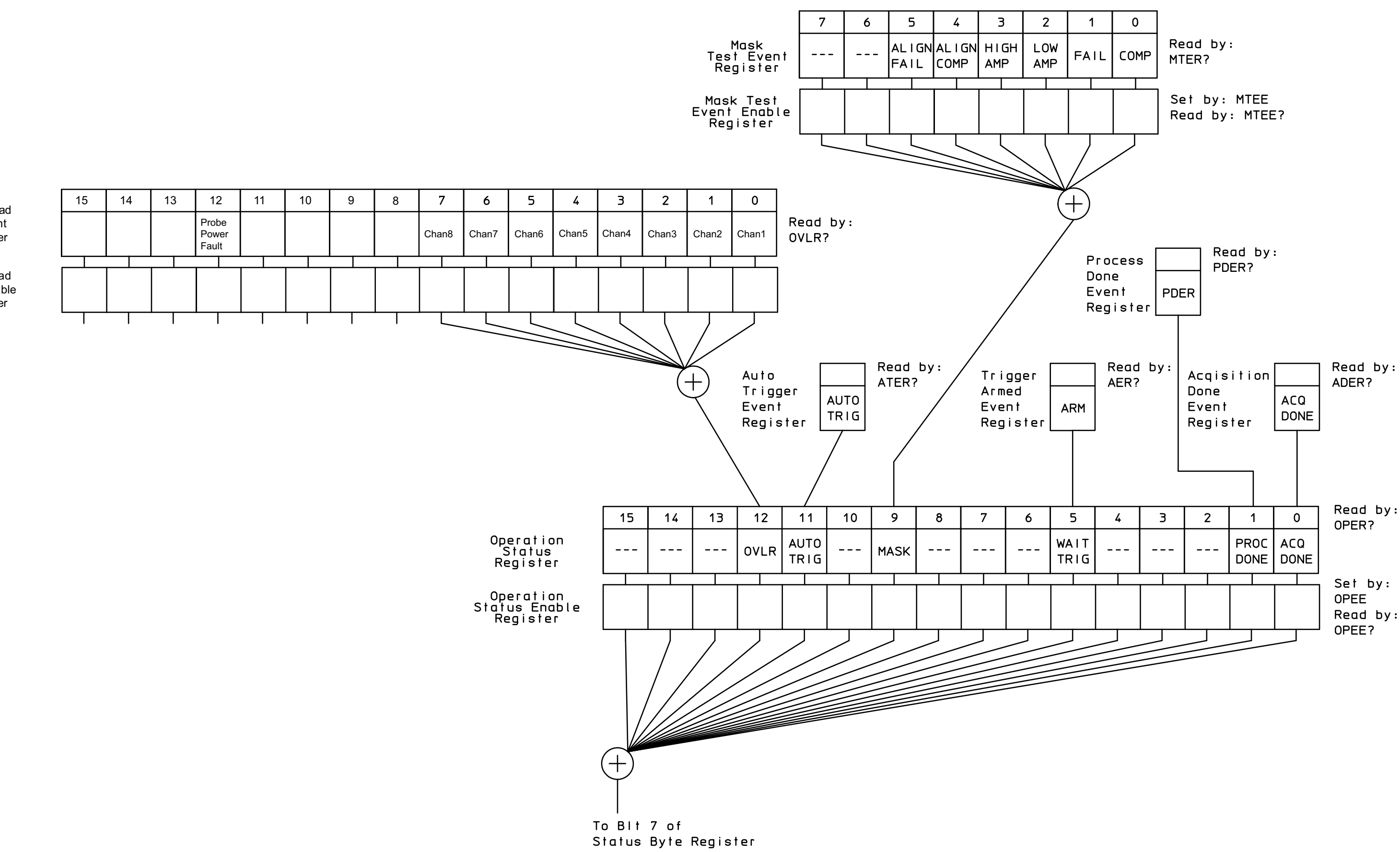


Figure 2 Status Reporting Data Structures

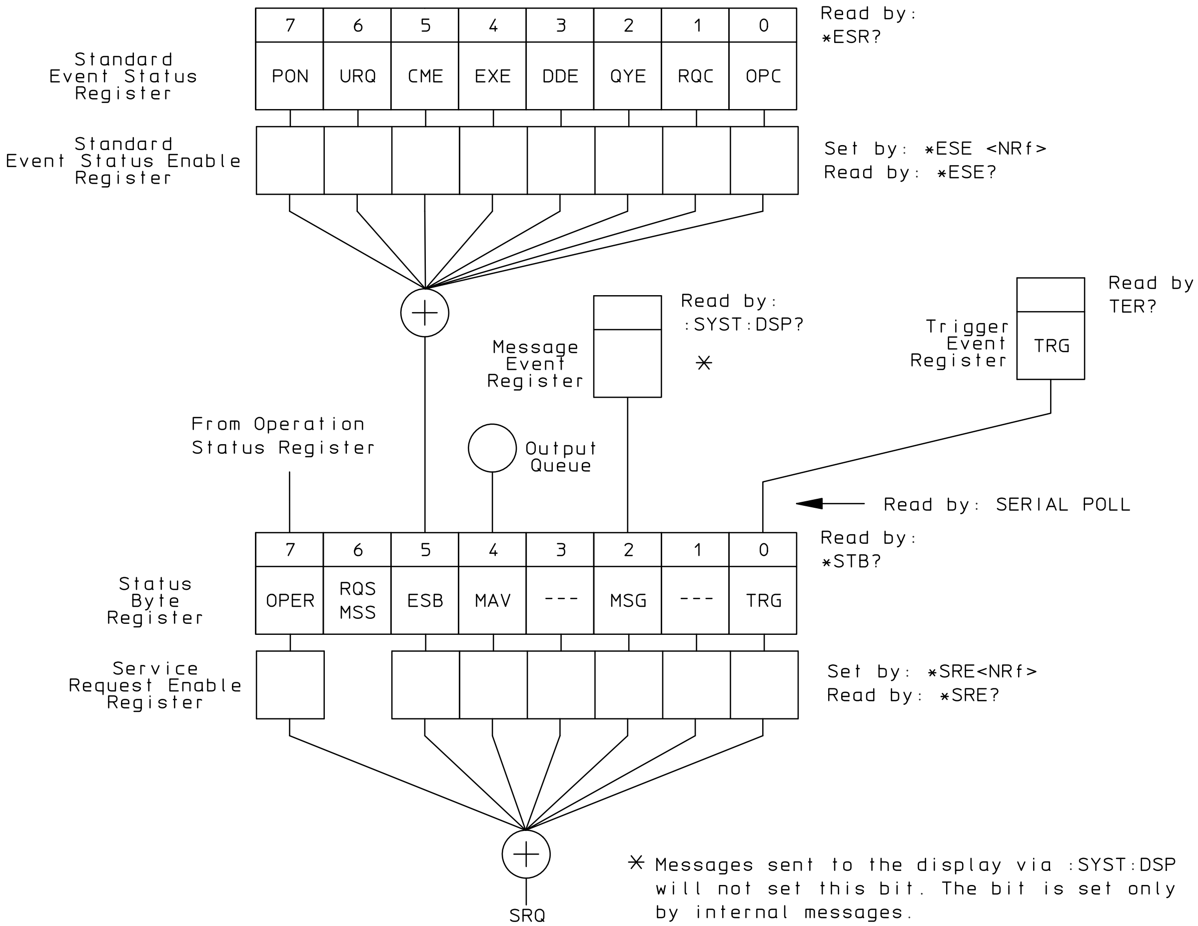


Figure 3 Status Reporting Data Structures (Continued)

Status Byte Register

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.

You can read the Status Byte Register using either the *STB? common command query or the GPIB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? query reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any effect 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.

The only other bit in the Status Byte Register affected by the *STB? query is 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? query.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, a program would print the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

Example 1 This example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register when none of the register's summary bits are enabled to generate an SRQ interrupt.

```
Dim varStbValue As Variant
myScope.WriteString ":SYSTem:HEADer OFF;*STB?"   'Turn headers off
varStbValue = myScope.ReadNumber
Debug.Print "Status Byte Register, Read: 0x" + Hex(varStbValue)
```

The next program prints "0x84" and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal 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 2 The following example uses the resource session object's ReadSTB method to read the contents of the oscilloscope's Status Byte Register.

```
varStbValue = myScope.IO.ReadSTB
Debug.Print "Status Byte Register, Serial Poll: 0x" + Hex(varStbValue)
```

NOTE

Use Serial Polling to Read the 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.

See Also

- "Example: Checking for Armed Status" on page 176

Service Request Enable Register

Setting the Service Request Enable Register bits enables corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command, and the bits that are set are read with the *SRE? query. Bit 6 always returns 0. Refer to the Status Reporting Data Structures shown in Figure 2.

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).

Message Event Register

This register sets the MSG bit in the status byte register when an internally generated message is written to the advisory line on the oscilloscope. The message is read using the :SYSTem:DSP? query. Note that messages written to the advisory line on the oscilloscope using the :SYSTem:DSP command does not set the MSG status bit.

Trigger Event Register

This register sets the TRG bit in the status byte register when a trigger event occurs.

The trigger event register stays set until it is cleared by reading the register with the TER? query or by using the *CLS (clear status) command. If your application needs to detect multiple triggers, the trigger event register must be cleared after each one.

If you are using the Service Request to interrupt a computer operation when the trigger bit is set, you must clear the event register after each time it is set.

Standard Event Status Register

The Standard Event Status Register (ESR) monitors the following oscilloscope status events:

- PON - Power On
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occurs, the corresponding bit is set in the register. If the corresponding bit is also enabled in the Standard Event Status Enable Register, a summary bit (ESB) in the Status Byte Register is set.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all bits set at the present time.

Example The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
myScope.WriteString ":SYSTem:HEADer OFF"   'Turn headers off
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
Debug.print "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.

Standard Event Status Enable Register

For any of the Standard Event Status Register bits to generate a summary bit, you must first enable the bit. Use the *ESE (Event Status Enable) common command to set the corresponding bit in the Standard Event Status Enable Register. Set bits are read with the *ESE? query.

Example Suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the (Standard) Event Status Register are bits 2 through 5 (hexadecimal value 0x3C). Therefore, you can enable any of these bits to generate the summary bit by sending:

```
myScope.WriteString "*ESE " + CStr(CInt("&H3C"))
```

Whenever an error occurs, it sets one of these bits in the (Standard) Event Status Register. Because all the error related bits are enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) 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 in the Status Byte Register.

Operation Status Register

This register hosts the following bits:

- ACQ DONE bit 0
- PROC DONE bit 1
- WAIT TRIG bit 5
- MASK bit 9
- AUTO TRIG bit 11
- OVLR bit 12

The ACQ DONE done bit is set by the Acquisition Done Event Register.

The PROC DONE bit is set by the Process Done Event Register and indicates that all functions and all math processes are done.

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates the trigger is armed.

The MASK bit is set whenever at least one of the Mask Test Event Register bits is enabled.

The AUTO TRIG bit is set by the Auto Trigger Event Register.

The OVLR bit is set whenever at least one of the Overload Event Register bits is enabled.

If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Register is read and cleared with the OPER? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

See Also

- "Example: Checking for Armed Status" on page 176

Operation Status Enable Register

For any of the Operation Status Register bits to generate a summary bit, you must first enable the bit. Use the OPEE (Operation Event Status Enable) command to set the corresponding bit in the Operation Status Enable Register. Set bits are read with the OPEE? query.

Example Suppose your application requires an interrupt whenever any event occurs in the mask test register. The error status bit in the Operation Status Register is bit 9. Therefore, you can enable this bit to generate the summary bit by sending:

```
myScope.WriteString ":OPEE " + CStr(CInt("&H200"))
```

Whenever an error occurs, the oscilloscope sets this bit in the Mask Test Event Register. Because this bit is enabled, a summary bit is generated to set bit 9 (OPER) in the Operation Status Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

NOTE

Disabled Operation Status Register Bits Respond, but Do Not Generate a Summary Bit

Operation 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 in the Status Byte Register.

Mask Test Event Register

This register hosts the following bits:

- Mask Test Complete bit (bit 0)
- Mask Test Fail bit (bit 1)
- Mask Low Amplitude bit (bit 2)
- Mask High Amplitude bit (bit 3)
- Mask Align Complete bit (bit 4)
- Mask Align Fail bit (bit 5)

The Mask Test Complete bit is set whenever the mask test is complete.

The Mask Test Fail bit is set whenever the mask test failed.

The Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.

The Mask High Amplitude bit is set whenever the signal is above the mask amplitude.

The Mask Align Complete bit is set whenever the mask align is complete.

The Mask Align Fail bit is set whenever the mask align failed.

If any of these bits are set, the MASK bit (bit 9) of the Operation Status Register is set. The Mask Test Event Register is read and cleared with the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

Mask Test Event Enable Register

For any of the Mask Test Event Register bits to generate a summary bit, you must first enable the bit. Use the MTEE (Mask Test Event Enable) command to set the corresponding bit in the Mask Test Event Enable Register. Set bits are read with the MTEE? query.

Example Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
myScope.WriteString ":MTEE " + CStr(CInt("&H2"))
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

NOTE

Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit

Mask Test Event 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 in the Operation Status Register.

Acquisition Done Event Register

The Acquisition Done Event Register (ACQ DONE) sets bit 0 (ACQ DONE bit) in the Operation Status Register when the oscilloscope acquisition is completed.

The ACQ DONE event register stays set until it is cleared by reading the register by a ADER? query. If your application needs to detect multiple acquisitions, the ACQ DONE event register must be cleared after each acquisition.

See Also

- "Example: Blocking and Polling Synchronization" on page 199

Process Done Event Register

The Process Done Event Register (PDER) sets bit 1 (PROC DONE) of the Operation Status Register when all functions and all math operations are completed. The PDER bit stays set until cleared by a PDER? query.

See Also

- "Example: Blocking and Polling Synchronization" on page 199

Trigger Armed Event Register

The Trigger Armed Event Register (TDER) sets bit 5 (WAIT TRIG) in the Operation Status Register when the oscilloscope becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

See Also

- "Example: Checking for Armed Status" on page 176

Auto Trigger Event Register

The Auto Trigger Event Register (AUTO TRIG) sets bit 11 (AUTO TRIG) in the Operation Status Register when an auto trigger event occurs. The AUTO TRIG register stays set until it is cleared by reading the register with the ATER? query. If the application needs to detect multiple auto trigger events, the AUT TRIG register must be cleared after each one.

Error Queue

As errors are detected, they are placed in an error queue. This queue is a first-in, first-out 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 oldest errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the "Queue overflow" message).

The error queue is read with the :SYSTem:ERRor? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return 0, "No error."

The error queue is cleared when any of these events occur:

- When the oscilloscope is powered up.
- When the oscilloscope receives the *CLS common command.
- When the last item is read from the error queue.

For more information on reading the error queue, refer to the :SYSTem:ERRor? query in the System Commands chapter. For a complete list of error messages, refer to the chapter, "Error Messages."

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain oscilloscope 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 queue is read with the :SYSTem:DSP? query. 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 following a program message terminator, the output queue is also cleared.

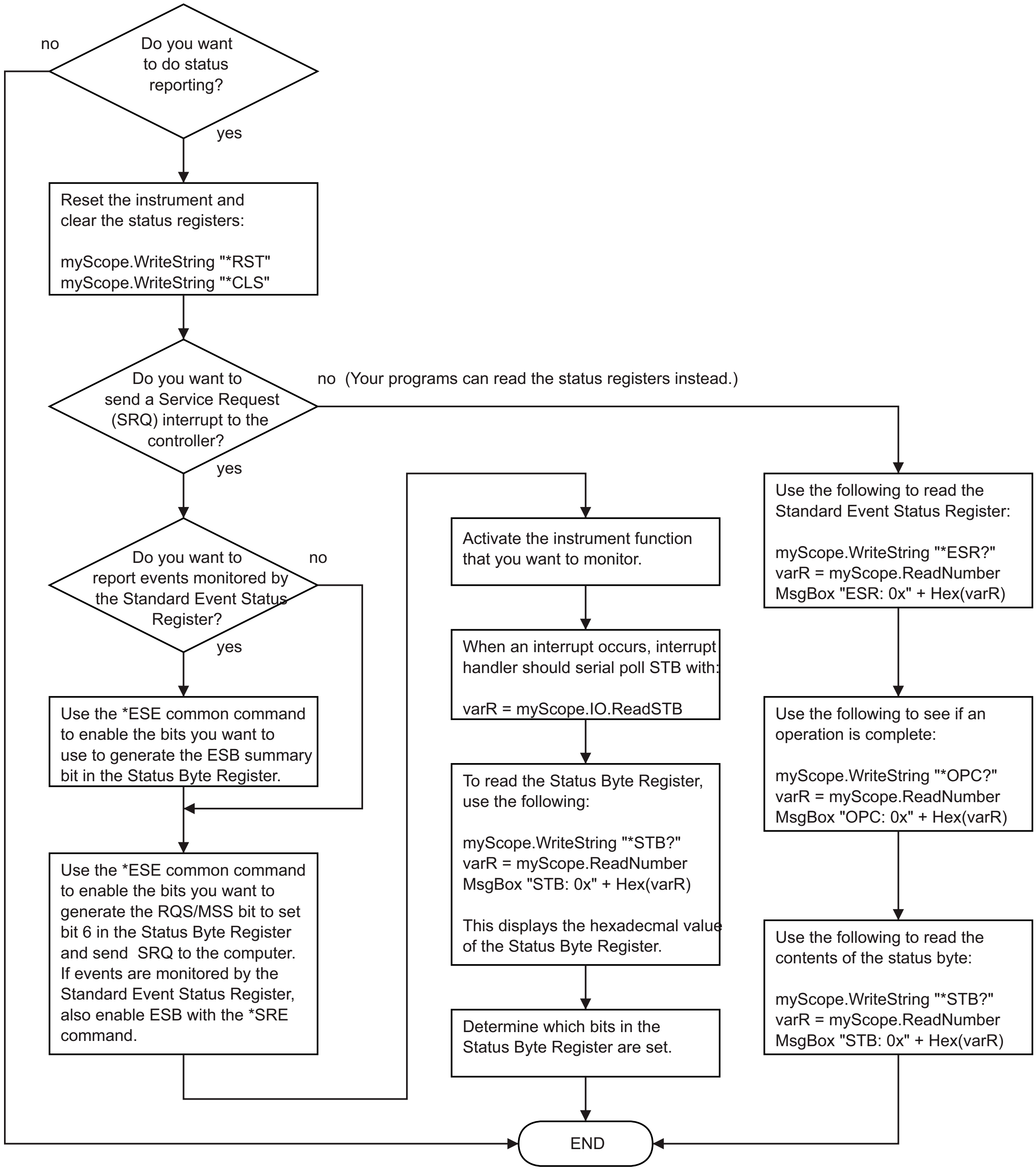


Figure 4 Status Reporting Decision Chart

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 Infiniium oscilloscope is armed.
# *********************************************************************

# Import modules
# ---------------------------------------------------------------------
import sys
import visa
import time

# Initialization constants
# ---------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = "TCPIP0::141.121.231.13::hislip0::INSTR"
GLOBAL_TOUT =  10000   # IO timeout in milliseconds

# Connect and initialize oscilloscope
# ---------------------------------------------------------------------
# Define VISA Resource Manager & Install directory
rm = visa.ResourceManager('C:\\Windows\\System32\\visa32.dll')

# Define and open the oscilloscope using the VISA address
KsInfiniiumScope = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
KsInfiniiumScope.timeout = GLOBAL_TOUT

# Clear the instrument bus
KsInfiniiumScope.clear()

# Reset the oscilloscope.
KsInfiniiumScope.write("*RST")

# Autoscale to set up vertical scale and trigger level on channel 1.
KsInfiniiumScope.write(":AUToscale:CHANnels DISPlayed")
KsInfiniiumScope.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
KsInfiniiumScope.write(":TIMebase:RANGe 10")
# Prevent Auto trigger.
KsInfiniiumScope.write(":TRIGger:SWEep TRIGgered")

# =====================================================================
# 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.
# ======================================================================

# Stop the oscilloscope.
KsInfiniiumScope.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.
KsInfiniiumScope.write("*CLS")

# Because the :AER? query will not work with :DIGitize (which is
# blocking), use the :SINGle command to start the acquisition.
KsInfiniiumScope.write(":SINGle")

# Method 1: Determine if armed using :AER? query.
# ----------------------------------------------------------------------
# Define armed criteria.
ARMED = 1

# Test for armed.
ARMED_STATUS = int(KsInfiniiumScope.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(KsInfiniiumScope.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.
# ======================================================================

# Stop the oscilloscope.
KsInfiniiumScope.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.
KsInfiniiumScope.write("*CLS")

# Mask out all bits in the Operation Status Register except for
# the ARM bit.
KsInfiniiumScope.write(":OPEE 32")  # "Unmask" only the arm bit

# Use the :DIGitize command to start the acquisition.
KsInfiniiumScope.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(KsInfiniiumScope.read_stb())
ARMED_STATUS = STATUS_BYTE & ARM_MASK
# Note that you could also do:
#  ARMED_STATUS = int(KsInfiniiumScope.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(KsInfiniiumScope.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 Register with :OPER?
# ----------------------------------------------------------------------
# This method reads the Operation Status Register's Wait Trig bit
# (bit 5) using the :OPER? query.
#
# The Operation Status Register bits do NOT go low (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, or by reading the
# Armed Event Register register with the :AER? query.
# ======================================================================
```

```
# Stop the oscilloscope.
KsInfiniiumScope.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.
KsInfiniiumScope.write("*CLS")

# Because the :OPER? query will not work with :DIGitize (which is
# blocking), use the :SINGle command to start the acquisition.
KsInfiniiumScope.write(":SINGle")

# Method 3: Determine if armed using :OPER? query.
# ---------------------------------------------------------------------
# Define register bit masks for the Operation Status Register
ARM_BIT = 5
# 1 leftshift 5 = 32 (bit 5 in the Operation Status Register)
ARM_MASK = 1 << ARM_BIT

# Define armed criteria.
ARMED = 1 << ARM_BIT   # 1 leftshift 5 = 32

# Test for armed.
STATUS_REGISTER = int(KsInfiniiumScope.query(":OPER?"))
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(KsInfiniiumScope.query(":OPER?"))
    ARMED_STATUS = STATUS_REGISTER & ARM_MASK

print("Oscilloscope is armed (method 3, using :OPER? query)!")
print("It took " + str(time.perf_counter() - now) +\
      " seconds to arm.\n")

# End of Script
# ---------------------------------------------------------------------
KsInfiniiumScope.clear()   # Clear communications interface
KsInfiniiumScope.close()   # Close communications interface
print("All done.")
```

8 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*, must finish their task before the execution of the next command starts.
- *Overlapped commands* run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

In the Infiniium oscilloscopes, for the most part, commands and queries operate differently.

- Most commands are overlapped.

  Exceptions to this are the :DIGitize command and the :DISK:SAVE commands, which are sequential (blocking).
- Most queries are sequential (blocking).

  Exceptions to this are queries, like measurement results (see Chapter 34, “:MEASure Commands,” starting on page 869), that copy information from the display without having to make acquisitions.

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.

Waiting for Overlapped Commands to Complete

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:

```
:WMEMory1:SAVE CHAN1;*OPC?;:WMEMory2:SAVE CHAN2
```

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.

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

- Chapter 9, “Using the *OPC? (Operation Complete) Query,” starting on page 183
- "*OPC — Operation Complete" on page 230
- "*ESR? — Event Status Register" on page 226
- Chapter 10, “Remote Acquisition Synchronization,” starting on page 193

9 Using the *OPC? (Operation Complete) Query

When you programmatically set up an instrument for an operation, you typically send multiple instrument commands. Most instruments are able to accept and process multiple commands. Provided the instrument can complete all these commands before an additional conflicting command or query is sent, the code will work as anticipated.

A better way to send a block of commands is to have the instrument report when the block is complete before sending additional commands.

The SCPI (Standard Commands for Programmable Instruments) standard defines the *OPC? query as a way to ensure an instrument has completed all operations before your program continues.

See Also

- "*OPC — Operation Complete" on page 230

KEYSIGHT TECHNOLOGIES

The *OPC? Query in Previous Oscilloscopes

In the previous Infiniium oscilloscope software that supports the Infiniium 9000 Series, S-Series, 90000A Series, 90000 X-Series, V-Series, 90000 Q-Series, and Z-Series oscilloscopes, the implementation of the *OPC? query does not honor the definition in the SCPI standard. Instead it returns from the *OPC? query after parsing the previous commands, not after the effects of the previous commands are completed. If your program sends `:SINGle;*OPC?`, the previous Infiniium oscilloscope software returns with the semantic equivalent of "Okay, I got your request for a single acquisition, and it is coming". This does not guarantee that the single acquisition is complete and that the oscilloscope is ready for the next command. Programs written to this flawed definition obviously have to adapt in order to get valid results. In fact, the previous Infiniium oscilloscope software created new synchronizing methods to work around this fault.

The :PDER? (Processing Done Event Register) query is one such evolution to work around the faulty *OPC? query. A coherent program can be written using only the :PDER? query (without using *OPC? query).

Here is the :PDER? equivalent of `:SINGle;*OPC?` that causes your program to wait until the single acquisition has actually completed.

```
*CLS
:SINGle
while (PDER == 0)
   read PDER
```

The :PDER? query provides an alternate way for remote applications to ensure all previous commands are complete before continuing. However, it is not a blocking query like the *OPC? query, and it does not solve all remote application problems since existing application software would need to be re-written to use the new :PDER? query sequence.

(The :PDER? query is still present and works in the UXR/MXR/EXR-Series oscilloscopes.)

See Also

- ":PDER? — Processing Done Event Register" on page 274

The *OPC? Query in UXR/MXR/EXR-Series Oscilloscopes

In the UXR/MXR/EXR-Series oscilloscopes, the *OPC? query is implemented as the SCPI standard definition intends. The *OPC? query is sent when you want to ensure all previous commands have completed, not just received. This is a significant change from the Infiniium oscilloscope software that supports the 9000 Series, S-Series, 90000A Series, 90000 X-Series, V-Series, 90000 Q-Series, and Z-Series oscilloscopes.

For example, in the UXR/MXR/EXR-Series oscilloscopes:

- With `:SINGle;*OPC?`, the single acquisition and all analysis that results due to the single acquisition completes before returning.
- With `:CHANnel1:DISPlay ON;*OPC?`, the channel is displayed and any configured analysis for that channel is performed before returning.

The *OPC? Query While Running

When the Infiniium oscilloscope is Running (that is, continuously performing and processing acquisitions, as initiated by the `:RUN` command), "operation complete" will be marked for every trigger. Applications wanting synchronous data queries while running must understand that the *OPC? query will return "1" on each trigger. Many Infiniium measurements require multiple passes of analysis, and cannot be deterministically queried by one *OPC? query when Running. This is very important. Many applications utilize `:RUN`, and synchronization is achieved via other mechanisms.

The *OPC? Query and Timeouts

All SCPI commands are subject to the IO timeout that applications declare, and the *OPC? query is no exception. A program that sets an IO timeout of 60 seconds declares that no SCPI command will take longer than 60 seconds. If an application has a mix of fast and slow expected responses, the application will typically use a timeout suitable for the longest operation.

The *OPC Command Versus the *OPC? Query

By definition, the *OPC? query is available as a blocking query, and the *OPC command is available to set a bit in the status register when an operation is complete; the status register bit can be polled for its state without blocking. Some applications prefer not to have their execution thread blocked, implementing instead internal *OPC command looping functions to facilitate independent timeout handling.

The synchronous *OPC? query is just that. Sending and reading the *OPC? is a synchronous call that will not return unless satisfied or the IO timeout has occurred.

The asynchronous *OPC command is shown in the following code fragment:

```
int AsyncOpcQuery()
{
   int opc = 0;
   while (opc == 0)
   {
      int esr = GetQueryAsInt("*ESR?", m_Session);
      esr &= 0x01;
      opc = esr;
      if (opc == 0)
      {
         UserTimeoutHandler();
      }
   }
   return opc;
}
```

The *ESR? query returns the contents of the Standard Event Status Register. Bit 0 of this register is the OPC bit. Within the while loop, an application can put whatever timeout handling is necessary.

See Also

- "*OPC — Operation Complete" on page 230
- "*ESR? — Event Status Register" on page 226

How to Use the *OPC? Query

The *OPC? query is great for synchronization. It is also a potential slowdown if used excessively. Not all commands require synchronization. How does an application writer decide when to use the *OPC? query?

- "Commands That Reasonably Could Be Synchronized With *OPC?" on page 189
- "Synchronize in Reasonably Sized Blocks" on page 190
- "*OPC? Is the Guarantee That the "Chain of Operations" Is Completed" on page 190
- "Set Program State When Data Is Not Active (If Possible)" on page 191

Commands That Reasonably Could Be Synchronized With *OPC?

Here are some examples of commands that can benefit an application if they are followed by an *OPC? query. This is not all of the commands. Hopefully, the examples demonstrate that if a command sequence enables one state as the predicate of following states, it makes sense to synchronize at the sub-state boundaries. For example, if a measurement is made on an FFT, it makes sense to set up and enable the FFT, synchronize, then proceed to make the measurement.

Table 6 Examples of Commands Synchronized With *OPC?

Action	Example Commands	Notes
Default setup	`*RST;*OPC?`	
Single acquisition	`:SINGLE;*OPC?`	Acquiring data followed by *OPC? ensures the acquisition and all dependent installed measurements and functions will be completed prior to returning.
Clear display	`:CDISplay;*OPC?`	Takes a long time.
Install measurement	`:MEASure:TVOLt 0, -1, CHANnel1` `*OPC?`	

Enable math function	`:FUNCtion1:FFTMagnitude CHANnel1`
	`:FUNCtion1:DISPlay ON`
	`*OPC?`
		Functions are activated when they are displayed (that is, turned on). For a function, setting DISPlay ON, or STATe ON, is equivalent to a channel and SINGle.
		FFT's by default enable a new graticule, time consuming, and are slow to compute. Synchronizing prior to enabling subsequent analysis is prudent.

Table 6 Examples of Commands Synchronized With *OPC? (continued)

Action	Example Commands	Notes

Enable jitter measurements	`:MEASure:THResholds:GENeral:METHod ALL,HYSTeresis`
	`:MEASure:THResholds:GENeral:HYSTeresis ALL,0.1,0.0`
	`:MEASure:RJDJ:STATe ON`
	`*OPC?`	

Turn off a graticule	`:DISPlay:GRATicule:AREA2:STATe OFF` `*OPC?`	Enabling, or disabling graticules reconfigures the memory allocation associated with the display, and is a complicated time consuming action.
Enable channel filters		Filters can be very slow, depending upon the configuration. InfiniiSim, Differential Channels, Equalization, Bandwidth Limits, cause everything in the installed analysis tree to recompute. In addition, these filters have the potential to run in software depending upon configuration. Configuring state prior to acquiring or enabling display/state is going to be efficient.

All of the above examples demonstrate the enabling of a complicated action followed by the synchronization. Any change that forces software to reanalyze data is a potential location to consider disabling/configuring/re-enabling.

Synchronize in Reasonably Sized Blocks

It is impossible to program an Infiniium oscilloscope without lots of commands. Sending them all and waiting once might not always be the best. Synchronizing by coherent blocks often is efficient. It evolves into an art of sorts. How many commands ought be sent before giving Infiniium a chance to catch up?

See Also

- "*OPC? Is the Guarantee That the "Chain of Operations" Is Completed" on page 190
- "Set Program State When Data Is Not Active (If Possible)" on page 191

*OPC? Is the Guarantee That the "Chain of Operations" Is Completed

For example:

1 Set up all the measurement of the CHANnel1 followed by the *OPC? query.

2 Send `:SINGle;*OPC?`

3 Infiniium returns from the *OPC? query after the acquisition and the measurement analysis of the new acquired waveform is complete.

Histogram Example

```
*CLS;:RUN
:STOP
:HISTogram:WINDow:SOURce CHANnel1
:HISTogram:AXIS VERTical
:HISTogram:SCALe:SIZE 4
:HISTogram:WINDow:WINDow:LLIMit -500e-9
:HISTogram:WINDow:WINDow:RLIMit 500e-9
:HISTogram:WINDow:WINDow:TLIMit 1.0
:HISTogram:WINDow:WINDow BLIMit -1.0
:HISTogram:MODE WAVeforms
:MEASure:HISTogram:MIN?
```

There are several things potentially wrong with the commands being sent. First, we do not know if we got a trigger, so we should at least wait for the :TERegister? event, or an *OPC? after sending :RUN.

The next opportunity is after sending :STOP. Sending the command is giving the order, but how the oscilloscope is configured determines how long it will take to analyze the last acquisition.

The third opportunity is after sending the eight histogram commands. The last send enables the histogram. Adding an *OPC? query here would ensure the :MEASure:HISTogram:MIN? query is happening after the histogram is installed and enabled. Here is the same code with the *OPC? query inserted in the most likely places:

```
*CLS;:RUN
*OPC?
:STOP
:HISTogram:WINDow:SOURce CHANnel1
:HISTogram:AXIS VERTical
:HISTogram:SCALe:SIZE 4
:HISTogram:WINDow:WINDow:LLIMit -500e-9
:HISTogram:WINDow:WINDow:RLIMit 500e-9
:HISTogram:WINDow:WINDow:TLIMit 1.0
:HISTogram:WINDow:WINDow BLIMit -1.0
:HISTogram:MODE WAVeforms
*OPC?
:MEASure:HISTogram:MIN?
```

We have ensured that at least one trigger has been received, and then we send a block of commands, and after enabling the configured histogram, we wait for all send commands to be complete.

Set Program State When Data Is Not Active (If Possible)

Working with Channels

Many Infiniium oscilloscope setups are complicated. If data is manifest then it is possible to get the worst performance possible by changing state while a measurement is active. For example,

```
:SINGle
:MEASure:RJDJ:STATe ON
:CHANnel1:DISPlay ON   // Now the channel will be analyzed
:MEASure:RJDJ::SOURce CHANnel1   // Channel 1 is our measurement source
:MEASure:RJDJ:EDGE RISing
:MEASure:RJDJ:MODE TIE
*OPC?
```

The sequence begins with data after the `:SINGle` command. Every command sent is enabling long analysis which begins, and then is stopped to handle the subsequent state changes. So the threads that run in the Infiniium software are continually thrashing to handle state change on existing data sets. This entire sequence would be better suited as follows:

```
:CDISplay
:MEASure:RJDJ:STATe ON
:CHANnel1:DISPlay ON   // Now the channel will be analyzed
:MEASure:RJDJ::SOURce CHANnel1   // Channel 1 is our measurement source
:MEASure:RJDJ:EDGE RISing
:MEASure:RJDJ:MODE TIE
:SINGle
*OPC?
```

Here we begin with a clear display. There is no data demanding analysis. Then, we program state, and after programming state, we acquire data. Finally, we wait for everything with one *OPC?, which will return when RJDJ has answers. We minimize the thread thrashing and unnecessary analysis.

Working With Memories

Many times, applications have taken acquired data and loaded this data into a memory. Here, clearing the display is not possible. The goal in this case is to enable the analysis after setting state. For a memory, display ON is equivalent to a Channel display ON and a Single. The above sequence working on a memory would look something like this:

```
:WMEMory1:DISPlay OFF
:MEASure:RJDJ:STATe OFF
:MEASure:RJDJ::SOURce WMEMory1   // Memory 1 is the measurement source
:MEASure:RJDJ:EDGE RISing
:WMEMory1:DISPlay ON
:MEASure:RJDJ:STATe ON
*OPC?
```

The above sequence configures the Infiniium oscilloscope for RJDJ analysis, and only after all parameters are configured, is the most expensive analysis, RJDJ, enabled.

Working With Functions

Functions, like memories, only begin analysis when enabled. Whether it be RJDJ or an FFT, configuration of the intended state should be performed without enabling the function state. Only at the end of a state configuration ought a function be enabled.

10 Remote Acquisition Synchronization

When remotely controlling an oscilloscope with SCPI commands, it is often necessary to know when the oscilloscope has finished the previous operation and is ready for the next SCPI command. The most common example is when an acquisition is started using the :DIG, :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.

Programming Flow

Most remote programming follows these three general steps:

1. Setup the oscilloscope and device under test
2. Acquire a waveform
3. Retrieve results

Setting 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? command.

NOTE **It is not necessary to use the *OPC? command, hard coded waits, or status checking when setting up the oscilloscope.**

After the oscilloscope is configured, it is ready for an acquisition.

Acquiring 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 based on the oscilloscope setup and device under test.
Advantages	▪ No need for polling ▪ Fast 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 ▪ Required polling loop ▪ Required known maximum wait time

Retrieving Results

Once the acquisition is complete, it is safe to retrieve measurements and statistics.

Acquisition Synchronization

Blocking Synchronization

Use the :DIGitize command to start the acquisition. This blocks subsequent queries until the acquisition and processing is complete.

Example

```
// Setup
:TRIGger:MODE EDGE
:TIMebase:SCALE 5e-9

//Acquire
:DIG

//Get results
:MEASure:RISetime?
```

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.

Example

```
TIMEOUT = 1000ms
currentTime = 0ms

// Setup
:STOP; *OPC?    // if not stopped
:ADER?          // clear ADER event

// Acquire
:SINGle

while(currentTime <= TIMEOUT)
{
   if (:ADER? == 1)
   {
      break;
   }
   else
   {
      // Use small wait to prevent excessive
      // queries to the oscilloscope
      wait (100ms)
      currentTime += 100ms
    }
}

//Get results
```

```
if (currentTime < TIMEOUT)
{
   :MEASure:RISetime?
}
```

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 Infiniium real-time 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 visa
import time

# Initialization constants
# ---------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = "TCPIP0::141.121.231.13::hislip0::INSTR"
GLOBAL_TOUT =  10000   # IO time out in milliseconds

TIME_TO_TRIGGER = 10   # Time in seconds
# ---------------------------------------------------------------------
# This is the time until the FIRST trigger event.
#
# While the script calculates a general time out 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.
# ---------------------------------------------------------------------

PROCESSING_TIME = 0   # Time in seconds - this is needed to account
# for additional time after an acquisition to do any processing such
# as FFTs, jitter, etc.
# ---------------------------------------------------------------------
# When using repetitive modes such as average or segmented mode,
# processing happens only once, at the end, on either the final
# averaged waveform or the last segment, thus this only needs to be
# accounted for once.
#
# This time cannot be known ahead of time. One needs to measure this
# time:
#
# To measure PROCESSING_TIME, a second oscilloscope is ideal.
#
#     1.  Connect the trigger output of the oscilloscope to be
#         programmed to an input on a second oscilloscope.  It is
#         often best to use peak detect mode, maximize the signal
#         display intensity, and set the trigger sweep to
#         Normal/Triggered, not auto.
#
#     2.  Feed the oscilloscope to be programmed a 1 MHz (or faster)
#         sine or square wave or other continuous signal such that the
#         trigger condition is always met and the oscilloscope will
#         trigger as fast as it can.
#
#     3.  Set up the oscilloscope to be programmed as it will be used,
#         and put it in the RUNNING state (press the front panel Run
#         key).
#
#     4.  Observe and record the delta time (DT) between trigger
#         output pulses.  There will be some variation, use the
#         largest observed delta time.
#
#     5.  Turn off any processing and again record the smallest delta
#         time of the trigger output pulses.
#
#     6.  Calculate PROCESSING_TIME as
#         DT_largest_processing_ON - DT_smallest_processing_OFF
#
#         For example, with an S-Series oscilloscope, at 67 MPts and
#         20 GSa/s (~3.35 ms time capture) with sinx/x interpolation
#         enabled, it takes on the order of ~10-15 ms per capture to
#         capture and display one channel repetitively.  Once an FFT
#         is enabled, it takes about ~8.2 seconds to repetitively
#         capture, calculate the FFT, and display it, repetitively.
```

```
#         Thus the PROCESSING_TIME should be 8.2 s - 15 ms = ~8.2
#         seconds.  Here, 10 seconds should then be used. (FFTs take a
#         logarithmically long time to compute.)
#
# The SCOPE_ACQUISITION_TIME_OUT calculation pads this by 1.2.
# --------------------------------------------------------------------


# ====================================================================
# Define a simple and fast function utilizing the blocking :DIGitize
# command in conjunction with *OPC?.
# --------------------------------------------------------------------
#
# Benefits of this method:
#
#    -  Fastest, compact, simple
#    -  Works for ALL modes including averaging.
#    -  Don't have to deal with the status registers, which can be
#       confusing.
#
# Drawbacks of this method:
#
#    -  Requires a well-chosen, hard-set timeout that will cover the
#       time to arm, trigger, finish acquisition AND any processing
#       that is already enabled, for example FFTs, math functions,
#       measurements, jitter separation... The script calculates this
#       timeout.
#
#       Please note that for segmented memory mode, any processing
#       would happen only for the final segment.
#
#       Please note that for average acquisition mode, any processing
#       would happen only for the final averaged waveform.
#
#    -  Requires Exception handling and a device clear for a possible
#       timeout (no trigger event)
#
# How it works:
#
#    -  The :DIGitize command is a blocking command, and thus, no
#       other SCPI commands will *execute* until :DIGitize is completely
#       done.  This includes any subsequent processing that is already
#       set up, such as math, jitter separation, 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:
#
#          KsInfiniiumScope.write(":DIGitize")
#          sys.stdout.write("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 = KsInfiniiumScope.query(":DIGitize;*OPC?")
#         sys.stdout.write("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 will need to
#      set a very long time out (that is, TIME_TO_TRIGGER should be
#      very long).
#
#   -  Because it is essentially impossible to know how long
#      additional processing (for example FFT) will take ahead of
#      time, it CAN be beneficial to turn on such things AFTER the
#      signal is acquired.  Further, because much of this processing
#      is done in the Windows OS and memory space, there CAN be a
#      large variation in the post-acquisition processing time.
#      However, read the comments at PROCESSING_TIME for how to
#      actually measure this, and it can be accounted for.
#
#   -  The timeout will need to be (should be) adjusted before and
#      after the :DIGitize operation, though this is not absolutely
#      required.
#
#   -  A :DIGitize can be aborted with a device clear:
#      KsInfiniiumScope.clear()
#
#      The device clear itself can timeout.  Can happen if issued
#      after acquisition done, but scope is still processing a long
#      FFT, for example.  A few (10) seconds is usually plenty.
# ======================================================================
def blocking_method():

    KsInfiniiumScope.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,
    # KsInfiniiumScope
    #
# 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.

    sys.stdout.write("Acquiring signal(s)...\n")
    # Set up a try/except block to catch a possible timeout and exit.
    try:
```

```
        KsInfiniiumScope.query(":DIGitize;*OPC?")
        # Acquire the signal(s) with :DIGitize (blocking) and wait
        # until *OPC? comes back with a one.
        sys.stdout.write("Signal acquired.\n")
    # 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")
        KsInfiniiumScope.clear() # Clear communications interface;
                                 # A device clear also aborts digitize.
        KsInfiniiumScope.close() # Close communications interface
        sys.exit("Exiting script.")

    # Reset timeout back to what it was, GLOBAL_TOUT.
    KsInfiniiumScope.timeout =  GLOBAL_TOUT


# =====================================================================
# Define a function using the non-blocking :SINGle command and polling
# on the Processing Done Event Register
# ---------------------------------------------------------------------
#
# Benefits of this method:
#
#  -  Don't have to worry about interface timeouts.
#  -  Easy to expand to know when the oscilloscope is armed.
#
# Drawbacks of this method:
#
#  -  Slow, as you don't want to poll the oscilloscope too fast.
#
#  -  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.
#
#     If using :PDER? max time out (here MAX_TIME_TO_WAIT) must also
#     account for any processing done (PROCESSING_TIME).
#
#     Max time out (here MAX_TIME_TO_WAIT) must also account for time
#     to arm the oscilloscope and finish the acquisition.
#
#     The script calculates this MAX_TIME_TO_WAIT as
#     SCOPE_ACQUISITION_TIME_OUT.
#
#  -  DOES NOT work for Equivalent time mode.  MUST use the blocking
#     method.
#
# How it works:
#
#  -  Basically, clear the status registers with *CLS.  Initiate the
#     acquisition with the non-blocking :SINGle.  Poll the
#     oscilloscope until the Processing Done Event Register comes
#     back with a 1, meaning that both the acquisition and any
#     enabled processing (FFTs, Math, jitter...) are done.
#
```

```
# Other Notes:
#
#  -  Instead of using the Processing Done Event Register, you could
#     use the Acquisition Done Event Register (see :ADER?).  The
#     benefit here is that one could potentially determine WHEN a
#     trigger occurred, but only within 100 ms (the poll wait time -
#     also need to know how much time acquired after the trigger....)
#     You could also do :ADER?  and then, when that comes back with a
#     1, do :PDER? possibly enabling processing in between...
#
#     Please note that for segmented memory mode, any processing would
#     happen only for the final segment.
#
#     Please note that for average acquisition mode, any processing
#     would happen only for the final averaged waveform.
# ======================================================================
def polling_method():

    MAX_TIME_TO_WAIT = SCOPE_ACQUISITION_TIME_OUT
    # Time in seconds to wait for the oscilloscope to arm, trigger,
    # finish acquisition, and finish any processing.
    #
    # Note that this is NOT a property of the device interface,
    # KsInfiniiumScope, but rather some constant in the script to be
    # used later with the Python module "time", and will be used with
    # time.perf_counter().
    #
    # If using ADER (below), set PROCESSING_TIME = 0.

    # Define completion criterion:
    ACQ_DONE = 1
    ACQ_NOT_DONE = 0

    sys.stdout.write("Acquiring signal(s)...\n")
    # Clear all status registers (set them to 0).  This could be
    # concatenated with the :SINGle command two lines below to speed
    # things up a little, like this ->
    # KsInfiniiumScope.write("*CLS;:SINGle")
    KsInfiniiumScope.write("*CLS")

    # Define acquisition start time. This is in seconds.
    StartTime = time.perf_counter()

    # Begin acquisition with non-blocking :SINGle command.
    KsInfiniiumScope.write(":SINGle")
    # KsInfiniiumScope.write("*CLS;:SINGle")
    # Recommended to concatenate these together for repeated
    # acquisition using this method as it goes slightly faster;
    # consider using method 1 instead if max throughput is desired

    # Immediately ask oscilloscope if it is done with the acquisition
    # and processing.
    Status = int(KsInfiniiumScope.query(":PDER?"))
    # NOTE: :ADER? could also be used, but :ADER does not cover any
    # processing.  If using ADER, set PROCESSING_TIME = 0.
    #
    # NOTE: :PDER? not supported on older Infiniiums. Use :ADER?
```

```
    # instead.

    # ------------------------------------------------------------------
    # For Average mode, MUST use :ADER? (and then PDER if needed) -
    # see "Other Notes" at bottom of this section.
    #
    # This needs to be changed in two places, one above here, and one
    # below....
    # ------------------------------------------------------------------

    # Poll the oscilloscope until Status (:PDER?) is a one. (This is
    # NOT a "Serial Poll.")
    while Status == ACQ_NOT_DONE and \
            (time.perf_counter() - StartTime <= MAX_TIME_TO_WAIT):
        # This loop is never entered if the acquisition completes
        # immediately. Exits if Status == 1 or MAX_TIME_TO_WAIT exceeded
        time.sleep(0.1)   # Pause 100 ms to prevent excessive queries
        Status = int(KsInfiniiumScope.query(":PDER?"))   # Read status
        # Loop exists when Status != NOT_DONE, that is, it exits the
        # loop when it is DONE

    if Status == ACQ_DONE:   # Acquisition fully completed
        sys.stdout.write("Signal acquired.\n")
    else:   # Acquisition failed for some reason
        print("Max wait time exceeded.")
        print("This can happen if there was not enough time to arm the "
 \
              "oscilloscope, there was no trigger event, the " \
              "oscilloscope did not finish acquiring, or the " \
              "processing did not finish.")
        print("Visually check the oscilloscope for a trigger, adjust " \
              "settings accordingly.\n")
        print("Properly closing the oscilloscope connection and " \
              "exiting the script.\n")

        # Always stop the oscilloscope when making any changes.
        KsInfiniiumScope.query(":STOP;*OPC?")
        KsInfiniiumScope.clear() # Clear communications interface
        KsInfiniiumScope.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(KsInfiniiumScope.query("MEASure:VPP? CHANnel1")).strip("\n")
 # The result comes back with 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 = visa.ResourceManager('C:\\Windows\\System32\\visa32.dll')

# Define and open the oscilloscope using the VISA address
KsInfiniiumScope = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
KsInfiniiumScope.timeout = GLOBAL_TOUT

# Clear the instrument bus
KsInfiniiumScope.clear()

# Clear all status registers and errors
KsInfiniiumScope.write("*CLS")

# 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.
# KsInfiniiumScope.query("*RST;*OPC?")   # Resets the oscilloscope

# Always stop the oscilloscope when making any changes.
KsInfiniiumScope.query(":STOP;*OPC?")

# For this example, the oscilloscope will be forced to trigger on the
# (AC input power) LINE voltage so something happens.
# Always use normal trigger sweep, never auto.
KsInfiniiumScope.write(":TRIGger:SWEep TRIGgered")
# This line simply gives the oscilloscope something to trigger on.
KsInfiniiumScope.query(":TRIGger:EDGE:SOURce LINE;*OPC?")

# Clear the display (so you can see the waveform being acquired -
# otherwise, there is no need for this).
KsInfiniiumScope.write(":CDISplay")

# Calculate acquisition timeout/wait time by short, overestimate method
# ---------------------------------------------------------------------

# Need to get some info
HO         = float(KsInfiniiumScope.query(":TRIGger:HOLDoff?"))
SR         = float(KsInfiniiumScope.query(":ACQuire:SRATe:ANALog?"))
N_SAMPLES  = float(KsInfiniiumScope.query(":ACQuire:POINts:ANALog?"))
# Note that the :WAVeform:POINts? command will also return interpolated
# values, so it is not useful.
T_RANGE    = N_SAMPLES / SR
# Note that using the :TIMebase:RANGe? command really only tells us
# what the oscilloscope is on screen, but Infiniium can be set up to
# capture off-screen data.
T_POSITION = float(KsInfiniiumScope.query(":TIMebase:POSition?"))

# Determine if Average mode is on
AVERAGE_MODE_STATE = \
   str(KsInfiniiumScope.query(":ACQuire:AVERage?").strip("\n"))
```

```
if AVERAGE_MODE_STATE == "1":
   N_AVERAGES = \
       float(KsInfiniiumScope.query(":ACQuire:AVERage:COUNt?"))
else:
   N_AVERAGES = 1

# Determine if Segmented Memory or Equivalent time modes are on.
ACQ_MODE = str(KsInfiniiumScope.query(":ACQuire:MODE?").strip("\n"))
ETIME_MULTIPLIER = 1   # For Equivalent Time mode.
# This is a multiplier used as Equivalent time mode builds up over
# numerous acquisitions.
if ACQ_MODE == "SEGM" or ACQ_MODE == "SEGH":
   N_SEGMENTS = \
       float(KsInfiniiumScope.query(":ACQuire:SEGMented:COUNt?"))
elif ACQ_MODE != "ETIM":
   N_SEGMENTS = 1
elif ACQ_MODE == "ETIM":
   N_SEGMENTS = 1
   ETIME_MULTIPLIER = 5   # Total guess.  Few use this mode.
   sys.stdout.write("Timeout calculation of Equivalent time mode "
                    "not thoroughly tested.")

# Calculate timeout from above info.
# Recall that PyVISA timeouts are in ms, so multiply by 1000.
SCOPE_ACQUISITION_TIME_OUT = (float(TIME_TO_TRIGGER)*1.1 +
       float(PROCESSING_TIME)*1.2 +
       (T_RANGE*2.0 + abs(T_POSITION)*2.0 + HO*1.1 +
       float(TIME_BETWEEN_TRIGGERS)*1.1)*N_SEGMENTS*N_AVERAGES*
       ETIME_MULTIPLIER)*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()
do_something_with_data()

polling_method()
do_something_with_data()

# End of Script
# ---------------------------------------------------------------------
KsInfiniiumScope.clear() # Clear communications interface
KsInfiniiumScope.close() # Close communications interface
print("All done.")
```

Single Shot Device Under Test (DUT)

The examples in the previous section (Acquisition Synchronization) assumed 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 as the previous example with the addition of checking for the armed event status.

Example

```
TIMEOUT = 1000ms
currentTime = 0ms

// Setup
:STOP; *OPC?   // if not stopped
:ADER?         // clear ADER event

// Acquire
:SINGle

while(AER? == 0)
{
   wait(100ms)
}

//oscilloscope is armed and ready, enable DUT here

while(currentTime <= TIMEOUT)
{
   if (:ADER? == 1)
   {
      break;
   }
   else
   {
      // Use small wait to prevent excessive
      // queries to the oscilloscope
      wait (100ms)
      currentTime += 100ms
    }
}

//Get results
if (currentTime < TIMEOUT)
{
   :MEASure:RISetime?
}
```

Averaging Acquisition Synchronization

When averaging, it is necessary to know when the average count has been reached. Since an ADER/PDER event occurs for every acquisition in the average count, these commands cannot be used. The :SINGle command does not average.

If it is known that a trigger will occur, a :DIG will acquire the complete number of averages, but if the number of averages is large, it may cause a timeout on the connection.

The example below acquires the desired number of averages and then stops running.

Example

```
AVERAGE_COUNT = 256

:STOP;*OPC?
:TER?
:ACQ:AVERage:COUNt AVERAGE_COUNT
:ACQ:AVERage ON
:RUN

//Assume the oscilloscope will trigger, if not put a check here

while (:WAV:COUNT? < AVERAGE_COUNT)
{
   wait(100ms)
}

:STOP;*OPC?

// Get results
```

11 Analyzing Multiple Acquisitions in Infiniium Offline

When analyzing data in Infiniium Offline, you typically:

1 Save setups and waveforms to a composite file on an Infiniium oscilloscope.
2 Move the composite file to a PC.
3 Open the composite file in Infiniium Offline for analysis there.

However, if your analysis requires waveforms from multiple acquisitions, you can use the :WAVeform:DATA command to load waveforms from multiple acquisitions into Infiniium Offline.

In order to handle analysis involving multiple channels (for example, a math function or measurement on two channels), Infiniium Offline waits for waveform data to be loaded for each channel being used before a "trigger" occurs. Channels being used are either being displayed (after a :CHANnel<N>:DISPlay ON command) or being included in some type of analysis (possibly without being displayed). After the "trigger" occurs, waveform math, measurements, and any other analysis processing takes place.

For Infiniium Offline to know when to process acquisition data sent by :WAVeform:DATA commands, it uses the same "running", "stopped", and "single" run control states that are used in an oscilloscope:

- In the "single" state (after sending the :SINGle command), Infiniium Offline will process a single acquisition.

  Infiniium Offline waits for :WAVeform:SOURce/:WAVeform:DATA on each channel being used, triggers when it gets them, performs analysis, and then goes to the "stopped" state.

- In the "running" state (after sending the :RUN command), Infiniium Offline will process multiple acquisitions.

  Infiniium Offline waits for :WAVeform:SOURce/:WAVeform:DATA on each channel being used, triggers when it gets them, performs analysis, and then repeats, waiting for the next acquisition's waveforms.

- In the "stopped" state (after sending the :STOP command), Infiniium Offline processes each :WAVeform:SOURce/:WAVeform:DATA command as a single acquisition.

  Analyzing multiple acquisitions of a single channel can be done in the "stopped" state. However, when acquisitions include more that one channel, multiple acquisition analysis must be done in the "single" or "running" states.

To allow the "running", "stopped", and "single" run control states in Infiniium Offline, you must enable the **Enable Offline Acquisition** user preference, either in the graphical user interface (**Utilities > User Preferences...**) or by sending the ":HOSTed:EOACq ON" command.

If :WAVeform:DATA is sent to a channel multiple times while waiting for a trigger, only the most recently sent data is used when the trigger occurs.

To find out when analysis is completed and Infiniium Offline is ready to process the next acquisition's set of waveforms, you can use the *OPC? query in the same way you would with an oscilloscope.

When sending acquisitions to Infiniium Offline using the :WAVeform:DATA command, measurement and waveform counts and statistics accumulate the same way they do in an oscilloscope. To reset counts and statistics, you can clear the display (by sending the :CDISplay command). Also, major configuration changes will reset statistics. The behavior is the same as on an actual Infiniium oscilloscope.

For examples of capturing and saving waveforms from multiple acquisitions into files and loading waveform files from multiple acquisitions into Infiniium Offline for analysis, see:

- "Example: Capturing Multiple Acquisitions From an Oscilloscope" on page 214
- "Example: Loading Multiple Acquisitions Into Infiniium Offline" on page 218

See Also

- ":WAVeform:DATA" on page 1675
- ":CHANnel<N>:DISPlay" on page 442
- ":HOSTed:EOACq" on page 723
- ":STOP" on page 282
- ":SINGle" on page 280
- ":RUN" on page 278
- ":WAVeform:SOURce" on page 1712
- ":WAVeform:FORMat" on page 1698

- ":WAVeform:BYTeorder" on page 1669
- "*OPC — Operation Complete" on page 230
- ":CDISplay" on page 263

Example: Capturing Multiple Acquisitions From an Oscilloscope

This Python language VISA.NET example shows capturing and saving waveforms from multiple acquisitions into files for later loading into Infiniium Offline for analysis.

```
#!python3
#
# Multi-acquisition (and channel) capture from Infiniium oscilloscope.
# *********************************************************
# Prerequisites: "pip install pythonnet"
# *********************************************************

# Import Python modules.
# ---------------------------------------------------------
import sys
sys.path.append("C:\\Program Files\\IVI Foundation\\VISA\\Microsoft.NET\
\Framework64\\v2.0.50727\\VISA.NET Shared Components 5.11.0")
import os
import json
import array

# 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 = "TCPIP0::lab-lynx-lp2-3.cos.is.keysight.com::inst0::INSTR"
number_of_acquisitions = 11
potential_waveform_sources = [
 "CHANnel1",
 "CHANnel2",
 "CHANnel3",
 "CHANnel4",
 "CHANnel5",
 "CHANnel6",
 "CHANnel7",
 "CHANnel8",
]
waveform_format = "WORD"   # :WAVeform:DATA supports WORD or BYTE.


# =========================================================
# Check for instrument errors:
# =========================================================
def check_instrument_errors(when):

 errors_found = False
 while True:
```

```
    # Keep reading errors until "No error".
    myScope.WriteLine(":SYSTem:ERRor? STRing")
    error_string = myScope.ReadLine().strip()
    if error_string:   # If there is an error string value.

     if error_string.find("0,", 0, 2) == -1:   # Not "No error".
      errors_found = True
      print(f"ERROR: {error_string}")

     else:   # "No error"
      break

    else:   # :SYSTem:ERRor? STRing should always return string.
     errors_found = True
     print("ERROR: ':SYSTem:ERRor? STRing' empty.")
     break

  if errors_found:
    print(f"Exited because error(s) found when: '{when}'")
    sys.exit(1)


# ===========================================================
# Main program:
# ===========================================================
session = GlobalResourceManager.Open(visa_addr)
session.TimeoutMilliseconds = 360000
myScope = MessageBasedFormattedIO(session)

# Get and display the device's *IDN? string.
myScope.WriteLine("*IDN?")
idn_string = myScope.ReadLine().strip()
print(f"Connected to: {visa_addr},\n   {idn_string}")

# Save oscilloscope setup to file.
myScope.WriteLine(":SYSTem:SETup?")
setup_bytes = myScope.ReadLineBinaryBlockOfByte()
File.WriteAllBytes("multi-acq.set", setup_bytes)
print(f"Setup bytes saved: {len(setup_bytes)}")

# Assume all channel waveforms to save are on.
# Which waveform sources are on?
waveform_sources = []
for waveform_source in potential_waveform_sources:

 myScope.WriteLine(f":STATus? {waveform_source}")
 qresult = myScope.ReadLine().strip()
 if qresult == "1":
  waveform_sources.append(waveform_source)

# Make acquisitions, save waveform data and information.
for acq_num in range(1, number_of_acquisitions + 1):

 # Make a folder to hold the acquisition data and metainfo for
 # each waveform source.
 acq_dir = f"acq{acq_num}"
 if not os.path.exists(acq_dir):
```

```
 os.mkdir(acq_dir)

# Capture an acquisition using :DIGitize.
myScope.WriteLine(":DIGitize")
print(f"------ Acquisition {acq_num}: ------")

# For each waveform source, get data and metainfo.
for waveform_source in waveform_sources:

 # Make a subfolder to hold the acquisition data and metainfo for
 # the waveform source.
 src_dir = f"{acq_dir}/{waveform_source}"
 if not os.path.exists(src_dir):
  os.mkdir(src_dir)

 # Set the waveform source.
 myScope.WriteLine(f":WAVeform:SOURce {waveform_source}")

 # Specify the waveform format.
 myScope.WriteLine(f":WAVeform:FORMat {waveform_format}")

 if waveform_format == "WORD":
  # Use the LSB first byte ordering.
  myScope.WriteLine(":WAVeform:BYTeorder LSBFirst")

 # Get waveform metainfo.
 metainfo_dict = {}
 myScope.WriteLine(":WAVeform:XORigin?")
 x_origin = myScope.ReadLineDouble()
 metainfo_dict["x_origin"] = x_origin

 myScope.WriteLine(":WAVeform:XINCrement?")
 x_increment = myScope.ReadLineDouble()
 metainfo_dict["x_increment"] = x_increment

 myScope.WriteLine(":WAVeform:YORigin?")
 y_origin = myScope.ReadLineDouble()
 metainfo_dict["y_origin"] = y_origin

 myScope.WriteLine(":WAVeform:YINCrement?")
 y_increment = myScope.ReadLineDouble()
 metainfo_dict["y_increment"] = y_increment

 # Save waveform metainfo.
 with open(f"{src_dir}/metainfo.json", "w") as metainfo_file:
  json.dump(metainfo_dict, metainfo_file)

 # Get waveform block data.
 myScope.WriteLine(":WAVeform:STReaming ON")
 myScope.BinaryEncoding = BinaryEncoding.IndefiniteLengthBlockData
 myScope.WriteLine(":WAVeform:DATA?")
 if waveform_format == "BYTE":
  block_data = myScope.ReadLineBinaryBlockOfSByte()
  data = array.array("b", block_data)
 else:   # waveform_format == "WORD":
  block_data = myScope.ReadLineBinaryBlockOfInt16()
  data = array.array("h", block_data)
```

```
    # Save waveform block data.
    with open(f"{src_dir}/block_data.bin", "wb") as block_data_file:
     data.tofile(block_data_file)

    print(f"Waveform data/metainfo saved to: {src_dir}")

  # Check for oscilloscope instrument errors.
  check_instrument_errors("End of program")

  # Close the connection to the instrument.
  session.Dispose()
  print("End of program.")
```

See Also

- "Example: Loading Multiple Acquisitions Into Infiniium Offline" on page 218

Example: Loading Multiple Acquisitions Into Infiniium Offline

This Python language VISA.NET example shows loading waveform files from multiple acquisitions into Infiniium Offline for analysis.

```
#!python3
#
# Multi-acquisition (and channel) load into Infiniium Offline.
# *********************************************************
# Prerequisites: "pip install pythonnet"
# *********************************************************

# Import Python modules.
# ---------------------------------------------------------
import sys
sys.path.append("C:\\Program Files\\IVI Foundation\\VISA\\Microsoft.NET\
\Framework64\\v2.0.50727\\VISA.NET Shared Components 5.11.0")
import os
import re
import json
import array

# 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 = "TCPIP0::localhost::inst0::INSTR"
waveform_format = "WORD"   # :WAVeform:DATA supports WORD or BYTE.


# =========================================================
# Check for instrument errors:
# =========================================================
def check_instrument_errors(when):

 errors_found = False
 while True:
  # Keep reading errors until "No error".
  myScope.WriteLine(":SYSTem:ERRor? STRing")
  error_string = myScope.ReadLine().strip()
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    errors_found = True
    print(f"ERROR: {error_string}")

   else:   # "No error"
    break
```

```
  else:   # :SYSTem:ERRor? STRing should always return string.
   errors_found = True
   print("ERROR: ':SYSTem:ERRor? STRing' empty.")
   break

 if errors_found:
   print(f"Exited because error(s) found when: '{when}'")
   sys.exit(1)


# =========================================================
# Main program:
# =========================================================
session = GlobalResourceManager.Open(visa_addr)
session.TimeoutMilliseconds = 360000
myScope = MessageBasedFormattedIO(session)

# Get and display the device's *IDN? string.
myScope.WriteLine("*IDN?")
idn_string = myScope.ReadLine().strip()
print(f"Connected to: {visa_addr},\n   {idn_string}")

# Clear the display.
myScope.WriteLine(":CDISplay")

# Read setup string from file and restore.
setup_bytes = File.ReadAllBytes("multi-acq.set")
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)}")

# Get all "acq<number>" directories.
acq_dirs = [fname for fname in os.listdir('.')
            if re.match(r"acq[0-9]+", fname)]
acq_dirs.sort(key = lambda x: int(x.split('acq')[1]))

# Enable the "Enable Offline Acquisition" user preference.
myScope.WriteLine(":HOSTed:EOAC ON")

# For each acquisition:
for acq_dir in acq_dirs:
 print(f"------ Acquisition from {acq_dir}: ------")

 # Start an Infiniium Offline single acquisition.
 myScope.WriteLine(":SINGle")

 waveforms = [fname for fname in os.listdir(f"{acq_dir}")]

 # Turn on waveforms.
 for waveform in waveforms:
  myScope.WriteLine(f":{waveform}:DISPlay ON")

 # Get waveform metainfo and write waveform data.
 for waveform in waveforms:
```

```
    src_dir = f"{acq_dir}/{waveform}"

    # Set the waveform source.
    myScope.WriteLine(f":WAVeform:SOURce {waveform}")

    # Specify the waveform format.
    myScope.WriteLine(f":WAVeform:FORMat {waveform_format}")

    if waveform_format == "WORD":
     # Use the LSB first byte ordering.
     myScope.WriteLine(":WAVeform:BYTeorder LSBFirst")

    # Get waveform metainfo.
    with open(f"{src_dir}/metainfo.json", "r") as metainfo_file:
     metainfo_dict = json.load(metainfo_file)
    x_origin = metainfo_dict["x_origin"]
    x_increment = metainfo_dict["x_increment"]
    y_origin = metainfo_dict["y_origin"]
    y_increment = metainfo_dict["y_increment"]

    # Get waveform block data.
    if waveform_format == "BYTE":
     data = array.array("b")
    else:   # waveform_format == "WORD":
     data = array.array("h")
    with open(f"{src_dir}/block_data.bin", "rb") as block_data_file:
     data.frombytes(block_data_file.read())

    # Write waveform to Infiniium Offline.
    myScope.WriteLine(":WAVeform:STReaming ON")
    myScope.Write(":WAVeform:DATA ")
    myScope.Write(f"{x_origin},{x_increment},{y_origin},{y_increment},")
    if waveform_format == "BYTE":
     binary_data = Array[SByte](data)
    else:   # waveform_format == "WORD":
     binary_data = Array[Int16](data)
    myScope.WriteBinary(binary_data)
    myScope.WriteLine()

    print(f"Waveform data/metainfo loaded from: {src_dir}")

   # Wait for operation complete before going to the next acquisition.
   myScope.WriteLine("*OPC?")
   opc_return_string = myScope.ReadLine().strip()

  # Check for oscilloscope instrument errors.
  check_instrument_errors("End of program")

  # Close the connection to the instrument.
  session.Dispose()
  print("End of program.")
```

See Also

- "Example: Capturing Multiple Acquisitions From an Oscilloscope" on page 214

12 * (Common) Commands

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments. Common commands can be received and processed by the oscilloscope, whether they are sent over the remote interface as separate program messages or within other program messages.

Receiving Common Commands

Common commands can be received and processed by the oscilloscope, whether they are sent over the remote interface as separate program messages or within other program messages. If a subsystem is currently selected and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
"ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the oscilloscope, the oscilloscope sets the acquire type, clears the status information, then sets the number of averages without leaving the selected subsystem.

NOTE

Headers and Common Commands.

Headers are not prepended to common commands.

Status Registers

The following two status registers used by common commands have an enable (mask) register. By setting bits in the enable register, you can select the status information for use. Refer to the chapter, "Status Reporting," for a complete discussion of status.

Table 7 Status and Enable Registers

Status Register	Enable Register
Event Status Register	Event Status Enable Register
Status Byte Register	Service Request Enable Register

*CLS – Clear Status

Command

```
*CLS
```

The *CLS command clears all status and error registers.

Example This example clears the status data structures of the oscilloscope.

```
myScope.WriteString "*CLS"
```

See Also

- Chapter 7, “Status Reporting,” starting on page 151 for a complete discussion of status.
- "Example: Blocking and Polling Synchronization" on page 199
- "Example: Checking for Armed Status" on page 176

History Legacy command (existed before version 3.10).

*ESE — Event Status Enable

Command

```
*ESE <mask>
```

The *ESE command sets the Standard Event Status Enable Register bits.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Standard Event Status Register as shown in Table 8.

Example This example enables the User Request (URQ) bit of the Standard Event Status Enable Register. When this bit is enabled and a front-panel key is pressed, the Event Summary bit (ESB) in the Status Byte Register is also set.

```
myScope.WriteString "*ESE 64"
```

Query

```
*ESE?
```

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

Returned Format

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

<mask> An integer, +0 to +255 (the plus sign is also returned), representing a mask value for the bits enabled in the Standard Event Status Register as shown in Table 8.

Example This example places the current contents of the Standard Event Status Enable Register in the numeric variable, varEvent. The value of the variable is printed on the computer's screen.

```
myScope.WriteString "*ESE?"
varEvent = myScope.ReadNumber
Debug.Print FormatNumber(varEvent, 0)
```

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 "0" in the enable register disables the corresponding bit.

Table 8 Standard Event Status Enable Register Bits

Bit	Weight	Enables	Definition
7	128	PON - Power On	Indicates power is turned on.
6	64		Not Used. Permanently set to zero.
5	32	CME - Command Error	Indicates whether the parser detected an error.
4	16	EXE - Execution Error	Indicates whether a parameter was out of range, or was inconsistent with the current settings.
3	8	DDE - Device Dependent Error	Indicates whether the device was unable to complete an operation for device-dependent reasons.

Table 8 Standard Event Status Enable Register Bits (continued)

Bit	Weight	Enables	Definition
2	4	QYE - Query Error	Indicates if the protocol for queries has been violated.
1	2	RQC - Request Control	Indicates whether the device is requesting control.
0	1	OPC - Operation Complete	Indicates whether the device has completed all pending operations.

See Also Refer to Chapter 7, "Status Reporting," starting on page 151 for a complete discussion of status.

History Legacy command (existed before version 3.10).

*ESR? — Event Status Register

Query

```
*ESR?
```

The *ESR? query returns the contents of the Standard Event Status Register. Reading this register clears the Standard Event Status Register, as does a *CLS.

Returned Format

```
<status><NL>
```

<status> An integer, 0 to 255, representing the total bit weights of all bits that are high at the time you read the register.

Example This example places the current contents of the Standard Event Status Register in the numeric variable, varEvent, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*ESR?"
varEvent = myScope.ReadNumber
Debug.Print FormatNumber(varEvent, 0)
```

Table 9 lists each bit in the Event Status Register and the corresponding bit weights.

Table 9 Standard Event Status Register Bits

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
7	128	PON	1 = OFF to ON transition has occurred.
6	64		Not Used. Permanently set to zero.
5	32	CME	0 = no command errors. 1 = a command error has been detected.
4	16	EXE	0 = no execution error. 1 = an execution error has been detected.
3	8	DDE	0 = no device-dependent errors. 1 = a device-dependent error has been detected.
2	4	QYE	0 = no query errors. 1 = a query error has been detected.
1	2	RQC	0 = request control - NOT used - always 0.
0	1	OPC	0 = operation is not complete. 1 = operation is complete.

History Legacy command (existed before version 3.10).

*IDN? — Identification Number

Query `*IDN?`

The *IDN? query returns the company name, oscilloscope model number, serial number, and software version by returning this string:

```
Keysight Technologies,<Model #>,<USXXXXXXXX>,<Rev #>[,<Options>]
```

<Model #> Specifies the model number of the oscilloscope.

<USXXXXXXXX> Specifies the serial number of the oscilloscope. The first four digits and letter are the serial prefix, which is the same for all identical oscilloscopes. The last five digits are the serial suffix, which is assigned sequentially, and is different for each oscilloscope.

<Rev #> Specifies the software version of the oscilloscope, and is the revision number.

<Options> Comma separated list of the installed options.

Returned Format `Keysight Technologies,DSO9404A,USXXXXXXXX,XX.XX.XXXX`

Example This example places the oscilloscope's identification information in the string variable, strIdentify, then prints the identification information to the computer's screen.

```
Dim strIdentify As String   ' Dimension variable.
myScope.WriteString "*IDN?"
strIdentify = myScope.ReadString
Debug.Print strIdentify
```

History Legacy command (existed before version 3.10).

*LRN? — Learn

Query `*LRN?`

The *LRN? query returns a block of data that contains the oscilloscope's current setup. You can store the oscilloscope's setup and send it back to the oscilloscope at a later time. This block of setup data should be sent to the oscilloscope just as it is. It works because of its embedded ":SYST:SET" header.

Returned Format `:SYST:SET <setup><NL>`

NOTE

***LRN? Returns Prefix to Setup Block**

The *LRN? query always returns ":SYST:SET " as a prefix to the setup block. The :SYSTem:HEADer command has no effect on this response.

<setup> This is a definite-length, arbitrary block response specifying the current oscilloscope setup. The block size is subject to change with different firmware revisions.

Example This Python and PyVISA example saves the *LRN? string to a file and then restores the oscilloscope setup from the file.

```
#!python3
# *********************************************************************
# Using the *LRN? string to save and restore the oscilloscope setup.
# *********************************************************************

# Import modules.
# -----------------------------------------------------------
import visa
import sys
import time

# =====================================================================
# Check for instrument errors:
# =====================================================================
def check_instrument_errors():

 while True:
  error_string = Infiniium.query(":SYSTem:ERRor? STRing")
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    print("ERROR: %s." % error_string)
    print("Exited because of error.")
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print("ERROR: :SYSTem:ERRor? STRing returned nothing.")
   print("Exited because of error.")
```

```
    sys.exit(1)

# =========================================================================
# Main program:
# =========================================================================

rm = visa.ResourceManager()
Infiniium = rm.open_resource("TCPIP0::141.121.231.13::hislip0::INSTR")

Infiniium.timeout = 20000
Infiniium.clear()

# Get oscilloscope setup from *LRN? string.
values_list = Infiniium.query_binary_values("*LRN?", datatype='s')
check_instrument_errors()
learn_bytes = values_list[0]

# Save *LRN? string.
f = open("setup_lrn.set", "wb")
f.write(learn_bytes)
f.close()
print("*LRN? string bytes saved: %d" % len(learn_bytes))

# Restore the default setup.
Infiniium.write("*RST")
time.sleep(5)

# Set up oscilloscope by loading previously saved setup.
f = open("setup_lrn.set", "rb")
lrn_bytes = f.read()
f.close()

Infiniium.write_binary_values(":SYSTem:SETup ", lrn_bytes, datatype='B')
check_instrument_errors()

print("*LRN? string bytes restored: %d" % len(lrn_bytes))

Infiniium.close()
```

See Also

- ":SYSTem:SETup" on page 1538

  When HEADers is ON and LONGform is OFF, the :SYSTem:SETup command performs the same function as the *LRN? query. However, *LRN and SETup block setup data are not interchangeable.

- "Definite-Length Block Response Data" on page 123

History Legacy command (existed before version 3.10).

*OPC – Operation Complete

Command `*OPC`

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

Example This example sets the operation complete bit in the Standard Event Status Register when the DIGitize operation is complete.

```
myScope.WriteString ":DIGitize CHANnel1;*OPC"
```

Query `*OPC?`

The *OPC? query places an ASCII character "1" in the oscilloscope's output queue when all pending selected device operations have finished.

Returned Format `1<NL>`

Example This example places an ASCII character "1" in the oscilloscope's output queue when the AUToscale operation is complete. Then the value in the output queue is placed in the numeric variable var"varComplete."

```
myScope.WriteString ":AUToscale;*OPC?"
varComplete = myScope.ReadNumber
Debug.Print FormatNumber(varComplete, 0)
```

The *OPC? query allows synchronization between the computer and the oscilloscope by using the message available (MAV) bit in the Status Byte or by reading the output queue. Unlike the *OPC command, the *OPC? query does not affect the OPC Event bit in the Standard Event Status Register.

See Also

- Chapter 9, “Using the *OPC? (Operation Complete) Query,” starting on page 183

History Legacy command (existed before version 3.10).

*OPT? — Option

Query `*OPT?`

The *OPT? query returns a string with a list of installed options. If no options are installed, the string will have a 0 as the first character.

The length of the returned string may increase as options become available in the future. Once implemented, an option name will be appended to the end of the returned string, delimited by a comma.

Returned Format
```
[002,EZP,EZJ,SDA,LSS,ABD,ABC,ABB,NRD,ERC,AIP,PCI1,ETH,DVI,HDM,B30,CAN,
SA1,DDR]<NL>
```

Table 10 Possible Installed Options and Descriptions

Installed Option	Description
01G	1 GPts
02G	2 GPts
13S	Display Port 1.4 Switch
200	200 MPts
3PC	MIPI 3-Phase Compliance
400	400 MPts
500	500 MPts
AER	MIL-STD 1553/ARINC 429 Protocols
ALT	Altera Probe
AP2	DDR1 Compliance
ASV	Spectrum Visualizer
B30	USB Compliance
BRP	BroadR-Reach Protocol
BRR	BroadR Compliance
BRS	BroadR Switch
BT1	1000BaseT1 Compliance
C3P	CSI3 Protocol
C4C	CAUI-4 Compliance
C4S	CAUI-4 Switch
CAN	CAN/LIN/FlexRay Protocols
CFD	CAN/CAN-FD/LIN/FlexRay Protocols
CFL	CAN/LIN/FlexRay Protocols

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
CFU	CAN/CAN-FD/LIN/FlexRay Protocols Upgrade
CRI	App Remote
D12	Display Port 1.2 Compliance
D13	Display Port 1.4 Compliance
D1S	DDR1 Switch
D2D	DDR2 Debug
D2S	DDR2 Switch
D3D	DDR3 Debug
D3S	DDR3 Switch
D4D	DDR4 Debug
D4S	DDR4 Switch
D9010ASIO	Infiniium Offline - Adv Signal Integrity Software (InfiniiSim Adv/EQ/Crosstalk)
D9010AUTP	Automotive Protocol Decode/Trigger Software (CAN, LIN, CAN-FD, FlexRay ...)
D9010BDLP	Protocol Decode/Trigger Software Bundle (Low Speed, Auto, MIPI, Military)
D9010BSEO	Infiniium Offline - Base Software
D9010DMBA	De-embedding Software (Precision Probe, InfiniiSim Basic)
D9010EKRP	10G/100GBASE-KR 64b/66b and Link Training Decode/Trigger Software
D9010EMBP	Embedded Protocol Decode/Trigger Software (USB2.0, 10/100 ETH, PCIe 2/1 ...)
D9010EXMA	External Mixer Assistant Software
D9010HSPO	Infiniium Offline - High Speed Protocol Software Bundle
D9010JITA	EZJIT Complete - Jitter and Vertical Noise Analysis Software for 9000/S-Series
D9010JITO	Infiniium Offline - EZJIT Complete Software
D9010LSPO	Infiniium Offline - Low Speed Protocol Software Bundle
D9010LSSP	Low Speed Protocol Decode/Trigger Software (I2C, SPI, RS232, I2S, JTAG ...)
D9010MCDP	MIPI CSI and DSI Protocol Decode/Trigger Software (C-PHY and D-PHY)
D9010MILP	Military Protocol Decode/Trigger Software (ARINC 429, MIL-STD 1553, SpaceWire)
D9010MPLP	Low Speed MIPI Protocol Decode/Trigger Software (RFFE, I3C, SPMI)
D9010MPMP	MIPI M-PHY Protocol Decode/Trigger Software (DigRF, LLI, CSI-3, UniPro, UFS, SSIC)
D9010PAMA	Pulse Amplitude Modulation PAM-N Analysis Software

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
D9010PCIP	Advanced PCIe Protocol Decode/Trigger Software (PCIe 4/3/2/1, SATA/SAS)
D9010POWA	Power Integrity Analysis Software
D9010SCNA	InfiniiScan Event Identification Software for 9000/S-Series
D9010UDAA	User Defined Application Software
D9010USBP	USB 3.x Protocol Decode/Trigger Software (USB 3.2 - 5 and 10 Gbps)
D9020ASIA	Advanced Signal Integrity Software (EQ, InfiniiSimAdv, Crosstalk)
D9020AUTP	High Speed Automotive Protocol Decode/Trigger Software (100BASE-T1)
D9020JITA	EZJIT Complete - Jitter and Vertical Noise Analysis Software for V/Z/UXR-Series
D9020SCNA	InfiniiScan Event Identification Software for V/Z/UXR-Series
DD3	DDR3 Compliance
DD4	DDR4 Compliance
DD5	DDR5 Compliance
DDB	DDR Bundle
DDR	DDR2 Compliance
DEA	InfiniiSim Advanced
DEB	InfiniiSim Basic
DEQ	Equalization
DPC	Display Port Compliance
DPS	Display Port Switch
DPT	Display Port Compliance
DRF	DigRF4 Protocol
DRP	DDR Protocol
DTS	Digital Test Apps Switch
DVI	DVI Compliance
E1C	100GBASE-CR4 Ethernet Compliance
E1K	100GBASE-KR4 Ethernet Compliance
E1S	100G-KR4 Ethernet Switch
E2M	2.5G MGBase-T Compliance
E2S	100G-CR4 Ethernet Switch
E4C	40GBASE Ethernet Compliance

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
E4S	40G Ethernet Switch
E5M	5G MGBase-T Compliance
ECU	ECU PHY Compliance
EDP	eDP 1.4 Compliance
EDS	eDP 1.4 Switch
EEE	Energy Efficient Ethernet
EEU	EE Ethernet Upgrade
EGR	100GBASE-KR/CR Ethernet Protocol
EKC	10GBASE-KR Ethernet Compliance
EKR	10GBASE-KR Ethernet Protocol
EKS	10GBASE-KR Ethernet Switch
EMC	eMMC Compliance
ESP	eSPI Protocol
ETH	Gigabit Ethernet Compliance
ETN	10G Ethernet Compliance
ETP	Ethernet Protocol
EUS	eUSB 2.0 Protocol
EZC	EZJIT Complete
EZJ	EZJIT
EZP	EZJIT Plus
FBD	FB DIMM Compliance
FBR	Fibre Channel Compliance
GD3	GDDR3 Compliance
GD4	GDDR4 Compliance
GD5	GDDR5 Compliance
GDR	GDDR Compliance
GMP	Manchester Protocol
H14	HDMI 14 Compliance
H1T	HDMI 1.4 TMDS Compliance
H21	HDMI FRL/TMDS Compliance
H2C	HDMI 2.0 Compliance

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
H2T	HDMI TMDS Compliance
HCS	Hybrid Memory Cube Switch
HDF	Hosted Digitizer Frame
HDM	HDMI Compliance
HDS	HDMI Switch
HMC	Hybrid Memory Cube Compliance
HSI	HSIC USB Compliance
HTE	HDMI 1.4 TMDS Compliance
I3C	MIPI I3C Protocol
ISP	I2S Protocol
JTP	JTAG Protocol
L4D	LPDDR4 Debug
LLI	LLI Protocol
LP2	LPDDR2 Compliance
LP3	LPDDR3 Compliance
LP4	LPDDR4 Compliance
LSS	SPI/I2C Protocols
M3C	MHL 3.0 Compliance
MCC	MIPI C-PHY Compliance
MCP	MIPI C-PHY
MDC	MIPI D-PHY 2.0 Compliance
MDS	MIPI DPHY Switch
MDU	MIPI D-PHY 2.0 Upgrade
MHL	Mobile HD Link Compliance
MHS	MHL Switch
MMS	MIPI MPHY Switch
MP4	MIPI M-PHY 4.1 Compliance
MPH	MIPI M-PHY Compliance
MPI	MIPI D-PHY Compliance
MPP	MIPI D-PHY Protocol
MSO	MSO Upgrade

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
MSS	MOST Switch
MST	MOST Compliance
MYC	User Defined App
NB2	2.5G NBase-T Compliance
NB5	5G NBase-T Compliance
NRD	Noise Reduction
OD2	ONFI-NVDDR2 Compliance
OSA	Oscilloscope Signal Analyzer
P3D	PCI Express 3.0 Protocol
P4D	PCI Express 4.0 Protocol
PC2	PCI Express 2.0 Compliance
PC3	PCI 3.0 Compliance
PC4	PCI Express 4.0 Compliance
PCE	PAM-4 CEI Application
PCI	PCI Express 1.0a Compliance
PEI	PAM-4 CEI 4.0 Application
PEP	PCI Express Protocol
PES	PCI Express Switch
PEU	PAM-4 CEI 4.0 Application Upgrade
PFC	PAM-4 Fibre Channel Application
PHN	Phase Noise
PI2	PAM-4 IEEE 802.3bs/cd Application
PIE	PAM-4 IEEE Application
PIU	PAM-4 IEEE 802.3bs/cd App Upgrade
PM3	PAM-3 Measurement
PM4	PAM-4 Measurement
PRN	PrecisionProbe
PSW	PAM-4 Switch
PWI	PowerIntegrity
QPI	QPI Compliance
QSS	SFP+ Switch

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
RFE	RFFE Protocol
RSP	RS232/UART Protocol
RXP	PCI-E Rx Compliance
RXT	Rx Compliance
RXU	USB Rx Compliance
S12	SAS 12G Compliance
S6G	SAS 6G Compliance
SA1	SATA 1 Compliance
SA2	SATA 2 Compliance
SA4	SAS 4 Compliance
SA6	SATA 3 Compliance
SAS	SAS Compliance
SDA	Serial Data Analysis
SDC	SD Card Compliance
SFP	SFP+ Compliance
SNT	SENT
SPP	SPMI Protocol
SPW	SpaceWire
SSC	SSIC Protocol
SSS	SAS Switch
STP	SATA/SAS Protocol
STS	SATA Switch
SVD	SVID Protocol
SWT	InfiniiScan
T2C	Thunderbolt 2.0 Compliance
T3C	Thunderbolt 3.0 Compliance
TBL	Thunderbolt Compliance
TBS	Thunderbolt Switch
TCC	Thunderbolt 3.0 Compliance
TGS	10G Ethernet Switch
U31	USB 3.1 Protocol

Table 10 Possible Installed Options and Descriptions (continued)

Installed Option	Description
U3P	USB 3.0 Protocol
U3S	USB 3.0 Switch
U4	USB 4 Protocol
UDF	User Def Fn
UDP	User Defined Protocol
UDS	User Defined App Switch
UFP	UFS Protocol
UFS	Universal Flash Storage Compliance
UH2	Ultra HS2 SD Compliance
UHS	Ultra HS SD Compliance
UNI	UniPro Protocol
UPD	USB Power Delivery Protocol
US3	USB 3.1/3.0 Transmitter Compliance
USC	USB 3.1 Compliance
USP	USB 2.0 Protocol
USS	USB3 SuperSpeed Plus Compliance
VX1	Vx1 Compliance
WGN	Waveform Function Generator
WUB	Wireless USB Compliance
XAI	XAUI Compliance
XTK	Crosstalk
XTU	CrosstalkUpgrade

Example This example places all options into the string variable, strOptions, then prints the option name to the computer's screen.

```
Dim strOptions As String
myScope.WriteString "*OPT?"
strOptions = myScope.ReadString
Debug.Print strOptions
```

History Legacy command (existed before version 3.10).

*PSC – Power-on Status Clear

Command

```
*PSC {{ON|1} | {OFF|0}}
```

The *PSC command determines whether or not the SRQ line is set upon the completion of the oscilloscope's boot process. When the *PSC flag is set to 1, the Power On (PON) bit of the Standard Event Status Register is 0 during the boot process. When the *PSC flag is set to 0, the PON bit is set to a 1 during the boot process.

When the *PSC flag is set to 0, the Standard Event Status Enable Register must be set to 128 decimal and the Service Request Enable Register must be set to 32 decimal. This allows the Power On (PON) bit to set the SRQ line when the oscilloscope is ready to receive commands.

NOTE If you are using a LAN interface rather than a GPIB interface, it is not possible to receive the SRQ during the boot process.

Example This example sets the *PSC flag to 0 which sets the SRQ line during the boot process.

```
myScope.WriteString "*PSC 0;*SRE 32;*ESE 128"
```

Query

```
*PSC?
```

The *PSC? query returns the value of the *PSC flag.

Returned Format

```
1<NL>
```

Example This example places the *PSC flag into the integer variable varPscflag.

```
myScope.WriteString "*PSC?"
varPscflag = myScope.ReadNumber
Debug.Print FormatNumber(varPscflag, 0)
```

History Legacy command (existed before version 3.10).

*RCL — Recall

Command

```
*RCL <register>
```

The *RCL command restores the state of the oscilloscope to a setup previously stored in the specified save/recall register. An oscilloscope setup must have been stored previously in the specified register. Registers 0 through 9 are general-purpose registers and can be used by the *RCL command.

<register> An integer, 0 through 9, specifying the save/recall register that contains the oscilloscope setup you want to recall.

Example This example restores the oscilloscope to the oscilloscope setup stored in register 3.

```
myScope.WriteString "*RCL 3"
```

See Also *SAV (Save). An error message appears on the oscilloscope's display if nothing has been previously saved in the specified register.

History Legacy command (existed before version 3.10).

*RST — Reset

Command

```
*RST
```

The *RST command performs a default setup which is the same as pressing the oscilloscope front panel **[Default Setup]** key.

Example

This example resets the oscilloscope to a known state.

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

See Also

- ":SYSTem:PRESet" on page 1536 (where the default values for Infiniium oscilloscope controls are described)

History

Legacy command (existed before version 3.10).

*SAV – Save

Command

```
*SAV <register>
```

The *SAV command stores the current state of the oscilloscope in a save register.

<register> An integer, 0 through 9, specifying the register used to save the current oscilloscope setup.

Example This example stores the current oscilloscope setup to register 3.

```
myScope.WriteString "*SAV 3"
```

See Also *RCL (Recall).

History Legacy command (existed before version 3.10).

*SRE – Service Request Enable

Command

```
*SRE <mask>
```

The *SRE command sets the Service Request Enable Register bits. By setting the *SRE, when the event happens, you have enabled the oscilloscope's interrupt capability. The oscilloscope will then do an SRQ (service request), which is an interrupt.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Service Request Enable Register as shown in Table 11.

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

```
myScope.WriteString "*SRE 16"
```

Query

```
*SRE?
```

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

Returned Format

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

<mask> An integer, 0 to 255, representing a mask value for the bits enabled in the Service Request Enable Register.

Example This example places the current contents of the Service Request Enable Register in the numeric variable, varValue, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*SRE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A "1" in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A "0" disables the bit.

Table 11 Service Request Enable Register Bits

Bit	Weight	Enables
7	128	OPER - Operation Status Register
6	64	Not Used
5	32	ESB - Event Status Bit
4	16	MAV - Message Available
3	8	Not Used
2	4	MSG - Message

Table 11 Service Request Enable Register Bits (continued)

Bit	Weight	Enables
1	2	USR - User Event Register
0	1	TRG - Trigger

History Legacy command (existed before version 3.10).

*STB? – Status Byte

Query `*STB?`

The *STB? query returns the current contents of the Status Byte, including the Master Summary Status (MSS) bit. See Table 12 for Status Byte Register bit definitions.

Returned Format `<value><NL>`

<value> An integer, 0 to 255, representing a mask value for the bits enabled in the Status Byte.

Example This example reads the contents of the Status Byte into the numeric variable, varValue, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*STB?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

In response to a serial poll (SPOLL), Request Service (RQS) is reported on bit 6 of the status byte. Otherwise, the Master Summary Status bit (MSS) is reported on bit 6. MSS is the inclusive OR of the bitwise combination, excluding bit 6, of the Status Byte Register and the Service Request Enable Register. The MSS message indicates that the oscilloscope is requesting service (SRQ).

Table 12 Status Byte Register Bits

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
7	128	OPER	0 = no enabled operation status conditions have occurred 1 = an enabled operation status condition has occurred
6	64	RQS/MSS	0 = oscilloscope has no reason for service 1 = oscilloscope is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	---	0 = not used
2	4	MSG	0 = no message has been displayed 1 = message has been displayed

Table 12 Status Byte Register Bits (continued)

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
1	2	USR	0 = no enabled user event conditions have occurred 1 = an enabled user event condition has occurred
0	1	TRG	0 = no trigger has occurred 1 = a trigger occurred

History Legacy command (existed before version 3.10).

*TRG – Trigger

Command

```
*TRG
```

The *TRG command has the same effect as the Group Execute Trigger message (GET) or RUN command. It acquires data for the active waveform display, if the trigger conditions are met, according to the current settings.

Example This example starts the data acquisition for the active waveform display according to the current settings.

```
myScope.WriteString "*TRG"
```

NOTE

Trigger Conditions Must Be Met

When you send the *TRG command in Single trigger mode, the trigger conditions must be met before the oscilloscope will acquire data.

History Legacy command (existed before version 3.10).

*TST? — Test

Query

```
*TST?
```

The *TST? query causes the oscilloscope to perform a self-test, and places a response in the output queue indicating whether or not the self-test completed without any detected errors. Use the :SYSTem:ERRor command to check for errors. A zero indicates that the test passed and a non-zero indicates the self-test failed.

NOTE

Disconnect Inputs First

You must disconnect all front-panel inputs before sending the *TST? command.

Returned Format

```
<result><NL>
```

<result> 0 for pass; non-zero for fail.

Example This example performs a self-test on the oscilloscope and places the results in the numeric variable, varResults. The program then prints the results to the computer's screen.

```
myScope.WriteString "*TST?"
varResults = myScope.ReadNumber
Debug.Print FormatNumber(varResults, 0)
```

If a test fails, refer to the troubleshooting section of the user's guide.

NOTE

Expanded Error Reporting

The :SELFtest:SCOPETEST command has expanded error reporting. Instead of using *TST?, Keysight recommends that you use the :SELFtest:SCOPETEST command. In either case, be sure you disconnect all front-panel inputs before sending the *TST? command.

History Legacy command (existed before version 3.10).

*WAI – Wait

Command `*WAI`

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

Example `myScope.WriteString "*WAI"`

History Legacy command (existed before version 3.10).

13 : (Root Level) Commands

KEYSIGHT TECHNOLOGIES

Root level commands control many of the basic operations of the oscilloscope that you can select by pressing the labeled keys on the front panel. These commands are always recognized by the parser if they are prefixed with a colon, regardless of the current tree position. After executing a root level command, the parser is positioned at the root of the command tree.

:ADER? — Acquisition Done Event Register

Query `:ADER?`

The :ADER? query reads the Acquisition Done Event Register and returns 1 or 0. After the Acquisition Done Event Register is read, the register is cleared. The returned value 1 indicates an acquisition completed event has occurred and 0 indicates an acquisition completed event has not occurred.

Once the Done bit is set, it is cleared only by doing :ADER? or by sending a *CLS command.

Returned Format `{1 | 0}<NL>`

See Also

- "Example: Blocking and Polling Synchronization" on page 199

History Legacy command (existed before version 3.10).

:AER? — Arm Event Register

Query `:AER?`

The :AER? query reads the Arm Event Register and returns 1 or 0. After the Arm Event Register is read, the register is cleared. The returned value 1 indicates a trigger armed event has occurred and 0 indicates a trigger armed has not occurred.

NOTE

Arm Event Returns

:AER? will allow the Arm Event to return either immediately (if you have armed but not triggered) or on the next arm (if you have already triggered). However, *CLS is always required to get an SRQ again.

Once the AER bit is set, it is cleared only by doing :AER? or by sending a *CLS command.

Returned Format `{1 | 0}<NL>`

See Also

- "Example: Checking for Armed Status" on page 176

History Legacy command (existed before version 3.10).

:ASTate?

Query `:ASTate?`

The :ASTate? query returns the acquisition state:

- ARM — The trigger is armed and the oscilloscope has acquired all of the pre-trigger data.
- TRIG — The trigger condition has occurred and the oscilloscope is acquiring post trigger data.
- ATRIG — The trigger condition has not been met, but the oscilloscope has auto triggered and is acquiring post trigger data.
- ADONE — The acquisition is done, and the data has been processed and is ready to be unloaded.

The :ASTate? query result has more meaning when you first know the run state (see ":RSTate?" on page 277).

Returned Format {ARM | TRIG | ATRIG | ADONE}<NL>

See Also

- ":RSTate?" on page 277

History New in version 4.60.

:ATER? — Auto Trigger Event Register

Query `:ATER?`

The :ATER? query reads the Auto Trigger Event Register and returns 1 or 0. After the Auto Trigger Event Register is read, the register is cleared. The returned value 1 indicates an auto trigger event has occurred and 0 indicates an auto trigger event has not occurred.

Returned Format `{1 | 0}<NL>`

History Legacy command (existed before version 3.10).

:AUToscale

Command `:AUToscale`

The :AUToscale command causes the oscilloscope to evaluate all input waveforms and find the optimum conditions for displaying the waveform. It searches each of the channels for input waveforms and shuts off channels where no waveform is found. It adjusts the vertical gain and offset for each channel that has a waveform and sets the time base on the lowest numbered input channel that has a waveform.

The trigger is found by searching each channel, starting with channel 4, then channel 3, channel 2, and channel 1, until a trigger waveform is detected. If waveforms cannot be found on any vertical input, the oscilloscope is returned to its former state.

Autoscale sets the following:

- Channel Display, Scale, and Offset
- Trigger Sweep, Mode, Edge, Source, Level, Slope, Hysteresis, and Holdoff
- Acquisition Sampling Rate and Memory Depth
- Time Base Scale and Position
- Marker Mode Set to Measurement
- Resets Acquisition Completion Criteria to 90%

Autoscale turns off the following:

- Measurements on sources that are turned off
- Functions
- Windows
- Memories
- InfiniiSim

Autoscale does not turn off:

- PrecisionProbe/PrecisionCable

No other controls are affected by Autoscale.

Example This example automatically scales the oscilloscope for the input waveform.

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

History Legacy command (existed before version 3.10).

:AUToscale:CHANnels

Command

```
:AUToscale:CHANnels {ALL | DISPlayed}
```

The :AUToscale:CHANnels command selects whether to apply autoscale to all of the input channels or just the input channels that are currently displayed.

Example This example automatically scales only the displayed channels.

```
myScope.WriteString ":AUToscale:CHANnels DISPlayed"
```

Query

```
:AUToscale:CHANnels?
```

The :AUToscale:CHANnels? query returns the selected channels setting.

Returned Format

```
[:AUToscale:CHANnels] {ALL | DISP}<NL>
```

History Legacy command (existed before version 3.10).

:AUToscale:PLACement

Command

```
:AUToscale:PLACement {STACk | SEParate | OVERlay}
```

The :AUToscale:PLACement command controls how the waveforms are displayed on the oscilloscope when the autoscale command is used. If Stack is chosen then each waveform's amplitude is decreased and then the waveforms are offset so each takes up a different vertical portion of the screen. This makes it easier to view them, but decreases the accuracy of any measurements performed on the waveforms because they no longer take up the full dynamic range of th ADC (analog to digital converter). If Separate is chosen then the screen is divided into the same number of grids that there are waveforms (for example, if three waveforms are displayed then the screen will be divided into three grids). Each grid represents the full dynamic range of the ADC so this choice maximizes measurement accuracy while still separating the waveforms so they are easy to see. If the Overlay option is chosen then the waveforms are displayed on top of each other. This maximizes measurement accuracy, but can making viewing difficult.

Example This example automatically overlays the waveforms after an autoscale.

```
myScope.WriteString ":AUToscale:OVERlay ON"
```

Query

```
:AUToscale:PLACement?
```

History Legacy command (existed before version 3.10).

:AUToscale:VERTical

Command `:AUToscale:VERTical <source>`

The :AUToscale:VERTical command autoscales the vertical position and scaling for the corresponding channel without changing anything else (for example, trigger or timebase settings).

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C>}

<N> An integer, 1 to the number of analog input channels.

<R> An integer, 1-4.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

Example This example automatically autoscales the vertical position and scale for the waveform on Channel 1.

```
myScope.WriteString ":AUToscale:VERTical CHAN1"
```

NOTE If you are using software 2.10 or earlier, the command syntax is (lower-case "t" in "vertical"):

AUToscale:VERTical <CHANnel 1 | CHANnel 2 | CHANnel 3 | CHANnel 4>

History Legacy command (existed before version 3.10).

:BEEP

Command

```
:BEEP <frequency>,<duration>
```

The :BEEP command makes the oscilloscope beep at a defined frequency and duration.

<frequency> A real number representing frequency of beep in Hertz.

<duration> A real number representing duration of beep in milliseconds.

Example This example will create a beep at 1000 Hz for 500 ms.

```
myScope.WriteString ":BEEP 1000,500"
```

History Legacy command (existed before version 3.10).

:BLANk

Command

```
:BLANk {CHANnel<N> | BUS<B> | DIFF<D> | COMMonmode<C>
        | DIGital<M> | FUNCtion<F> | HISTogram | WMEMory<R>
        | MTRend | MSPectrum | EQUalize<L> | POD<P> | ALL}
```

The :BLANk command turns off an active channel, function, histogram, waveform memory, measurement trend, measurement spectrum, or Feed-Forward Equalized waveform. The :VIEW command turns them on.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<M> An integer, 0-15.

<F> An integer, 1-16.

<R> An integer, 1-4.

<L> An integer, 1-4.

<P> An integer, 1-2.

Example This example turns off channel 1.

```
myScope.WriteString ":BLANk CHANnel1"
```

See Also

- ":VIEW" on page 287
- ":STATus?" on page 281

History Legacy command (existed before version 3.10).

:CDISplay

Command `:CDISplay`

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all of the data in active channels and functions is erased; however, new data is displayed on the next acquisition. Waveform memories are not erased.

Example This example clears the oscilloscope display.

```
myScope.WriteString ":CDISplay"
```

History Legacy command (existed before version 3.10).

:DIGitize

Command

```
:DIGitize [CHANnel<N> | DIGital<M> | DIFF<D> | COMMonmode<C>
    | POD<P>][,...]
```

The :DIGitize command invokes a special mode of data acquisition that is more efficient than using the :RUN command. This command initializes the selected channels or functions, then acquires them according to the current oscilloscope settings. When all waveforms are completely acquired, the oscilloscope is stopped. The waveform completion criteria is set with the ":ACQuire:COMPlete" command.

If you specify channel parameters, then these are the only waveforms acquired and the display waveforms of the specified channels are turned off.

NOTE

Full Range of Measurement and Math Operators are Available

Even though digitized waveforms are not displayed, you may perform the full range of measurement and math operators on them.

NOTE

Channel parameters are not supported in a MultiScope system because acquisitions require at least one channel per connected frame. Only the parameterless version of :DIGitize is supported in a MultiScope system.

If you use the :DIGitize command with no parameters, the digitize operation is performed on the channels that are being displayed in the Infiniium waveform viewing area. In this case, the display state of the acquired waveforms is not changed after the :DIGitize command is completed. Because the command executes more quickly without parameters, this form of the command is useful for repetitive measurement sequences. You can also use this mode if you want to view the digitize results because the display state of the digitized waveforms is not affected.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

Example This example acquires data on channel 1.

```
myScope.WriteString ":DIGitize CHANnel1"
```

The ACQuire subsystem commands set up conditions such as COUNt for the next :DIGitize command. The WAVeform subsystem commands determine how the data is transferred out of the oscilloscope, and how to interpret the data.

See Also

- Chapter 8, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 181
- "Example: Blocking and Polling Synchronization" on page 199
- "Example: Checking for Armed Status" on page 176
- See the Chapter 52, “Example Programs,” starting on page 2027 for examples of how to use :DIGitize and its related commands.

History Legacy command (existed before version 3.10).

:DISable DIGital

Command

```
:DISable DIGital
```

The :DISable DIGital command disables the digital channels 0-15.

Example This example will disable the digital channels.

```
myScope.WriteString ":DISable DIGital"
```

History Legacy command (existed before version 3.10).

:ENABle DIGital

Command `:ENABle DIGital`

The :ENABle DIGital command enables the digital channels 0-15.

Example This example will enable the digital channels.

```
myScope.WriteString ":ENABle DIGital"
```

History Legacy command (existed before version 3.10).

:MODel?

Query `:MODel?`

The :MODel? query returns the model number for the oscilloscope.

Returned Format A six-character alphanumeric model number in quotation marks. Output is determined by header and longform status as in Table 13.

Table 13 MODel? Returned Format

:SYSTem:HEADer		:SYSTem:LONGform		Response (for example)
ON	OFF	ON	OFF	
	X		X	DSO90804A
	X	X		DSO90804A
X			X	:MOD DSO90804A
X		X		:MODEL DSO90804A

Example This example places the model number in a string variable, strModel, then prints the contents of the variable on the computer's screen.

```
Dim strModel As String   ' Dimension variable.
myScope.WriteString ":MODel?"
strModel = myScope.ReadString
Debug.Print strModel
```

History Legacy command (existed before version 3.10).

:MTEEnable — Mask Test Enable Register

Command `:MTEEnable <enable_mask>`

The :MTEEnable command is used to set bits in the Mask Test Enable Register.

<enable_mask> The <enable_mask> is a 16-bit signed decimal value that enables the following bits of the Mask Test Event Register:

Bit 0	Mask Test Complete
Bit 1	Mask Test Fail
Bit 2	Mask Low Amplitude
Bit 3	Mask High Amplitude
Bit 4	Mask Align Complete
Bit 5	Mask Align Fail
Bits 6-14	are not used

Query `:MTEEnable?`

The :MTEEnable? query returns the value stored in the Mask Test Enable Register.

Returned Format `[:MTEEnable] <enable_mask>`

Example Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
myScope.WriteString "MTEEnable 2"
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

History Legacy command (existed before version 3.10).

:MTERegister? — Mask Test Event Register

Query `:MTERegister?`

The :MTERegister? query returns the value stored in the Mask Test Event Register. The bits stored in the register have the following meanings:

Bit 0	Mask Test Complete bit is set whenever the mask test is complete.
Bit 1	Mask Test Fail bit is set whenever the mask test failed.
Bit 2	Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.
Bit 3	Mask High Amplitude bit is set whenever the signal is above the mask amplitude.
Bit 4	Mask Align Complete bit is set whenever the mask align is complete.
Bit 5	Mask Align Fail bit is set whenever the mask align failed.

The Mask Test Event Register is read and cleared by the MTERegister? query. The register output is enabled or disabled using the mask value supplied with the MTEEnable command.

Returned Format `0-63 decimal value.`

NOTE

Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit

Mask Test Event 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 in the Operation Status Register.

History Legacy command (existed before version 3.10).

:OPEEnable — Operation Status Enable

Command

```
:OPEEnable <mask>
```

<mask> The decimal weight of the enabled bits.

The :OPEEnable command sets a mask in the Operation Status Enable register. Each bit that is set to a "1" enables that bit to set bit 7 in the status byte register, and potentially causes an SRQ to be generated. Bit 5, Wait for Trig is used. Other bits are reserved.

Query

```
:OPEEnable?
```

The query returns the current value contained in the Operation Status Enable register as a decimal number.

Returned Format

```
[OPEEnable] <value><NL>
```

History Legacy command (existed before version 3.10).

:OPERegister? — Operation Status Register

Query `:OPERegister?`

The :OPERegister? query returns the value contained in the Operation Status Register as a decimal number. This register contains the WAIT TRIG bit (bit 5) and the OVLR bit (bit 11).

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed. The OVLR bit is set by the Overload Event Register.

Returned Format `<value><NL>`

See Also

- "Example: Checking for Armed Status" on page 176

History Legacy command (existed before version 3.10).

:OVLRegister?

Query `:OVLRegister?`

The :OVLRegister? query returns the value stored in the Overload Event Register.

The integer value returned by this query represents the channels as follows:

- Bit 0 - Channel 1
- Bit 1 - Channel 2
- Bit 2 - Channel 3
- Bit 3 - Channel 4
- Bit 4 - Channel 5
- Bit 5 - Channel 6
- Bit 6 - Channel 7
- Bit 7 - Channel 8
- Bits 12 - Probe Power Fault
- Bits 8-11 and 13-15 are not used and are set to zero (0)

Returned Format `<value><NL>`

History Legacy command (existed before version 3.10).

Version 11.00: The Overload Event Register is widened to 16 bits, making room for up to 8 analog input channels and one additional Probe Power Fault bit.

:PDER? — Processing Done Event Register

Query `:PDER?`

The :PDER? query reads the Processing Done Event Register and returns 1 or 0. After the Processing Done Event Register is read, the register is cleared. The returned value 1 indicates indicates that all math and measurements are complete and 0 indicates they are not complete. :PDER? is non-blocking.

:PDER? can be used in place of :ADER?.

Returned Format `{1 | 0}<NL>`

See Also

- "Example: Blocking and Polling Synchronization" on page 199

History Legacy command (existed before version 3.10).

:PRINt

Command `:PRINt`

The :PRINt command outputs a copy of the screen to a printer or other device destination specified in the HARDcopy subsystem. You can specify the selection of the output and the printer using the HARDcopy subsystem commands.

Example This example outputs a copy of the screen to a printer or a disk file.

```
myScope.WriteString ":PRINt"
```

History Legacy command (existed before version 3.10).

:RECall:SETup

Command

```
:RECall:SETup <setup_memory_num>
```

<setup_memory_num> Setup memory number, an integer, 0 through 9.

The :RECall:SETup command recalls a setup that was saved in one of the oscilloscope's setup memories. You can save setups using either the :STORe:SETup command or the front panel.

Examples This command recalls a setup from setup memory 2.

```
myScope.WriteString ":RECall:SETup 2"
```

History Legacy command (existed before version 3.10).

:RSTate?

Query `: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.

Returned Format {RUN | STOP | SING}<NL>

See Also

- ":ASTate?" on page 255

History New in version 4.60.

:RUN

Command `:RUN`

The :RUN command starts the oscilloscope running. When the oscilloscope is running, it acquires waveform data according to its current settings. Acquisition runs repetitively until the oscilloscope receives a :STOP command, or until there is only one acquisition if Trigger Sweep is set to Single. However, the :TRIGger:SWEep SINGle should not be used in new programs. The :SINGle command should be used instead to acquire a single acquisition.

Example This example causes the oscilloscope to acquire data repetitively.

```
myScope.WriteString ":RUN"
```

History Legacy command (existed before version 3.10).

:SERial — Serial Number

Command
```
:SERial {<serial_number>}
```

The :SERial command sets the serial number of the oscilloscope. A serial number was entered in your oscilloscope by Keysight Technologies before it was shipped to you. Therefore, setting the serial number is not normally required unless the oscilloscope is serialized for a different application.

The oscilloscope's serial number is part of the string returned for the *IDN? query described in the Common Commands chapter.

<serial _number> A ten-character alphanumeric serial number enclosed with quotation marks.

Example This example sets the serial number for the oscilloscope to "US12345678".

```
myScope.WriteString ":SERial ""US12345678"""
```

Query
```
:SERial?
```

The query returns the current serial number string for the oscilloscope.

Returned Format
```
[:SERial] US12345678
```

Example This example places the serial number for the oscilloscope in the string variable strSerial, then prints the contents of the variable to the computer's screen.

```
Dim strSerial As String   ' Dimension variable.
myScope.WriteString ":SERial?"
strSerial = myScope.ReadString
Debug.Print strSerial
```

History Legacy command (existed before version 3.10).

:SINGle

Command `:SINGle`

The :SINGle command causes the oscilloscope to make a single acquisition when the next trigger event occurs. However, this command does not set the :TRIGger:SWEep to SINGle.

Example This example sets up the oscilloscope to make a single acquisition when the next trigger event occurs.

```
myScope.WriteString ":SINGle"
```

See Also
- ":TRIGger:SWEep" on page 1574
- "Example: Blocking and Polling Synchronization" on page 199
- "Example: Checking for Armed Status" on page 176

History Legacy command (existed before version 3.10).

:STATus?

Query

```
:STATus? <source>
```

The :STATus? query shows whether the specified waveform is on or off. A return value of 1 means on and a return value of 0 means off.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | HISTogram | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR | BUS<B> | DIGital<M> | POD<L>}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Returned Format

```
[:STATus] {0 | 1}<NL>
```

Example This example returns and prints the current status of channel 1.

```
myScope.WriteString ":STATus? CHANnel1"
strCurrent = myScope.ReadString
Debug.Print strCurrent
```

See Also

- ":BLANk" on page 262
- ":VIEW" on page 287

History Legacy command (existed before version 3.10).

:STOP

Command

```
:STOP
```

The :STOP command causes the oscilloscope to stop acquiring data.

Sending one :STOP command allows the current acquisition cycle to exit post-capture analysis of the last acquisition without aborting. This is known as a *soft stop*.

Sending two or more :STOP commands in sequence aborts the current analysis and stops as quickly as possible.

To restart the acquisition, use the :RUN or :SINGle command.

Example This example stops the current data acquisition.

```
myScope.WriteString ":STOP"
```

History Legacy command (existed before version 3.10).

:STORe:JITTer

Command

```
:STORe:JITTer <file_name>
```

The :STORe:JITTer command saves all of the RJ/DJ jitter measurement data to the specified file name. The file that is created has a header section followed by the RJ/DJ measurement results section. After the RJ/DJ measurement results section is the data for each of the measurements. Each data section has a header showing what the measurement data is that follows.

<file_name> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example stores the RJ/DJ jitter measurements to a file.

```
myScope.WriteString _
":STORe:JITTer ""C:\Users\Public\Documents\Infiniium\jitter"""
```

History Legacy command (existed before version 3.10).

:STORe:SETup

Command

```
:STORe:SETup <setup_memory_num>
```

<setup_memory_num> Setup memory number, an integer, 0 through 9.

The :STORe:SETup command saves the current oscilloscope setup in one of the setup memories.

Example This example stores the current oscilloscope setup to setup memory 0.

```
myScope.WriteString ":STORe:SETup 0"
```

History Legacy command (existed before version 3.10).

:STORe:WAVeform

Command

```
:STORe:WAVeform {{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | EQUalized<L> | XT<X> | MTRend | MSPectrum},
   {WMEMory<R>}}
```

The :STORe:WAVeform command copies a channel, function, stored waveform, measurement trend, or measurement spectrum to a waveform memory. The parameter preceding the comma specifies the source and can be any channel, function, or waveform memory. The parameter following the comma is the destination, and can be any waveform memory.

The :WAVeform:VIEW command determines the view of the data being stored.

The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<L> An integer, 1-4.

<X> An integer, 1-4, identifying the crosstalk waveform.

Example This example copies channel 1 to waveform memory 3.

```
myScope.WriteString ":STORe:WAVeform CHANnel1,WMEMory3"
```

History Legacy command (existed before version 3.10).

:TERegister? — Trigger Event Register

Query `:TERegister?`

The :TERegister? query reads the Trigger Event Register. A "1" is returned when an acquisition is complete. A "0" is returned when an acquisition has not completed.

The autotrigger does not set this register.

The register is set to a value of 1 only when the waveform meets the trigger criteria and the acquisition completes.

Returned Format `{1 | 0}<NL>`

Example This example checks the current status of the Trigger Event Register, places the status in the string variable, strCurrent, then prints the contents of the variable to the computer's screen.

```
Dim strCurrent As String   ' Dimension variable.
myScope.WriteString ":TERegister?"
strCurrent = myScope.ReadString
Debug.Print strCurrent
```

Once this bit is set, you can clear it only by reading the register with the :TERegister? query, or by sending a *CLS common command. After the Trigger Event Register is read, it is cleared.

History Legacy command (existed before version 3.10).

:VIEW

Command

```
:VIEW {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
      | HISTogram | WMEMory<R> | MSTrend | MSPectrum | BUS<B>
      | DIGital<M> | POD<P>}
```

The :VIEW command turns on a channel, function, histogram, or waveform memory. The :BLANk command turns them off.

The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<B> An integer, 1-4.

<M> An integer, 0-15.

<P> An integer, 1-2.

Example This example turns on channel 1.

```
myScope.WriteString ":VIEW CHANnel1"
```

See Also

- ":BLANk" on page 262
- ":STATus?" on page 281

History Legacy command (existed before version 3.10).

14 :ACQuire Commands

The ACQuire subsystem commands set up conditions for executing a :DIGitize root level command to acquire waveform data. The commands in this subsystem select the type of data, the number of averages, and the number of data points.

:ACQuire:ADC:CLIPped:CLEar

Command `:ACQuire:ADC:CLIPped:CLEar`

The :ACQuire:ADC:CLIPped:CLEar command clears clipping status on all channels at once.

This is a non-blocking command that is useful if there was a change in the device-under-test that may have caused a temporary clipped condition that needs to be cleared before checking for a real clipped condition.

See Also

- ":CHANnel<N>:ADC:CLIPped" on page 436
- ":DISPlay:CLIPped" on page 564

History New in version 10.10.

:ACQuire:ADCRes

Command

```
:ACQuire:ADCRes {BITS10 | BITS11 | BITS12 | BITS13 | BITS14 | BITS15
    | BITS16 | BITS16_4 | BITS16_2}
```

In earlier Infiniium oscilloscopes, the High Resolution acquisition mode uses uncorrected data (that is, no response or interleave correction) and a boxcar average is performed to produce high-resolution output of each acquisition.

In the MXR/EXR-Series oscilloscopes, the analog-to-digital converter (ADC) can perform both response and interleave correction on the ADC input stream while bandwidth-limiting the data to produce corrected, high-resolution data with an effective bits performance that is superior to the earlier high-resolution method.

The starting sample rate for High Resolution Oscilloscope (HRO) is 6.4 GSa/s and goes down to 50 MSa/s as shown the following table.

ADC Resolution	HRO GSa/s	Bandwidth	Effective Bits
BITS11	6.4	2.56 GHz	10.7
BITS12	3.2	1.28 GHz	11.2
BITS13	1.6	640 MHz	11.7
BITS14	0.800	320 MHz	12.2
BITS15	0.400	160 MHz	12.7
BITS16 (80 MHz)	0.200	80 MHz	13.2
BITS16_4 (40 MHz)	0.100	40 MHz	13.7
BITS16_2 (20 MHz)	0.050	20 MHz	14.2

NOTE

Selecting an ADC Resolution of 11 Bits or greater locks in the ADC sample rate and bandwidth according to the above table.

If the time/div and memory depth settings are such that there are more samples than can fit into memory, simple decimation (or peak detect in that acquisition mode) is used, not additional high-resolution decimation (where samples in a time bin are averaged).

In earlier Infiniium oscilloscopes, the High Resolution and Peak Detect acquisition modes were mutually exclusive. In the MXR/EXR-Series oscilloscopes, you can still select between the **Normal** and **Peak Detect** acquisition modes when high-resolution options (between 11 and 16 bits) are selected.

For more information on the ADC Resolution settings, see the MXR/EXR-Series oscilloscopes online help.

Example This example sets the bit resolution setting to a minimum of 14 bits.

```
myScope.WriteString ":ACQuire:ADCRes BITS14"
```

Query `:ACQuire:ADCRes?`

The :ACQuire:ADCRes? query returns the bit resolution setting.

Returned Format

```
[:ACQuire:ADCRes] {BITS10 | BITS11 | BITS12 | BITS13 | BITS14 | BITS15
    | BITS16 | BITS16_4 | BITS16_2}<NL>
```

Example This example places the current bit resolution setting in the string variable, strBitRes, then prints the contents of the variable to the computer's screen.

```
Dim strBitRes As String    ' Dimension variable.
myScope.WriteString ":ACQuire:ADCRes?"
strBitRes = myScope.ReadString
Debug.Print strBitRes
```

See Also

- ":ACQuire:MODE" on page 308
- ":ACQuire:SRATe[:ANALog] — Analog Sample Rate" on page 323
- ":TIMebase:SCALe" on page 1549
- ":TIMebase:RANGe" on page 1543

History New in version 11.00.

:ACQuire:AVERage

Command `:ACQuire:AVERage {{ON|1} | {OFF|0}}`

The :ACQuire:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :ACQuire:AVERage:COUNt command described next.

Averaging is not available in PDETect mode.

The :MTESt:AVERage command performs the same function as this command.

Example This example turns averaging on.

```
myScope.WriteString ":ACQuire:AVERage ON"
```

Query `:ACQuire:AVERage?`

The :ACQuire:AVERage? query returns the current setting for averaging.

Returned Format `[:ACQuire:AVERAGE] {1|0}<NL>`

Example This example places the current settings for averaging into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":ACQuire:AVERage?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:ACQuire[:AVERage]:COUNt

Command

```
:ACQuire[:AVERage]:COUNt <count_value>
```

The :ACQuire[:AVERage]:COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :ACQuire[:AVERage]:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :MTESt:AVERage:COUNt command performs the same function as this command.

<count_value> An integer, 2 to 10,486,575, specifying the number of data values to be averaged.

Example This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :ACQuire:COMPlete command.

```
myScope.WriteString ":ACQuire:COUNt 16"
```

Query

```
:ACQuire[:AVERage]:COUNt?
```

The :ACQuire[:AVERage]:COUNt? query returns the currently selected count value.

Returned Format

```
[:ACQuire[:AVERage]:COUNt] <value><NL>
```

<value> An integer, 2 to 10,486,575, specifying the number of data values to be averaged.

Example This example checks the currently selected count value and places that value in the string variable, strResult. The program then prints the contents of the variable to the computer's screen.

```
Dim strResult As String
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:AVERage:COUNt?"
strResult = myScope.ReadString
Debug.Print strResult
```

History Legacy command (existed before version 3.10).

:ACQuire:BANDwidth

Command

```
:ACQuire:BANDwidth {AUTO | MAX | <bandwidth>}
```

The :ACQuire:BANDwidth command changes the bandwidth frequency control for the acquisition system.

- AUTO — The bandwidth is automatically selected based on the sample rate setting in order to make a good a trade-off between bandwidth, noise, and aliasing.
- MAX — Sets the oscilloscope to the hardware bandwidth limit and disables the bandwidth filter.
- <bandwidth> — a real number representing the bandwidth of the bandwidth filter whose range of values depends on the model number of your oscilloscope.

Query

```
:ACQuire:BANDwidth?
```

The :ACQuire:BANDwidth? query returns the bandwidth setting of the bandwidth control.

Returned Format

```
[:ACQuire:BANDwidth] <bandwidth><NL>
```

History New in version 3.10.

Version 4.00: Added a MAX option for selecting the maximum bandwidth.

:ACQuire:BANDwidth:FRAMe?

Query `:ACQuire:BANDwidth:FRAMe?`

The :ACQuire:BANDwidth:FRAMe? query returns the maximum bandwidth associated with oscilloscope model.

Returned Format `<bandwidth><NL>`

`<bandwidth> ::= max. BW of oscilloscope model`

History New in version 5.20.

:ACQuire:BANDwidth:TESTLIMITS?

Query `:ACQuire:BANDwidth:TESTLIMITS?`

The :ACQuire:BANDwidth:TESTLIMITS? query returns the oscilloscope's acquisition bandwidth maximum and minimum limits.

Returned Format `<num_parms>,<<type>><min>:<max><NL>`

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<numeric>" for this query.

<min> Lower bandwidth limit value.

<max> Upper bandwidth limit value.

See Also

- ":ACQuire:BANDwidth" on page 296
- ":ACQuire:BANDwidth:FRAMe?" on page 297
- ":ACQuire:POINts:TESTLIMITS?" on page 314
- ":ACQuire:SRATe:TESTLIMITS?" on page 327

History New in version 5.60.

:ACQuire:COMPlete

Command

```
:ACQuire:COMPlete <percent>
```

The :ACQuire:COMPlete command specifies how many of the data point storage bins (time buckets) in the waveform record must contain a waveform sample before a measurement will be made. For example, if the command :ACQuire:COMPlete 60 has been sent, 60% of the storage bins in the waveform record must contain a waveform data sample before a measurement is made.

- If :ACQuire:AVERage is set to OFF, the oscilloscope only needs one value per time bucket for that time bucket to be considered full.
- If :ACQuire:AVERage is set to ON, each time bucket must have *n* hits for it to be considered full, where *n* is the value set by :ACQuire:AVERage:COUNt.

Due to the nature of real time acquisition, 100% of the waveform record bins are filled after each trigger event, and all of the previous data in the record is replaced by new data when :ACQuire:AVERage is off. Hence, the complete mode really has no effect, and the behavior of the oscilloscope is the same as when the completion criteria is set to 100% (this is the same as in PDETect mode). When :ACQuire:AVERage is on, all of the previous data in the record is replaced by new data.

The range of the :ACQuire:COMPlete command is 0 to 100 and indicates the percentage of time buckets that must be full before the acquisition is considered complete. If the complete value is set to 100%, all time buckets must contain data for the acquisition to be considered complete. If the complete value is set to 0, then one acquisition cycle will take place. Completion is set by default setup or *RST to 90%. Autoscale changes it to 100%.

<percent> An integer, 0 to 100, representing the percentage of storage bins (time buckets) that must be full before an acquisition is considered complete.

Example This example sets the completion criteria for the next acquisition to 90%.

```
myScope.WriteString ":ACQuire:COMPlete 90"
```

Query

```
:ACQuire:COMPlete?
```

The :ACQuire:COMPlete? query returns the completion criteria.

Returned Format

```
[:ACQuire:COMPlete] <percent><NL>
```

<percent> An integer, 0 to 100, representing the percentage of time buckets that must be full before an acquisition is considered complete.

Example This example reads the completion criteria and places the result in the variable, varPercent. Then, it prints the content of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:COMPlete?"
varPercent = myScope.ReadNumber
Debug.Print FormatNumber(varPercent, 0)
```

History Legacy command (existed before version 3.10).

:ACQuire:COMPlete:STATe

Command `:ACQuire:COMPlete:STATe {{ON | 1} | {OFF | 0}}`

The :ACQuire:COMPlete:STATe command specifies whether function averaging should complete before measurements are made.

- ON — The oscilloscope waits for a function average to complete before measurements are made. This lets you filter out noise in your function waveform before the measurement.
- OFF — The oscilloscope makes measurements without waiting for function averages to complete.

This command maps to the **Wait for function average to complete for measurements** check box in the front panel user interface's Measurement Setup dialog box, General tab.

Note that acquisition averaging can take place in hardware before function averaging.

Query `:ACQuire:COMPlete:STATe?`

The :ACQuire:COMPlete:STATe? query returns the :ACQuire:COMPlete:STATe setting.

History Legacy command (existed before version 3.10).

Version 10.00: Because acquisition averaging now takes place in hardware, this command changed from **Wait for acquisition average to complete for measurements** to **Wait for function average to complete for measurements**.

:ACQuire:DIFFerential:PARTner

Command `:ACQuire:DIFFerential:PARTner {EOTHer | ADJacent}`

The :ACQuire:DIFFerential:PARTner command specifies the pairs of channels that are used for differential signals:

- EOTHer — Every other channel are used for differential pairs.

  For example, when you use ":CHANnel1:DIFFerential ON" or ":CHANnel3:COMMonmode ON", the channel 1 and channel 3 inputs are used for the differential pair. The channel 1 waveform displays the differential waveform (channel 1 - channel 3) and the channel 3 waveform displays the common mode waveform (channel 1 + channel 3).

  Likewise, when you use ":CHANnel2:DIFFerential ON" or ":CHANnel4:COMMonmode ON", the channel 2 and channel 4 inputs are used for the differential pair. The channel 2 waveform displays the differential waveform (channel 2 - channel 4) and the channel 4 waveform displays the common mode waveform (channel 2 + channel 4).

- ADJacent — Adjacent channels are used for differential pairs.

  For example, when you use ":CHANnel1:DIFFerential ON" or ":CHANnel2:COMMonmode ON", the channel 1 and channel 2 inputs are used for the differential pair. The channel 1 waveform displays the differential waveform (channel 1 - channel 2) and the channel 2 waveform displays the common mode waveform (channel 1 + channel 2).

  Likewise, when you use ":CHANnel3:DIFFerential ON" or ":CHANnel4:COMMonmode ON", the channel 3 and channel 4 inputs are used for the differential pair. The channel 3 waveform displays the differential waveform (channel 3 - channel 4) and the channel 4 waveform displays the common mode waveform (channel 3 + channel 4).

Query `:ACQuire:DIFFerential:PARTner?`

The :ACQuire:DIFFerential:PARTner? query returns the differential pairing selection.

Returned Format `<pairing><NL>`

`<pairing> ::= {EOTH | ADJ}`

See Also

- ":CHANnel<N>:DIFFerential" on page 440
- ":CHANnel<N>:COMMonmode" on page 439

History New in version 10.00.

:ACQuire:FPLot

Command `:ACQuire:FPLot {{0 | OFF} | {1 | ON}}`

The :ACQuire:FPLot command enables or disables the acquisition system's fast plotting mode.

The Infiniium UXR-Series oscilloscopes have a "Fast Plot" acquisition and plotting system that is much faster than the time it takes to perform data analysis and computation. In the time it takes to perform one analysis cycle, there could be thousands (or a million) acquisitions and plots.

To slow acquisitions/plots to the rate of analysis and computation, you can turn Fast Plot off.

- OFF — When Fast Plot is off, analysis cycles are performed synchronously with acquisitions and plots. The same data that was just analyzed is plotted.

  Hardware-assisted edge finding, clock recovery, and filtering continue to work while Fast Plot mode is off.
- ON — When Fast Plot is on, analysis cycles are performed asynchronously with acquisitions and plots. Analysis results are likely based on a previous acquisitions.

The :ACQuire:FPLot command maps to the **Fast Plot** control in the Acquisition dialog box of the front panel graphical user interface.

Because you sometimes want analysis to match the data being displayed, several analysis and post-capture-processing features require synchronous analysis and automatically disable Fast Plot:

- Limit Test
- Protocol Decode
- Mask Test
- Jitter/Noise Analysis
- InfiniiScan (or other software trigger qualification)
- FIR filtering performed by software

For example, when Limit Test finds a failure and stops acquisitions, you want the data that caused the failure to be shown.

Query `:ACQuire:FPLot?`

The :ACQuire:FPLot? query returns whether the acquisition system's Fast Plot mode is enabled (ON or 1) or disabled (OFF or 0).

Returned Format `<setting><NL>`

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

History New in version 10.00.

:ACQuire:HISTory:COUNt

Query

```
:ACQuire:HISTory:COUNt?
```

The :ACQuire:HISTory:COUNt? query returns the number of acquisitions in the history (after running acquisitions are stopped).

The amount of acquisition history is based on amount of licensed memory and the acquisition record length.

Returned Format

```
<#acquisitions><NL>

<#acquisitions> ::= integer number of acquisitions in NR1 format
```

See Also

- ":ACQuire:HISTory:INDex" on page 305
- ":ACQuire:HISTory:PLAY" on page 306

History New in version 11.00.

:ACQuire:HISTory:INDex

Command

```
:ACQuire:HISTory:INDex <index#>
```

The :ACQuire:HISTory:INDex command navigates to the specified acquisition in the history.

When running acquisitions are stopped, the history index is placed at the last acquisition. This index number is the same as the number returned by the :ACQuire:HISTory:COUNt? query.

When viewing acquisition history, stepping one at a time will clear measurement statistics, Play (see :ACQuire:HISTory:PLAY) will accumulate measurement statistics.

<index#> Integer in NR1 format.

Query

```
:ACQuire:HISTory:INDex?
```

The :ACQuire:HISTory:INDex? query returns the currently displayed acquisition history.

Returned Format

```
<index#><NL>
```

See Also

- ":ACQuire:HISTory:COUNt" on page 304
- ":ACQuire:HISTory:PLAY" on page 306

History New in version 11.00.

:ACQuire:HISTory:PLAY

Command

```
:ACQuire:HISTory:PLAY {{0 | OFF} | {1 | ON}}
```

The :ACQuire:HISTory:PLAY command turns the acquisition history play setting on or off.

When viewing acquisition history, stepping one at a time (see :ACQuire:HISTory:INDex) will clear measurement statistics, Play will accumulate measurement statistics.

Query

```
:ACQuire:HISTory:PLAY?
```

The :ACQuire:HISTory:PLAY? query returns the acquisition history play status.

Returned Format

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

See Also

- ":ACQuire:HISTory:COUNt" on page 304
- ":ACQuire:HISTory:INDex" on page 305

History New in version 11.00.

:ACQuire:INTerpolate

Command

```
:ACQuire:INTerpolate {{ON | 1} | {OFF | 0} | INT1 | INT2 | INT4 | INT8
    | INT16 | INT32}
```

The :ACQuire:INTerpolate command turns the sin(x)/x interpolation filter on or off when the oscilloscope is in one of the real time sampling modes. You can also specify the 1, 2, 4, 8, 16, or 32 point Sin(x)/x interpolation ratios using INT1, INT2, INT4, INT8, INT16, or INT32. When ON, the number of interpolation points is automatically determined.

Query

```
:ACQuire:INTerpolate?
```

The :ACQuire:INTerpolate? query returns the current state of the sin(x)/x interpolation filter control.

Returned Format

```
[:ACQuire:INTerpolate] {1 | 0 | INT1 | INT2 | INT4 | INT8 | INT16
    | INT32}<NL>
```

History

Legacy command (existed before version 3.10).

Version 3.10: Added the INT1, INT2, INT4, INT8, INT16 options for specifying the 1, 2, 4, 8, or 16 point Sin(x)/x interpolation ratios.

Version 10.00: Added the INT32 option for specifying the 32 point Sin(x)/x interpolation ratio.

:ACQuire:MODE

Command

```
:ACQuire:MODE {RTIMe | PDETect | HRESolution | SEGMented
    | SEGPdetect | ROLL}
```

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

RTIMe In Real Time Normal mode, the complete data record is acquired on a single trigger event.

PDETect In Real Time Peak Detect mode, the oscilloscope acquires all of the waveform data points during one trigger event. The data is acquired at the fastest sample rate of the oscilloscope regardless of the horizontal scale setting. The sampling rate control then shows the storage rate into the channel memory rather than the sampling rate. The storage rate determines the number of data points per data region. From each data region, four sample points are chosen to be displayed for each time column. The four sample points chosen from each data region are:

- the minimum voltage value sample
- the maximum voltage value sample
- a randomly selected sample
- an equally spaced sample

The number of samples per data region is calculated using the equation:

$$\text{Number of Samples} = \frac{\text{Sampling Rate}}{\text{Storage Rate}}$$

The remainder of the samples are not used for display purposes.

HRESolution In Real Time High Resolution mode, the oscilloscope acquires all the waveform data points during one trigger event and averages them thus reducing noise and improving voltage resolution. The data is acquired at the fastest sample rate of the oscilloscope regardless of the horizontal scale setting. The sampling rate control then shows the storage rate into the channel memory rather than the sampling rate. The number of samples that are averaged together per data region is calculated using the equation

$$\text{Number of Samples} = \frac{\text{Sampling Rate}}{\text{Storage Rate}}$$

This number determines how many samples are averaged together to form the 16-bit samples that are stored into the channel memories.

To set the desired bits of vertical resolution, see ":ACQuire:HRESolution" on page 1835.

SEGMented In this sampling mode you can view waveform events that are separated by long periods of time without capturing waveform events that are not of interest to you.

SEGPdetect Enables Peak Detect Segmented mode.

ROLL Enables Roll mode. Roll Mode is a triggerless acquisition mode in which acquisition data is displayed in a rolling fashion starting at the right side of the display and continuing to the left (while the acquisition is running). Roll mode is useful when making manual adjustments to low-frequency waveforms, finding disturbances in low-frequency waveforms, or monitoring the power-up cycle of a power supply voltage. You can also enable or disable Roll Mode using the :TIMebase:ROLL:ENABle (see page 1548) command.

Example This example sets the acquisition mode to Real Time Normal.

```
myScope.WriteString ":ACQuire:MODE RTIMe"
```

Query

```
:ACQuire:MODE?
```

The :ACQuire:MODE? query returns the current acquisition sampling mode.

Returned Format

```
[:ACQuire:MODE] {RTIM | PDET | HRES | SEGM | SEGP | ROLL}<NL>
```

Example This example places the current acquisition mode in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":ACQuire:MODE?"
strMode = myScope.ReadString
Debug.Print strMode
```

History Legacy command (existed before version 3.10).

Version 10.00: The Equivalent Time (ETIMe) option is not available in the Keysight UXR Series oscilloscopes.

Version 11.00: In the MXR/EXR-Series oscilloscopes, ROLL mode is available and SEGHres is no longer available since acquisition modes and high-resolution oscilloscope modes are now separate.

:ACQuire:POINts[:ANALog] — Memory depth

Command

```
:ACQuire:POINts[:ANALog] {AUTO | <points_value>}
```

The :ACQuire:POINts[:ANALog] command sets the requested analog memory depth for an acquisition. Before you download data from the oscilloscope to your computer, always query the points value with the :WAVeform:POINts? query or :WAVeform:PREamble? query to determine the actual number of acquired points.

You can set the points value to AUTO, which allows the oscilloscope to select the optimum memory depth and display update rate.

<points_value>

An integer representing the memory depth.

The range of points available for a channel depends on the oscilloscope settings of sampling mode, sampling rate, and trigger sweep.

Interaction between :ACQuire:SRATe[:ANALog] and :ACQuire:POINts[:ANALog]

If you assign a sample rate value with :ACQuire:SRATe[:ANALog] or a points value using :ACQuire:POINts[:ANALog] the following interactions will occur. "Manual" means you are setting a non-AUTO value for SRATe or POINts.

SRATe	POINts	Result
AUTO	Manual	POINts value takes precedence (sample rate is limited)
Manual	AUTO	SRATe value takes precedence (memory depth is limited)
Manual	Manual	SRATe value takes precedence (memory depth is limited)

Example

This example sets the memory depth to 500 points.

```
myScope.WriteString ":ACQuire:POINts:ANALog 500"
```

Query

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

The :ACQuire:POINts[:ANALog]? query returns the value of the analog memory depth control.

Returned Format

```
[:ACQuire:POINts:ANALog] <points_value><NL>
```

Example

This example checks the current setting for memory depth and places the result in the variable, varLength. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:POINts:ANALog?"
varLength = myScope.ReadNumber
Debug.Print FormatNumber(varLength, 0)
```

See Also

- ":WAVeform:DATA" on page 1675

History Legacy command (existed before version 3.10).

:ACQuire:POINts:AUTO

Command

```
:ACQuire:POINts:AUTO {{ON | 1} |{OFF | 0}}
```

The :ACQuire:POINts:AUTO command enables (automatic) or disables (manual) the automatic memory depth selection control. When enabled, the oscilloscope chooses a memory depth that optimizes the amount of waveform data and the display update rate. When disabled, you can select the amount of memory using the :ACQuire:POINts command.

Example This example sets the automatic memory depth control to off.

```
myScope.WriteString ":ACQuire:POINts:AUTO OFF"
```

Query

```
:ACQuire:POINts:AUTO?
```

The :ACQuire:POINts:AUTO? query returns the automatic memory depth control state.

Returned Format

```
[:ACQuire:POINts:AUTO] {1 | 0}<NL>
```

Example This example checks the current setting for automatic memory depth control and places the result in the variable, varState. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:POINts:AUTO?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

See Also :WAVeform:DATA?

History Legacy command (existed before version 3.10).

:ACQuire:POINts:DIGital?

Query `:ACQuire:POINts:DIGital?`

The :ACQuire:POINts:DIGital query returns the current memory depth for the digital channels (MSO models only).

History Legacy command (existed before version 3.10).

:ACQuire:POINts:TESTLIMITS?

Query

```
:ACQuire:POINts:TESTLIMITS?
```

The :ACQuire:POINts:TESTLIMITS? query returns the oscilloscope's acquisition points maximum and minimum limits.

Returned Format

```
<num_parms>,<<type>><min>:<max><NL>
```

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<numeric>" for this query.

<min> Lower points limit value.

<max> Upper points limit value.

See Also

- ":ACQuire:POINts[:ANALog] — Memory depth" on page 310
- ":ACQuire:POINts:AUTO" on page 312
- ":ACQuire:BANDwidth:TESTLIMITS?" on page 298
- ":ACQuire:SRATe:TESTLIMITS?" on page 327

History New in version 5.60.

:ACQuire:REDGe

Command `:ACQuire:REDGe {OFF | 0}`

In the UXR Series oscilloscopes, the only valid setting is OFF.

Query `:ACQuire:REDGe?`

The :ACQuire:REDGe? query returns the current setting.

Returned Format `[:ACQuire:REDGe] 0<NL>`

History New in version 4.00.

:ACQuire:RESPonse

Command

```
:ACQuire:RESPonse {FLATmag | GAUSsianmag}
```

The Flat Magnitude filter is the default one and is the filter typically used on Infiniium oscilloscopes. The Gaussian Magnitude filter eliminates all ringing (preshoot or overshoot) caused by the oscilloscope's response. Therefore, any ringing you see in the displayed signal is actually in your signal and is not caused by the oscilloscope. The main drawback to using the Gaussian Magnitude Filter is the decrease in bandwidth. Please consult the Flat Magnitude / Magnitude Magnitude Filters topic in the help system for specific information regarding the decrease in bandwidth.

Example This example turns on the Gaussian Magnitude filter.

```
myScope.WriteString ":ACQuire:RESPonse GAUSsianmag"
```

Query

```
:ACQuire:RESPonse?
```

The :ACQuire:RESPonse? query returns the current filter being used.

Returned Format

```
[:ACQ:RESP] {FLAT | GAUS}<NL>
```

Example This example checks the current filter setting and places the result in the variable, state. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:RESPonse?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

History Legacy command (existed before version 3.10).

:ACQuire:SEGMented:AUToplay

Command

```
:ACQuire:SEGMented:AUToplay {{0 | OFF} | {1 | ON}}
```

The :ACQuire:SEGMented:AUToplay command specifies whether segments are automatically played after a segmented memory acquisition.

Query

```
:ACQuire:SEGMented:AUToplay?
```

The :ACQuire:SEGMented:AUToplay? query returns the segmented memory autoplay setting.

Returned Format

```
<setting><NL>

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

See Also

- ":ACQuire:MODE" on page 308
- ":ACQuire:SEGMented:PLAY" on page 320
- ":ACQuire:SEGMented:PRATe" on page 321
- ":ACQuire:SEGMented:COUNt" on page 318
- ":ACQuire:SEGMented:INDex" on page 319
- ":ACQuire:SEGMented:TTAGs" on page 322

History New in version 6.00.

:ACQuire:SEGMented:COUNt

Command

```
:ACQuire:SEGMented:COUNt <#segments>
```

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

<#segments> An integer representing the number of segments to acquire.

Example This example sets the segmented memory count control to 1000.

```
myScope.WriteString ":ACQuire:SEGMented:COUNt 1000"
```

Query

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

The :ACQuire:SEGMented:COUNT? query returns the number of segments control value.

Returned Format

```
[:ACQuire:SEGMented:COUNt] <#segments><NL>
```

Example This example checks the current setting for segmented memory count control and places the result in the variable, varSegments. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SEGMented:COUNt?"
varSegments = myScope.ReadNumber
Debug.Print FormatNumber(varSegments, 0)
```

See Also

- ":ACQuire:SEGMented:INDex" on page 319
- ":ACQuire:SEGMented:PLAY" on page 320
- ":ACQuire:SEGMented:TTAGs" on page 322

History Legacy command (existed before version 3.10).

:ACQuire:SEGMented:INDex

Command

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

The :ACQuire:SEGMented:INDex command sets the index number for the segment that you want to display on screen in the segmented memory mode. If an index value larger than the total number of acquired segments is sent, an error occurs indicating that the data is out of range and the segment index is set to the maximum segment number.

<index#> An integer representing the index number of the segment that you want to display.

Example This example sets the segmented memory index number control to 1000.

```
myScope.WriteString ":ACQuire:SEGMented:INDex 1000"
```

Query

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

The :ACQuire:SEGMented:INDex? query returns the segmented memory index number control value.

Returned Format

```
[:ACQuire:SEGMented:INDex] <index#><NL>
```

Example This example checks the current setting for segmented memory index number control and places the result in the variable, varIndex. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SEGMented:INDex?"
varIndex = myScope.ReadNumber
Debug.Print FormatNumber(varIndex, 0)
```

See Also

- ":ACQuire:SEGMented:COUNt" on page 318
- ":ACQuire:SEGMented:PLAY" on page 320
- ":ACQuire:SEGMented:TTAGs" on page 322

History Legacy command (existed before version 3.10).

:ACQuire:SEGMented:PLAY

Command

```
:ACQuire:SEGMented:PLAY {{0 | OFF} | {1 | ON}}
```

The :ACQuire:SEGMented:PLAY command plays (or stops) acquired segments.

- ON — is the same as clicking the play button in the graphical user interface.
- OFF — is the same as clicking the stop button in the graphical user interface.

Playing acquired segments can take a while depending on the analysis taking place. You can query to determine when playing is complete.

Query

```
:ACQuire:SEGMented:PLAY?
```

The :ACQuire:SEGMented:PLAY? query returns whether segments are currently being played (1) or are stopped (0).

Returned Format

```
[:ACQuire:SEGMented:PLAY] <setting><NL>
```

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

See Also

- ":ACQuire:SEGMented:PRATe" on page 321
- ":ACQuire:SEGMented:COUNt" on page 318
- ":ACQuire:SEGMented:INDex" on page 319
- ":ACQuire:SEGMented:TTAGs" on page 322

History New in version 5.60.

:ACQuire:SEGMented:PRATe

Command

```
:ACQuire:SEGMented:PRATe <time_per_seg>
```

```
<time_per_seg> ::= seconds from 0.001 to 1 in NR3 format.
```

The :ACQuire:SEGMented:PRATe command specifies the segmented memory navigation play rate.

When playing segments, the current segment through the last segment are displayed at the specified rate. Playing segments lets you collect measurement statistics across all the played-back segments.

Query

```
:ACQuire:SEGMented:PRATe?
```

The :ACQuire:SEGMented:PRATe? query returns segmented memory navigation play rate.

Returned Format

```
<time_per_seg><NL>
```

See Also

- ":ACQuire:SEGMented:PLAY" on page 320

History New in version 5.70.

:ACQuire:SEGMented:TTAGs

Command

```
:ACQuire:SEGMented:TTAGs {{ON | 1} | {OFF | 0}}
```

The :ACQuire:SEGMented:TTAGs command turns the time tags feature on or off for the segmented memory sampling mode.

Example This example turns the time tags on for segmented memory.

```
myScope.WriteString ":ACQuire:SEGMented:TTAGs ON"
```

Query

```
:ACQuire:SEGMented:TTAGs?
```

The :ACQuire:SEGMented:TTAGs? query returns the segmented memory time tags control value.

Returned Format

```
[:ACQuire:SEGMented:TTAGs] {1 | 0}<NL>
```

Example This example checks the current setting for segmented memory time tags control and places the result in the variable, varTimeTags. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SEGMented:TTAGs?"
varTimeTags = myScope.ReadNumber
Debug.Print FormatNumber(varTimeTags, 0)
```

See Also

- ":ACQuire:SEGMented:COUNt" on page 318
- ":ACQuire:SEGMented:INDex" on page 319
- ":ACQuire:SEGMented:PLAY" on page 320

History Legacy command (existed before version 3.10).

:ACQuire:SRATe[:ANALog] — Analog Sample Rate

Command

```
:ACQuire:SRATe[:ANALog] {AUTO | MAX | <rate>}
```

The :ACQuire:SRATe[:ANALog] command sets the analog acquisition sampling rate.

AUTO The AUTO rate allows the oscilloscope to select a sample rate that best accommodates the selected memory depth and horizontal scale.

MAX The MAX rate enables the oscilloscope to select maximum available sample rate.

<rate> A real number representing the sample rate. You can send any value, but the value is rounded to the next fastest sample rate.

Interaction between :ACQuire:SRATe[:ANALog] and :ACQuire:POINts[:ANALog]

If you assign a sample rate value with :ACQuire:SRATe[:ANALog] or a points value using :ACQuire:POINts[:ANALog] the following interactions will occur. "Manual" means you are setting a non-AUTO value for SRATe or POINts.

SRATe	POINts	Result
AUTO	Manual	POINts value takes precedence (sample rate is limited)
Manual	AUTO	SRATe value takes precedence (memory depth is limited)
Manual	Manual	SRATe value takes precedence (memory depth is limited)

Example This example sets the sample rate to 250 MSa/s.

```
myScope.WriteString ":ACQuire:SRATe:ANALog 250E+6"
```

Query

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

The :ACQuire:SRATe[:ANALog]? query returns the current analog acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe:ANALog] {<rate>}<NL>
```

Example This example places the current sample rate in the string variable, strSample, then prints the contents of the variable to the computer's screen.

```
Dim strSample As String   ' Dimension variable.
myScope.WriteString ":ACQuire:SRATe:ANALog?"
strSample = myScope.ReadString
Debug.Print strSample
```

History Legacy command (existed before version 3.10).

:ACQuire:SRATe[:ANALog]:AUTO

Command

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

The :ACQuire:SRATe[:ANALog]:AUTO command enables (ON) or disables (OFF) the automatic analog sampling rate selection control. On the oscilloscope front-panel interface, ON is equivalent to Automatic and OFF is equivalent to Manual.

Example

This example changes the sampling rate to manual.

```
myScope.WriteString ":ACQuire:SRATe:ANALog:AUTO OFF"
```

Query

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

The :ACQuire:SRATe[:ANALog]:AUTO? query returns the current acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe:ANALog:AUTO] {1 | 0}<NL>
```

Example

This example places the current analog sample rate in the variable, varSample, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SRATe:ANALog:AUTO?"
varSample = myScope.ReadNumber
Debug.Print FormatNumber(varSample, 0)
```

History

Legacy command (existed before version 3.10).

:ACQuire:SRATe:DIGital — Digital Channels Sample Rate

Command

```
:ACQuire:SRATe:DIGital {AUTO | MAX | <rate>}
```

The :ACQuire:SRATe:DIGital command sets the digital acquisition sampling rate.

AUTO The AUTO rate allows the oscilloscope to select a sample rate that best accommodates the selected memory depth and horizontal scale.

MAX The MAX rate enables the oscilloscope to select maximum available sample rate.

<rate> A real number representing the digital sample rate. You can send any value, but the value is rounded to the next fastest sample rate.

Interaction between :ACQuire:SRATe:DIGital and :ACQuire:POINts:DIGital? If you assign a sample rate value with :ACQuire:SRATe:DIGital, the digital memory depth is automatically adjusted and can be seen by using the :ACQuire:POINts:DIGital? query.

Query

```
:ACQuire:SRATe:DIGital?
```

The :ACQuire:SRATe:DIGital? query returns the current digital acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe:DIGital] {<rate>}<NL>
```

Example This example places the current digital channel sample rate in the string variable, strSample, then prints the contents of the variable to the computer's screen.

```
Dim strSample As String   ' Dimension variable.
myScope.WriteString ":ACQuire:SRATe:DIGital?"
strSample = myScope.ReadString
Debug.Print strSample
```

History Legacy command (existed before version 3.10).

:ACQuire:SRATe:DIGital:AUTO

Command

```
:ACQuire:SRATe:DIGital:AUTO {{ON | 1} | {OFF | 0}}
```

The :ACQuire:SRATe:DIGital:AUTO command enables (ON) or disables (OFF) the automatic digital channel sampling rate selection control.

Example This example changes the digital channel sampling rate to manual.

```
myScope.WriteString ":ACQuire:SRATe:DIGital:AUTO OFF"
```

Query

```
:ACQuire:SRATe:DIGital:AUTO?
```

The :ACQuire:SRATe:DIGital:AUTO? query returns the current digital channel acquisition sample rate.

Returned Format

```
[:ACQuire:SRATe:DIGital:AUTO] {1 | 0}<NL>
```

Example This example places the current digital channel sample rate in the variable, varSample, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SRATe:DIGital:AUTO?"
varSample = myScope.ReadNumber
Debug.Print FormatNumber(varSample, 0)
```

History Legacy command (existed before version 3.10).

:ACQuire:SRATe:TESTLIMITS?

Query

```
:ACQuire:SRATe:TESTLIMITS?
```

The :ACQuire:SRATe:TESTLIMITS? query returns the oscilloscope's acquisition sampling rate maximum and minimum limits.

Returned Format

```
<num_parms>,<<type>><min>:<max><NL>
```

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<numeric>" for this query.

<min> Lower sample rate limit value.

<max> Upper sample rate limit value.

See Also

- ":ACQuire:SRATe[:ANALog] — Analog Sample Rate" on page 323
- ":ACQuire:SRATe[:ANALog]:AUTO" on page 324
- ":ACQuire:BANDwidth:TESTLIMITS?" on page 298
- ":ACQuire:POINts:TESTLIMITS?" on page 314

History New in version 5.60.

15 :ANALyze Commands

The commands in the ANALyze subsystem are used to:

- Specify whether to use a single edge or all edges in the acquisition for horizontal measurements (:ANALyze:AEDGes command).

- Set up clock recovery (:ANALyze:CLOCk commands).
- Specify a waveform source's signal type:
  - The PAM4 signal type sets up a new paradigm for serial data signal analysis with multiple data levels and edges for clock recovery (as compared to high and low level NRZ signals).
  - The MMWave signal type supports analysis of millimeter-wave signals that have been down-converted to IF band signals by an external smart mixer and an LO signal from a signal generator.
- Support MMWave millimeter-wave signal analysis.
- Specify whether to use the data on screen or the entire acquisition for measurements, functions, and analysis (:ANALyze:VIEW command).

Sources for Analyze Commands

Some :ANALyze commands let you specify the waveform source(s) using a <source> parameter:

```
<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R>
   | FUNCtion<F> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor
   | LFPR}
```

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

:ANALyze:AEDGes

Command

```
:ANALyze:AEDGes {{0 | OFF} | {1 | ON}}
```

The :ANALyze:AEDGes command specifies whether a single edge or all edges in the acquisition are used for horizontal measurements.

The :ANALyze:AEDGes command maps to the **Measure All Edges** control in the user interface's Measurement Setup dialog box.

When all edges in an acquisition are used for horizontal measurements, the entire acquisition is also used for measurements, functions, and analysis (see :ANALyze:VIEW ALL).

Some measurements require all edges: ":MEASure:NPULses" on page 1003, ":MEASure:PPULses" on page 1073, ":MEASure:ETOedge" on page 930, ":MEASure:HOLDtime" on page 965, ":MEASure:NPERiod" on page 1002, ":MEASure:PHASe" on page 1053, ":MEASure:SETuptime" on page 1117, and EZJIT clock and data measurements. When you add one of these measurements, the :ANALyze:AEDGes option is automatically turned ON.

Also, turning on a real-time eye (:MTESt:FOLDing ON) sets :ANALyze:AEDGes to ON, and it cannot be disabled.

Query

```
:ANALyze:AEDGes?
```

The :ANALyze:AEDGes? query returns the value that is currently set.

Returned Format

```
[:ANALyze:AEDGes] {0 | 1}<NL>
```

See Also

- ":ANALyze:VIEW" on page 407

History

New in version 5.30. This commands replaces the now deprecated command ":MEASure:JITTer:STATistics" on page 1885.

:ANALyze:CLOCk

Command

```
:ANALyze:CLOCk {{{ON|1},<source>} | {OFF|0}}
```

The :ANALyze:CLOCk command turns the recovered clock display on or off and sets the clock recovery channel source.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R>}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

Example This example turns the recovered clock display on for channel 1.

```
myScope.WriteString ":ANALyze:CLOCk ON,CHANnel1"
```

Query

```
:ANALyze:CLOCk?
```

The :ANALyze :CLOCk? query returns the state of the recovered clock display.

Returned Format

```
[:ANALyze:CLOCk] {1 | 0}<NL>
```

Example This example places the current setting of the recovered clock display in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk" on page 1862.

:ANALyze:CLOCk:METHod

Command

```
:ANALyze:CLOCk:METHod
    {FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}
  | {TOPLL[,<data_rate>[,<natural_frequency>[,<pole_frequency>[,
      <damping_factor>[,<PLL_settling_time>]]]]]}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH}[,
      <multiplier>[,<clock_freq>]]}
  | {EXPTOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<natural_frequency>,
      <pole_frequency>,<damping_factor>}
  | {EQTOPLL[,<data_rate>[,<natural_frequency>[,<pole_frequency>[,
      <damping_factor>[,<PLL_settling_time>]]]]]}
  | {FC,{FC1063 | FC2125 | FC425}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
  | {PWM}
  | {CPHY[,<symbol_rate>[,<setup_UI>]]}
  | {BMC}
  | {LFPS}
  | {PCIE5,{PCIE8 | PCIE16 | PCIE32}
  | {PCIE6,{PCIE8 | PCIE16 | PCIE32}}
```

The :ANALyze:CLOCk:METHod command sets the clock recovery method to:

- FIXed (Constant Frequency)
- TOPLL (Third Order PLL)
- EXPlicit (Explicit Clock)
- EXPTOPLL (Explicit Third Order PLL)
- EQTOPLL (Equalized Third Order PLL)
- FC (Fibre Channel)
- FLEXR (FlexRay Receiver)
- FLEXT (FlexRay Transmitter)
- PWM (MIPI M-PHY PWM)
- CPHY (MIPI C-PHY)
- BMC (USB PD bi-phase mark coding)
- LFPS (USB 3 low frequency periodic signaling)
- PCIE5 (PCIe 5 CXL Behavioral SRIS CC) and 8 Gb/s, 16 Gb/s, or 32 Gb/s data rates
- PCIE6 (PCIe 6 CXL Behavioral SRIS CC) and 8 Gb/s, 16 Gb/s, or 32 Gb/s data rates

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

For setting first order and second order phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":ANALyze:CLOCk:METHod:OJTF" on page 345.

For setting first order and second order phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":ANALyze:CLOCk:METHod:JTF" on page 343.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<data_rate> A real number for the base data rate in Hertz.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

<natural_frequency> A real number for the natural frequency of the PLL.

<pole_frequency> A real number for the pole frequency of the PLL.

<damping_ factor> A real number for the damping factor of the PLL.

<PLL_settling_time> A real number for the PLL settling time.

<multiplier> An integer used as the multiplication factor.

<clock_freq> To perform hardware-accelerated clock recovery with the EXPlicit clock recovery method (when hardware acceleration is licensed), you must specify the expected clock frequency of the explicit clock source using the optional <clock_freq> parameter. When the <clock_freq> parameter is not included, clock recovery is performed in software instead of being hardware-accelerated.

Hardware-accelerated EXPlicit clock recovery can also be enabled by setting the **Clock Frequency** in the graphical user interface (GUI) or by loading a newer setup file that includes the setting. Loading an older setup file that does not have the setting will cause clock recovery to be performed in software.

<baud_rate> A real number used for the baud rate.

<symbol_rate> When the signal type is CPHY (see :ANALyze:SIGNal:TYPE), CPHY is automatically selected as the clock recovery method, but you can use the :ANALyze:CLOCk:METHod CPHY command to specify the nominal a symbol rate in baud.

<setup_UI> When the signal type is CPHY (see :ANALyze:SIGNal:TYPE), CPHY is automatically selected as the clock recovery method, but you can use the :ANALyze:CLOCk:METHod CPHY command to specify the time (in Unit Intervals) used to cluster edges for clock recovery.

Example This example sets the explicit clock recovery method on channel 1, rising edge, with a multiplier of 2.

```
myScope.WriteString ":ANALyze:CLOCk:METHod EXPlicit,CHANnel1,RISing,2"
```

Query

```
:ANALyze:CLOCk:METHod?
```

The :ANALyze:CLOCk:METHod? query returns the state of the clock recovery method.

NOTE

You can use the :ANALyze:CLOCk:METHod? query when phase-locked loop (PLL) clock recovery methods are set up. The format returned will be that of the :ANALyze:CLOCk:METHod:OJTF? query. See ":ANALyze:CLOCk:METHod:OJTF" on page 345.

Returned Format

```
[:ANALyze:CLOCk:METHod]
    {FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
  | {TOPLL,<data_rate>,<natural_frequency>,<pole_frequency>,
      <damping_factor>}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>}
  | {EXPTOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<natural_frequency>,<pole_frequency>,
      <damping_factor>}
  | {EQTOPLL,<data_rate>,<natural_frequency>,<pole_frequency>,
      <damping_factor>}
  | {FC,{FC1063 | FC2125 | FC425}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
  | {PWM}
  | {CPHY,<symbol_rate>,<setup_UI>}
  | {BMC}
  | {LFPS}
  | {PCIE5,{PCIE8 | PCIE16 | PCIE32}
  | {PCIE6,{PCIE8 | PCIE16 | PCIE32}}
```

Example

This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":ANALyze:SIGNal:TYPE" on page 404

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod" on page 1863.

Version 5.50: When the signal type is PAM-4, a symbol rate (baud) is specified instead of a data rate (b/s).

Version 5.60: Added the PWM and CPHY methods.

Version 10.10: Added the BMC (USB PD bi-phase mark coding) and LFPS (USB 3 low frequency periodic signaling) methods. Added the optional <clock_freq> parameter for EXPlicit clock recovery.

Version 10.20: Added the TOPLL (Third Order PLL), EXPTOPLL (Explicit Third Order PLL), and EQTOPLL (Equalized Third Order PLL) clock recovery methods.

Version 11.10: Added the "PCIE5,PCIE8", "PCIE5,PCIE16", and "PCIE5,PCIE32" PCIe 5 CXL Behavioral SRIS CC clock recovery methods for 8 Gb/s, 16 Gb/s, and 32 Gb/s data rates, respectively.

Version 11.15: Added the "PCIE6,PCIE8", "PCIE6,PCIE16", and "PCIE6,PCIE32" PCIe 6 CXL Behavioral SRIS CC clock recovery methods for 8 Gb/s, 16 Gb/s, and 32 Gb/s data rates, respectively.

:ANALyze:CLOCk:METHod:ALIGn

Command

```
:ANALyze:CLOCk:METHod:ALIGn {CENTer | EDGE}
```

When using an explicit method of clock recovery, the :ANALyze:CLOCk:METHod:ALIGn command specifies how the clock is aligned with data:

- CENTer — Clock edges are aligned with the center of data.
- EDGE — Clock edges are aligned with data edges. In this case, Time Interval Error (TIE) is measured directly from the data edge to the clock edge.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

Example

When using an explicit method of clock recovery, this example specifies that clock edges are aligned with the center of data.

```
myScope.WriteString ":ANALyze:CLOCk:METHod:ALIGn CENTer"
```

Query

```
:ANALyze:CLOCK:METHod:ALIGn?
```

The :ANALyze:CLOCk:METHod:ALIGn? query returns the clock recovery method's edge alignment setting.

Returned Format

```
[:ANALyze:CLOCk:METHod:ALIGn] {CENT | EDGE}
```

Example

This example places the current edge alignment setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:ALIGn?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340

History

New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:ALIGn" on page 1867.

:ANALyze:CLOCk:METHod:DEEMphasis

Command

```
:ANALyze:CLOCk:METHod:DEEMphasis {OFF | ON}
```

The :ANALyze:CLOCk:METHod:DEEMphasis command turns de-emphasis on or off.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

See the help system for more information on de-emphasis.

Example

This example enables de-emphasis.

```
myScope.WriteString ":ANALyze:CLOCk:METHod:DEEMphasis ON"
```

Query

```
:ANALyze:CLOCk:METHod:DEEMphasis?
```

The :ANALyze:CLOCk:METHod:DEEMphasis? query returns whether or not de-emphasis is turned on.

Returned Format

```
[:ANALyze:CLOCk:METHod:DEEMphasis] {OFF | ON}
```

Example

This example places the current setting of the de-emphasis mode in the string variable strDeemph, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:DEEMphasis?"
strDeemph = myScope.ReadString
Debug.Print strDeemph
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340

History

New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868.

:ANALyze:CLOCk:METHod:EDGE

Command

```
:ANALyze:CLOCk:METHod:EDGE {RISing | FALLing | BOTH}
```

The :ANALyze:CLOCk:METHod:EDGE command specifies which edge(s) of the data are used to recover a clock. (In the front panel GUI, this control appears in the Advanced Clock Recovery dialog box.) Normally, both edges are used. However, if you are performing clock recovery on a low duty cycle clock signal, for example, you may want to use just the rising or falling edge.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

This command applies to the following clock recovery methods:

- FIXed (Constant Frequency).
- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EXPlicit (Explicit Clock).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).

To measure jitter on only rising (or falling) edges of a clock, you must also set :ANALyze:RJDJ:EDGE to the same RISing or FALLing option, and you must set :ANALyze:RJDJ:CLOCk ON to force the pattern to be a clock and set the jitter for edges not examined to zero (0).

Example

This example specifies that both rising and falling edges of the data are used to recover a clock.

```
myScope.WriteString ":ANALyze:CLOCk:METHod:EDGE BOTH"
```

Query

```
:ANALyze:CLOCK:METHod:EDGE?
```

The :ANALyze:CLOCk:METHod:EDGE? query returns the clock recovery method's edge setting.

Returned Format

```
[:ANALyze:CLOCk:METHod:EDGE] {RIS | FALL | BOTH}
```

Example

This example places the current edge setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:EDGE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351

- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":MEASure:RJDJ:EDGE" on page 1097
- ":MEASure:RJDJ:CLOCk" on page 1095

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:EDGE" on page 1869.

:ANALyze:CLOCk:METHod:IDLe

Command

```
:ANALyze:CLOCk:METHod:IDLe <#_idle_clocks>
```

The :ANALyze:CLOCk:METHod:IDLe command lets you specify the number of additional clocks output by the clock recovery PLL for situations where valid data can occur during electrical idles.

<#_idle_clocks> Number of PLL idle clocks from 6 to 1000 in NR1 format.

Query

```
:ANALyze:CLOCk:METHod:IDLe?
```

The :ANALyze:CLOCk:METHod:IDLe? query returns the number of PLL idle clocks setting.

Returned Format

```
<#_idle_clocks><NL>
```

See Also

- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:CLOCk:METHod:SOURce" on page 351

History New in version 6.20.

:ANALyze:CLOCk:METHod:JTF

Command

```
:ANALyze:CLOCk:METHod:JTF
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
```

The :ANALyze:CLOCk:METHod:JTF command specifies the clock recovery PLL's response in terms of the Jitter Transfer Function's (JTF) 3 dB bandwidth.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL)
- SOPLL (Second Order PLL)
- EQFOPLL (Equalized First Order PLL)
- EQSOPLL (Equalized Second Order PLL)
- EXPFOPLL (Explicit First Order PLL)
- EXPSOPLL (Explicit Second Order PLL)

The equalized clock recovery methods are available when the Advanced Signal Integrity Software license is installed.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":ANALyze:CLOCk:METHod:OJTF" on page 345.

For setting other clock recovery methods, see ":ANALyze:CLOCk:METHod" on page 334.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<data_rate> A real number for the base data rate in bits per second.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

<jtf_loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<peaking> The peaking value in dB.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a peaking value of 1.25 dB.

```
myScope.WriteString ":ANALyze:CLOCk:METHod:JTF SOPLL,4E9,3.822E6,1.25"
```

Query

```
:ANALyze:CLOCk:METHod:JTF?
```

The :ANALyze:CLOCk:METHod:JTF? query returns the state of the clock recovery method.

Returned Format

```
[:ANALyze:CLOCk:METHod:JTF]
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:JTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":ANALyze:SIGNal:TYPE" on page 404

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:JTF" on page 1871.

Version 5.50: When the signal type is PAM-4, a symbol rate (baud) is specified instead of a data rate (b/s).

:ANALyze:CLOCk:METHod:OJTF

Command

```
:ANALyze:CLOCk:METHod:OJTF
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
```

The :ANALyze:CLOCk:METHod:OJTF command specifies the clock recovery PLL's response in terms of the Observed Jitter Transfer Function's (OJTF) 3 dB bandwidth.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL)
- SOPLL (Second Order PLL)
- EQFOPLL (Equalized First Order PLL)
- EQSOPLL (Equalized Second Order PLL)
- EXPFOPLL (Explicit First Order PLL)
- EXPSOPLL (Explicit Second Order PLL)

The equalized clock recovery methods are available when the Advanced Signal Integrity Software license is installed.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":ANALyze:CLOCk:METHod:JTF" on page 343.

For setting other clock recovery methods, see ":ANALyze:CLOCk:METHod" on page 334.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<data_rate> A real number for the base data rate in bits per second.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

<ojtf_loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<damping_factor> A real number for the damping factor of the PLL.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a damping factor of 1.0.

```
myScope.WriteString ":ANALyze:CLOCk:METHod:OJTF SOPLL,4E9,2.4E6,1.0"
```

Query

```
:ANALyze:CLOCk:METHod:OJTF?
```

The :ANALyze:CLOCk:METHod:OJTF? query returns the state of the clock recovery method.

Returned Format

```
[:ANALyze:CLOCk:METHod:OJTF]
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_fact>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:OJTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":ANALyze:SIGNal:TYPE" on page 404

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:OJTF" on page 1873.

Version 5.50: When the signal type is PAM-4, a symbol rate (baud) is specified instead of a data rate (b/s).

:ANALyze:CLOCk:METHod:PLLadvanced

Command `:ANALyze:CLOCk:METHod:PLLadvanced {{0 | OFF} | {1 | ON}}`

The :ANALyze:CLOCk:METHod:PLLadvanced command enables or disables the "Advanced PLL for closed eyes" option.

If you are trying to recover a clock on closed eyes due to ISI or other jitter and noise sources, you can enable **Advanced PLL for closed eyes**. Our normal PLL clock recovery algorithms use edge timing of the data waveform to establish the clock edges. If the eye is closed, there may be so much jitter that this approach fails to yield accurate clocks. The Advanced PLL does not use edge information and can recover a clock even for closed eyes. However, if the eye is not closed, we recommend you use the traditional "Golden PLL" approach based upon edge timing.

Query `:ANALyze:CLOCk:METHod:PLLadvanced?`

The :ANALyze:CLOCk:METHod:PLLadvanced? query returns whether the setting is enabled or disabled.

Returned Format `<setting><NL>`

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

History New in version 10.25.

:ANALyze:CLOCk:METHod:PLLTrack

Command

```
:ANALyze:CLOCk:METHod:PLLTrack {OFF | ON}
```

The :ANALyze:CLOCk:METHod:PLLTrack command turns transition density dependence on or off. See the help system for more information on the Transition Density Dependent setting.

This command applies to the clock recovery method being set up for the waveform source selected by the :ANALyze:CLOCk:METHod:SOURce command.

Example

This example enables the Transition Density Dependent setting.

```
myScope.WriteString ":MEASure:CLOCk:METHod:PLLTrack ON"
```

Query

```
:ANALyze:CLOCk:METHod:PLLTrack?
```

The :ANALyze:CLOCk:METHod:PLLTrack? query returns whether or not the Transition Density Dependent setting is turned on.

Returned Format

```
[:ANALyze:CLOCk:METHod:PLLTrack] {OFF | ON}
```

Example

This example places the current setting of the Transition Density Dependent setting in the string variable strTDD, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:METHod:PLLTrack?"
strTDD = myScope.ReadString
Debug.Print strTDD
```

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:EDGE" on page 340

History

New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:PLLTrack" on page 1875.

:ANALyze:CLOCk:METHod:SKEW

Command `:ANALyze:CLOCk:METHod:SKEW <time>`

When clock recovery is being performed on a PAM-4 signal type (see :ANALyze:SIGNal:TYPE), the :ANALyze:CLOCk:METHod:SKEW command can be used to center the eye opening at the clock locations by shifting the clocks relative to the data.

<time> Seconds in NR3 format.

Query `:ANALyze:CLOCk:METHod:SKEW?`

The :ANALyze:CLOCk:METHod:SKEW? query returns the skew setting.

Returned Format `<time><NL>`

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:CLOCk:METHod:SKEW:AUTomatic" on page 350
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

History New in version 6.10.

:ANALyze:CLOCk:METHod:SKEW:AUTomatic

Command `:ANALyze:CLOCk:METHod:SKEW:AUTomatic`

When clock recovery is being performed on a PAM-4 signal type (see :ANALyze:SIGNal:TYPE), the :ANALyze:CLOCk:METHod:SKEW:AUTomatic command automatically shifts clocks relative to the data to center the eye opening at the clock locations. The current real-time eye data is used to determine the eye center locations.

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349

History New in version 10.10.

:ANALyze:CLOCk:METHod:SOURce

Command

```
:ANALyze:CLOCk:METHod:SOURce {ALL | <source>}
```

The :ANALyze:CLOCk:METHod:SOURce command selects the waveform source (or ALL sources) to which other clock recovery method setup commands apply.

Clock recovery methods can be set up for each waveform source (or for all waveform sources).

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

Query

```
:ANALyze:CLOCk:METHod:SOURce?
```

The :ANALyze:CLOCk:METHod:SOURce? query returns the waveform source to which other clock recovery method commands currently apply.

Returned Format

```
[:ANALyze:CLOCk:METHod:SOURce] <source><NL>
```

```
<source> ::= {ALL | CHAN<N> | FUNC<F> | WMEM<N> | EQU<L> | XT<X>
    | INPut | CORRected | ERRor | LFPR}
```

See Also

- ":ANALyze:CLOCk:METHod" on page 334
- ":ANALyze:CLOCk:METHod:OJTF" on page 345
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:PLLTrack" on page 348
- ":ANALyze:CLOCk:METHod:EDGE" on page 340

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:METHod:SOURce" on page 1876.

:ANALyze:CLOCk:VERTical

Command
```
:ANALyze:CLOCk:VERTical {AUTO | MANual}
```

The :ANALyze:CLOCk:VERTical command sets the recovered clock vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the recovered clock vertical scale mode to automatic.

```
myScope.WriteString ":ANALyze:CLOCk:VERTical AUTO"
```

Query
```
:ANALyze:CLOCk:VERTical?
```

The :ANALyze:CLOCk:VERTical? query returns the current recovered clock vertical scale mode setting.

Returned Format
```
[:ANALyze:CLOCk:VERTical] {AUTO | MANual}
```

Example This example places the current setting of the recovered clock vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ANALyze:CLOCk:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:VERTical" on page 1877.

:ANALyze:CLOCk:VERTical:OFFSet

Command

```
:ANALyze:CLOCk:VERTical:OFFSet <offset>
```

The :ANALyze:CLOCk:VERTical:OFFSet command sets the recovered clock vertical offset.

<offset> A real number for the recovered clock vertical offset.

Example This example sets the clock recovery vertical offset to 1 volt.

```
myScope.WriteString ":ANALyze:CLOCk:VERTical:OFFSet 1"
```

Query

```
:ANALyze:CLOCk:VERTical:OFFSet?
```

The :ANALyze:CLOCk:VERTical:OFFSet? query returns the clock recovery vertical offset setting.

Returned Format

```
[:ANALyze:CLOCk:VERTical:OFFSet] <value><NL>
```

<value> The clock recovery vertical offset setting.

Example This example places the current value of recovered clock vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ANALyze:CLOCk:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:VERTical:OFFSet" on page 1878.

:ANALyze:CLOCk:VERTical:RANGe

Command

```
:ANALyze:CLOCk:VERTical:RANGe <range>
```

The :ANALyze:CLOCk:VERTical:RANGe command sets the recovered clock vertical range.

<range> A real number for the full-scale recovered clock vertical range.

Example This example sets the recovered clock vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":ANALyze:CLOCk:VERTical:RANGe 16"
```

Query

```
:ANALyze:CLOCk:VERTical:RANGe?
```

The :ANALyze:CLOCk:VERTical:RANGe? query returns the recovered clock vertical range setting.

Returned Format

```
[:ANALyze:CLOCk:VERTical:RANGe] <value><NL>
```

<value> The recovered clock vertical range setting.

Example This example places the current value of recovered clock vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ANALyze:CLOCk:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History New in version 5.30. This command replaces the now deprecated command ":MEASure:CLOCk:VERTical:RANGe" on page 1879.

:ANALyze:HCRecovery

Query

```
:ANALyze:HCRecovery? <source>
```

The :ANALyze:HCRecovery? query returns a 1 if hardware-assisted clock recovery is being used on the specified source or a 0 if not.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C>}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

Returned Format

```
<setting><NL>

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

See Also

- ":ANALyze:HEQualizer" on page 356

History New in version 10.10.

:ANALyze:HEQualizer

Query

```
:ANALyze:HEQualizer?
```

The :ANALyze:HEQualizer? query returns a 1 if hardware-assisted equalization is being used or a 0 if not.

Returned Format

```
<setting><NL>

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

See Also

- ":ANALyze:HCRecovery" on page 355

History New in version 10.10.

:ANALyze:SNDR:CORRected

Command

```
:ANALyze:SNDR:CORRected {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SNDR:CORRected command lets you enable or disable the display of the Pulse Corrected waveform.

Query

```
:ANALyze:SNDR:CORRected?
```

The :ANALyze:SNDR:CORRected? query returns whether the Pulse Corrected waveform display is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected:HORizontal" on page 358
- ":ANALyze:SNDR:CORRected:HORizontal:POSition" on page 359
- ":ANALyze:SNDR:CORRected:HORizontal:SCALe" on page 360
- ":ANALyze:SNDR:CORRected:VERTical" on page 361
- ":ANALyze:SNDR:CORRected:VERTical:OFFSet" on page 362
- ":ANALyze:SNDR:CORRected:VERTical:SCALe" on page 363

History New in version 11.30.

:ANALyze:SNDR:CORRected:HORizontal

Command `:ANALyze:SNDR:CORRected:HORizontal {AUTO | MANual}`

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected), the :ANALyze:SNDR:CORRected:HORizontal command selects either automatic or manual horizontal scaling.

Query `:ANALyze:SNDR:CORRected:HORizontal?`

The :ANALyze:SNDR:CORRected:HORizontal? query returns the horizontal scaling selection.

Returned Format `<setting><NL>`

`<setting> ::= {AUTO | MAN}`

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:HORizontal:POSition" on page 359
- ":ANALyze:SNDR:CORRected:HORizontal:SCALe" on page 360

History New in version 11.30.

:ANALyze:SNDR:CORRected:HORizontal:POSition

Command

```
:ANALyze:SNDR:CORRected:HORizontal:POSition <position_value>
```

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected) and manual horizontal scaling is selected (see :ANALyze:SNDR:CORRected:HORizontal), the :ANALyze:SNDR:CORRected:HORizontal:POSition command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.

<position_value> A real number for the horizontal timebase reference's time value relative to the Pulse Corrected waveform's t=0 location.

Query

```
:ANALyze:SNDR:CORRected:HORizontal:POSition?
```

The :ANALyze:SNDR:CORRected:HORizontal:POSition? query returns the horizontal timebase reference's time value relative to the Pulse Corrected waveform's t=0 location.

Returned Format

```
<position_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:HORizontal" on page 358
- ":ANALyze:SNDR:CORRected:HORizontal:SCALe" on page 360

History New in version 11.30.

:ANALyze:SNDR:CORRected:HORizontal:SCALe

Command

```
:ANALyze:SNDR:CORRected:HORizontal:SCALe <scale_value>
```

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected) and manual horizontal scaling is selected (see :ANALyze:SNDR:CORRected:HORizontal), the :ANALyze:SNDR:CORRected:HORizontal:SCALe command specifies the waveform's horizontal time per division.

<scale_value> A real number for the Pulse Corrected waveform's horizontal scale in seconds per division.

Query

```
:ANALyze:SNDR:CORRected:HORizontal:SCALe?
```

The :ANALyze:SNDR:CORRected:HORizontal:SCALe? query returns the Pulse Corrected waveform's horizontal scale in seconds per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:HORizontal" on page 358
- ":ANALyze:SNDR:CORRected:HORizontal:POSition" on page 359

History New in version 11.30.

:ANALyze:SNDR:CORRected:VERTical

Command `:ANALyze:SNDR:CORRected:VERTical {AUTO | MANual}`

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected), the :ANALyze:SNDR:CORRected:VERTical command selects either automatic or manual vertical scaling.

Query `:ANALyze:SNDR:CORRected:VERTical?`

The :ANALyze:SNDR:CORRected:VERTical? query returns the vertical scaling selection.

Returned Format `<setting><NL>`

`<setting> ::= {AUTO | MAN}`

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:VERTical:OFFSet" on page 362
- ":ANALyze:SNDR:CORRected:VERTical:SCALe" on page 363

History New in version 11.30.

:ANALyze:SNDR:CORRected:VERTical:OFFSet

Command `:ANALyze:SNDR:CORRected:VERTical:OFFSet <offset_value>`

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected) and manual vertical scaling is selected (see :ANALyze:SNDR:CORRected:VERTical), the :ANALyze:SNDR:CORRected:VERTical:OFFSet command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.

<offset_value> A real number for the vertical center-of-the-screen value relative to the Pulse Corrected waveform's 0V vertical reference.

Query `:ANALyze:SNDR:CORRected:VERTical:OFFSet?`

The :ANALyze:SNDR:CORRected:VERTical:OFFSet? query returns the vertical center-of-the-screen value relative to the Pulse Corrected waveform's 0V vertical reference.

Returned Format `<offset_value><NL>`

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:VERTical" on page 361
- ":ANALyze:SNDR:CORRected:VERTical:SCALe" on page 363

History New in version 11.30.

:ANALyze:SNDR:CORRected:VERTical:SCALe

Command

```
:ANALyze:SNDR:CORRected:VERTical:SCALe <scale_value>
```

When the Pulse Corrected waveform is displayed (see :ANALyze:SNDR:CORRected) and manual vertical scaling is selected (see :ANALyze:SNDR:CORRected:VERTical), the :ANALyze:SNDR:CORRected:VERTical:SCALe command specifies the waveform's vertical volts per division.

<scale_value> A real number for the Pulse Corrected waveform's vertical scale in volts per division.

Query

```
:ANALyze:SNDR:CORRected:VERTical:SCALe?
```

The :ANALyze:SNDR:CORRected:VERTical:SCALe? query returns the Pulse Corrected waveform's vertical scale in volts per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:CORRected:VERTical" on page 361
- ":ANALyze:SNDR:CORRected:VERTical:OFFSet" on page 362

History New in version 11.30.

:ANALyze:SNDR:ERRor

Command

```
:ANALyze:SNDR:ERRor {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SNDR:ERRor command lets you enable or disable the display of the linear fit Error e(k) waveform.

Query

```
:ANALyze:SNDR:ERRor?
```

The :ANALyze:SNDR:ERRor? query returns whether the linear fit Error e(k) waveform display is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:ERRor:VERTical" on page 365
- ":ANALyze:SNDR:ERRor:VERTical:OFFSet" on page 366
- ":ANALyze:SNDR:ERRor:VERTical:SCALe" on page 367

History

New in version 11.30.

:ANALyze:SNDR:ERRor:VERTical

Command

```
:ANALyze:SNDR:ERRor:VERTical {AUTO | MANual}
```

When the linear fit Error e(k) waveform is displayed (see :ANALyze:SNDR:ERRor), the :ANALyze:SNDR:ERRor:VERTical command command selects either automatic or manual vertical scaling.

Query

```
:ANALyze:SNDR:ERRor:VERTical?
```

The :ANALyze:SNDR:ERRor:VERTical? query returns the vertical scaling selection.

Returned Format

```
<setting><NL>
```

```
<setting> ::= {AUTO | MAN}
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:ERRor:VERTical:OFFSet" on page 366
- ":ANALyze:SNDR:ERRor:VERTical:SCALe" on page 367

History New in version 11.30.

:ANALyze:SNDR:ERRor:VERTical:OFFSet

Command

```
:ANALyze:SNDR:ERRor:VERTical:OFFSet <offset_value>
```

When the linear fit Error e(k) waveform is displayed (see :ANALyze:SNDR:ERRor) and manual vertical scaling is selected (see :ANALyze:SNDR:ERRor:VERTical), the :ANALyze:SNDR:ERRor:VERTical:OFFSet command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.

<offset_value> A real number for the vertical center-of-the-screen value relative to the linear fit Error e(k) waveform's 0V vertical reference.

Query

```
:ANALyze:SNDR:ERRor:VERTical:OFFSet?
```

The :ANALyze:SNDR:ERRor:VERTical:OFFSet? query returns the vertical center-of-the-screen value relative to the linear fit Error e(k) waveform's 0V vertical reference.

Returned Format

```
<offset_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:ERRor:VERTical" on page 365
- ":ANALyze:SNDR:ERRor:VERTical:SCALe" on page 367

History New in version 11.30.

:ANALyze:SNDR:ERRor:VERTical:SCALe

Command

```
:ANALyze:SNDR:ERRor:VERTical:SCALe <scale_value>
```

When the linear fit Error e(k) waveform is displayed (see :ANALyze:SNDR:ERRor) and manual vertical scaling is selected (see :ANALyze:SNDR:ERRor:VERTical), the :ANALyze:SNDR:ERRor:VERTical:SCALe command specifies the waveform's vertical volts per division.

<scale_value> A real number for the Error waveform's vertical scale in volts per division.

Query

```
:ANALyze:SNDR:ERRor:VERTical:SCALe?
```

The :ANALyze:SNDR:ERRor:VERTical:SCALe? query returns the Error waveform's vertical scale in volts per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:ERRor:VERTical" on page 365
- ":ANALyze:SNDR:ERRor:VERTical:OFFSet" on page 366

History New in version 11.30.

:ANALyze:SNDR:LFPR

Command

```
:ANALyze:SNDR:LFPR {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SNDR:LFPR command lets you enable or disable the display of the Linear Fit Pulse Response p(k) waveform.

Query

```
:ANALyze:SNDR:LFPR?
```

The :ANALyze:SNDR:LFPR? query returns whether the Linear Fit Pulse Response p(k) waveform display is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR:HORizontal" on page 369
- ":ANALyze:SNDR:LFPR:HORizontal:POSition" on page 370
- ":ANALyze:SNDR:LFPR:HORizontal:SCALe" on page 371
- ":ANALyze:SNDR:LFPR:VERTical" on page 372
- ":ANALyze:SNDR:LFPR:VERTical:OFFSet" on page 373
- ":ANALyze:SNDR:LFPR:VERTical:SCALe" on page 374

History New in version 11.30.

:ANALyze:SNDR:LFPR:HORizontal

Command

```
:ANALyze:SNDR:LFPR:HORizontal {AUTO | MANual}
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see :ANALyze:SNDR:LFPR), the :ANALyze:SNDR:LFPR:HORizontal command selects either automatic or manual horizontal scaling.

Query

```
:ANALyze:SNDR:LFPR:HORizontal?
```

The :ANALyze:SNDR:LFPR:HORizontal? query returns the horizontal scaling selection.

Returned Format

```
<setting><NL>

<setting> ::= {AUTO | MAN}
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:HORizontal:POSition" on page 370
- ":ANALyze:SNDR:LFPR:HORizontal:SCALe" on page 371

History

New in version 11.30.

:ANALyze:SNDR:LFPR:HORizontal:POSition

Command

```
:ANALyze:SNDR:LFPR:HORizontal:POSition <position_value>
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see) and manual horizontal scaling is selected (see), the :ANALyze:SNDR:LFPR:HORizontal:POSition command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.

<position_value> A real number for the horizontal timebase reference's time value relative to the Linear Fit Pulse Response waveform's t=0 location.

Query

```
:ANALyze:SNDR:LFPR:HORizontal:POSition?
```

The :ANALyze:SNDR:LFPR:HORizontal:POSition? query returns the horizontal timebase reference's time value relative to the Linear Fit Pulse Response waveform's t=0 location.

Returned Format

```
<position_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:HORizontal" on page 369
- ":ANALyze:SNDR:LFPR:HORizontal:SCALe" on page 371

History New in version 11.30.

:ANALyze:SNDR:LFPR:HORizontal:SCALe

Command

```
:ANALyze:SNDR:LFPR:HORizontal:SCALe <scale_value>
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see :ANALyze:SNDR:LFPR) and manual horizontal scaling is selected (see :ANALyze:SNDR:LFPR:HORizontal), the :ANALyze:SNDR:LFPR:HORizontal:SCALe command specifies the waveform's horizontal time per division.

<scale_value> A real number for the Linear Fit Pulse Response waveform's horizontal scale in seconds per division.

Query

```
:ANALyze:SNDR:LFPR:HORizontal:SCALe?
```

The :ANALyze:SNDR:LFPR:HORizontal:SCALe? query returns the Linear Fit Pulse Response waveform's horizontal scale in seconds per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:HORizontal" on page 369
- ":ANALyze:SNDR:LFPR:HORizontal:POSition" on page 370

History New in version 11.30.

:ANALyze:SNDR:LFPR:VERTical

Command

```
:ANALyze:SNDR:LFPR:VERTical {AUTO | MANual}
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see :ANALyze:SNDR:LFPR), the :ANALyze:SNDR:LFPR:VERTical command selects either automatic or manual vertical scaling.

Query

```
:ANALyze:SNDR:LFPR:VERTical?
```

The :ANALyze:SNDR:LFPR:VERTical? query returns the vertical scaling selection.

Returned Format

```
<setting><NL>

<setting> ::= {AUTO | MAN}
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:VERTical:OFFSet" on page 373
- ":ANALyze:SNDR:LFPR:VERTical:SCALe" on page 374

History

New in version 11.30.

:ANALyze:SNDR:LFPR:VERTical:OFFSet

Command

```
:ANALyze:SNDR:LFPR:VERTical:OFFSet <offset_value>
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see :ANALyze:SNDR:LFPR) and manual vertical scaling is selected (see :ANALyze:SNDR:LFPR:VERTical), the :ANALyze:SNDR:LFPR:VERTical:OFFSet command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.

<offset_value> A real number for the vertical center-of-the-screen value relative to the Linear Fit Pulse Response waveform's 0V vertical reference.

Query

```
:ANALyze:SNDR:LFPR:VERTical:OFFSet?
```

The :ANALyze:SNDR:LFPR:VERTical:OFFSet? query returns the vertical center-of-the-screen value relative to the Linear Fit Pulse Response waveform's 0V vertical reference.

Returned Format

```
<offset_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:VERTical" on page 372
- ":ANALyze:SNDR:LFPR:VERTical:SCALe" on page 374

History New in version 11.30.

:ANALyze:SNDR:LFPR:VERTical:SCALe

Command

```
:ANALyze:SNDR:LFPR:VERTical:SCALe <scale_value>
```

When the Linear Fit Pulse Response p(k) waveform is displayed (see :ANALyze:SNDR:LFPR) and manual vertical scaling is selected (see :ANALyze:SNDR:LFPR:VERTical), the :ANALyze:SNDR:LFPR:VERTical:SCALe command specifies the waveform's vertical volts per division.

<scale_value> A real number for the Linear Fit Pulse Response waveform's vertical scale in volts per division.

Query

```
:ANALyze:SNDR:LFPR:VERTical:SCALe?
```

The :ANALyze:SNDR:LFPR:VERTical:SCALe? query returns the Linear Fit Pulse Response waveform's vertical scale in volts per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:LFPR:VERTical" on page 372
- ":ANALyze:SNDR:LFPR:VERTical:OFFSet" on page 373

History New in version 11.30.

:ANALyze:SNDR:INPut

Command

```
:ANALyze:SNDR:INPut {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SNDR:INPut command lets you enable or disable the display of the SNDR Input waveform.

The SNDR Input waveform is the source input waveform with the pattern averaging (see :ANALyze:SNDR:INPut:NAVerages) and points-per-UI (see :ANALyze:SNDR:INPut:PPUI) linear fit math parameters applied.

Query

```
:ANALyze:SNDR:INPut?
```

The :ANALyze:SNDR:INPut? query returns whether the SNDR Input waveform display is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:HORizontal" on page 376
- ":ANALyze:SNDR:INPut:HORizontal:POSition" on page 377
- ":ANALyze:SNDR:INPut:HORizontal:SCALe" on page 378
- ":ANALyze:SNDR:INPut:VERTical" on page 381
- ":ANALyze:SNDR:INPut:VERTical:OFFSet" on page 382
- ":ANALyze:SNDR:INPut:VERTical:SCALe" on page 383

History

New in version 11.30.

:ANALyze:SNDR:INPut:HORizontal

Command `:ANALyze:SNDR:INPut:HORizontal {AUTO | MANual}`

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut), the :ANALyze:SNDR:INPut:HORizontal command selects either automatic or manual horizontal scaling.

Query `:ANALyze:SNDR:INPut:HORizontal?`

The :ANALyze:SNDR:INPut:HORizontal? query returns the horizontal scaling selection.

Returned Format `<setting><NL>`

`<setting> ::= {AUTO | MAN}`

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:HORizontal:POSition" on page 377
- ":ANALyze:SNDR:INPut:HORizontal:SCALe" on page 378

History New in version 11.30.

:ANALyze:SNDR:INPut:HORizontal:POSition

Command

```
:ANALyze:SNDR:INPut:HORizontal:POSition <position_value>
```

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut) and manual horizontal scaling is selected (see :ANALyze:SNDR:INPut:HORizontal), the :ANALyze:SNDR:INPut:HORizontal:POSition command specifies the horizontal timebase reference's time value relative to the waveform's t=0 location.

<position_value> A real number for the horizontal timebase reference's time value relative to the SNDR Input waveform's t=0 location.

Query

```
:ANALyze:SNDR:INPut:HORizontal:POSition?
```

The :ANALyze:SNDR:INPut:HORizontal:POSition? query returns the horizontal timebase reference's time value relative to the SNDR Input waveform's t=0 location.

Returned Format

```
<position_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:HORizontal" on page 376
- ":ANALyze:SNDR:INPut:HORizontal:SCALe" on page 378

History New in version 11.30.

:ANALyze:SNDR:INPut:HORizontal:SCALe

Command

```
:ANALyze:SNDR:INPut:HORizontal:SCALe <scale_value>
```

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut) and manual horizontal scaling is selected (see :ANALyze:SNDR:INPut:HORizontal), the :ANALyze:SNDR:INPut:HORizontal:SCALe command specifies the waveform's horizontal time per division.

<scale_value> A real number for the SNDR Input waveform's horizontal scale in seconds per division.

Query

```
:ANALyze:SNDR:INPut:HORizontal:SCALe?
```

The :ANALyze:SNDR:INPut:HORizontal:SCALe? query returns the SNDR Input waveform's horizontal scale in seconds per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:HORizontal" on page 376
- ":ANALyze:SNDR:INPut:HORizontal:POSition" on page 377

History New in version 11.30.

:ANALyze:SNDR:INPut:NAVerages

Command

```
:ANALyze:SNDR:INPut:NAVerages <count>
```

The :ANALyze:SNDR:INPut:NAVerages command specifies the number of patterns to average in the linear fit math function computation.

The SNDR Input waveform (see :ANALyze:SNDR:INPut) will reflect the specified number of patterns averaged.

<count> An integer number of averages from 2-65534.

Query

```
:ANALyze:SNDR:INPut:NAVerages?
```

The :ANALyze:SNDR:INPut:NAVerages? query returns the specified number of patterns averaged in the linear fit math function computation.

Returned Format

```
<count><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:ANALyze:SNDR:INPut:PPUI

Command

```
:ANALyze:SNDR:INPut:PPUI <points_per_UI>
```

The :ANALyze:SNDR:INPut:PPUI command specifies the number of sample points per UI (symbol) to use in the linear fit math function computation.

The SNDR Input waveform (see :ANALyze:SNDR:INPut) will have the specified number of sample points per UI.

<points_per_UI> An integer number of points per unit interval (UI) from 8-1024.

Query

```
:ANALyze:SNDR:INPut:PPUI?
```

The :ANALyze:SNDR:INPut:PPUI? query returns the specified number of sample points per UI (symbol) used in the linear fit math function computation.

Returned Format

```
<points_per_UI><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:ANALyze:SNDR:INPut:VERTical

Command

```
:ANALyze:SNDR:INPut:VERTical {AUTO | MANual}
```

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut), the :ANALyze:SNDR:INPut:VERTical command selects either automatic or manual vertical scaling.

Query

```
:ANALyze:SNDR:INPut:VERTical?
```

The :ANALyze:SNDR:INPut:VERTical? query returns the vertical scaling selection.

Returned Format

```
<setting><NL>
```

```
<setting> ::= {AUTO | MAN}
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:VERTical:OFFSet" on page 382
- ":ANALyze:SNDR:INPut:VERTical:SCALe" on page 383

History New in version 11.30.

:ANALyze:SNDR:INPut:VERTical:OFFSet

Command

```
:ANALyze:SNDR:INPut:VERTical:OFFSet <offset_value>
```

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut) and manual vertical scaling is selected (see :ANALyze:SNDR:INPut:VERTical), the :ANALyze:SNDR:INPut:VERTical:OFFSet command specifies the vertical center-of-the-screen value relative to the waveform's 0V vertical reference.

<offset_value> A real number for the vertical center-of-the-screen value relative to the SNDR Input waveform's 0V vertical reference.

Query

```
:ANALyze:SNDR:INPut:VERTical:OFFSet?
```

The :ANALyze:SNDR:INPut:VERTical:OFFSet? query returns the vertical center-of-the-screen value relative to the SNDR Input waveform's 0V vertical reference.

Returned Format

```
<offset_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:VERTical" on page 381
- ":ANALyze:SNDR:INPut:VERTical:SCALe" on page 383

History New in version 11.30.

:ANALyze:SNDR:INPut:VERTical:SCALe

Command

```
:ANALyze:SNDR:INPut:VERTical:SCALe <scale_value>
```

When the SNDR Input waveform is displayed (see :ANALyze:SNDR:INPut) and manual vertical scaling is selected (see :ANALyze:SNDR:INPut:VERTical), the :ANALyze:SNDR:INPut:VERTical:SCALe command specifies the waveform's vertical volts per division.

<scale_value> A real number for the SNDR Input waveform's vertical scale in volts per division.

Query

```
:ANALyze:SNDR:INPut:VERTical:SCALe?
```

The :ANALyze:SNDR:INPut:VERTical:SCALe? query returns the SNDR Input waveform's vertical scale in volts per division.

Returned Format

```
<scale_value><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:VERTical" on page 381
- ":ANALyze:SNDR:INPut:VERTical:OFFSet" on page 382

History New in version 11.30.

:ANALyze:SNDR:LFPDelay

Command

```
:ANALyze:SNDR:LFPDelay <integer>
```

The :ANALyze:SNDR:LFPDelay command sets the linear fit math function's pulse delay parameter. This parameter is used in the filter to compute the linear fit pulse response. It is sometimes denoted using D_p.

<integer> An integer for the Linear Fit Pulse Delay from 0-20.

Query

```
:ANALyze:SNDR:LFPDelay?
```

The :ANALyze:SNDR:LFPDelay? query returns the linear fit math function's pulse delay parameter.

Returned Format

```
<integer><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:ANALyze:SNDR:LFPLength

Command

```
:ANALyze:SNDR:LFPLength <integer>
```

The :ANALyze:SNDR:LFPLength command sets the linear fit math function's pulse length parameter. This parameter is used in the filter to compute the linear fit pulse response. It is sometimes denoted using N_p.

<integer> An integer for the Linear Fit Pulse Length from 2-500.

Query

```
:ANALyze:SNDR:LFPLength?
```

The :ANALyze:SNDR:LFPLength? query returns the linear fit math function's pulse length parameter.

Returned Format

```
<integer><NL>
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:ANALyze:SNDR:SOURce

Command

```
:ANALyze:SNDR:SOURce <source>
```

The :ANALyze:SNDR:SOURce command specifies the input waveform source for the PAM-4 Signal to Noise and Distortion Ratio (SNDR) measurements.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L>}

Query

```
:ANALyze:SNDR:SOURce?
```

The :ANALyze:SNDR:SOURce? query returns the currently selected input waveform source for the PAM-4 SNDR measurements.

Returned Format

```
<source><NL>

<source> ::= {CHAN<N> | DIFF<D> | COMM<C> | WMEM<R> | FUNC<F> | EQU<L>}
```

See Also

- ":MEASure:SNDR" on page 1122
- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:ANALyze:SIGNal:DATarate

Command

```
:ANALyze:SIGNal:DATarate <source>,<data_rate>
```

When the source signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the :ANALyze:SIGNal:DATarate command specifies the data rate of the signal.

With PAM-4, the data rate is twice the symbol rate because each voltage level represents two bits of data. Changing the data rate also changes the symbol rate (see :ANALyze:SIGNal:SYMBolrate) and vice-versa.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<data_rate> Bits/second in NR3 format.

Query

```
:ANALyze:SIGNal:DATarate? <source>
```

The :ANALyze:SIGNal:DATarate? query returns the data rate of the source signal.

Returned Format

```
[:ANALyze:SIGNal:DATarate] <data_rate><NL>

<data_rate> ::= bits/second in NR3 format.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

:ANALyze:SIGNal:MIXer:CABLeloss

Command `:ANALyze:SIGNal:MIXer:CABLeloss <source>,<loss>`

The :ANALyze:SIGNal:MIXer:CABLeloss command sets the loss of the cable connecting the mixer to the oscilloscope.

<source> {CHANnel<N>}

<loss> The dB cable return loss in NR3 format.

Query `:ANALyze:SIGNal:MIXer:CABLeloss? <source>`

The :ANALyze:SIGNal:MIXer:CABLeloss? query returns the specified cable loss.

Returned Format `<loss><NL>`

See Also

- ":ANALyze:SIGNal:MMWave:CALibrate" on page 390
- ":ANALyze:SIGNal:MMWave:CFRequency" on page 391
- ":ANALyze:SIGNal:MMWave:CONNect" on page 392
- ":ANALyze:SIGNal:MMWave:LOADdress" on page 393
- ":ANALyze:SIGNal:MMWave:MBANdwidth" on page 394

History New in version 5.60.

:ANALyze:SIGNal:MMWave:CALibrate

Command `:ANALyze:SIGNal:MMWave:CALibrate`

The :ANALyze:SIGNal:MMWave:CALibrate command initiates a mixer/LO (local oscillator) power calibration.

The LO is identified by the :ANALyze:SIGNal:MMWave:LOADdress command.

The calibration takes about a minute.

You may send additional commands that will not be executed until this command completes. (The command is blocking.)

You may initiate a query (*IDN?, etc.) to wait until the calibration is complete.

See Also

- ":ANALyze:SIGNal:MIXer:CABLeloss" on page 389
- ":ANALyze:SIGNal:MMWave:CFRequency" on page 391
- ":ANALyze:SIGNal:MMWave:CONNect" on page 392
- ":ANALyze:SIGNal:MMWave:LOADdress" on page 393
- ":ANALyze:SIGNal:MMWave:MBANdwidth" on page 394

History New in version 5.60.

:ANALyze:SIGNal:MMWave:CFRequency

Command `:ANALyze:SIGNal:MMWave:CFRequency <center_freq>`

The :ANALyze:SIGNal:MMWave:CFRequency command sets the center frequency for the oscilloscope's FFT math function.

Because all channels using mmWave share a LO, there is only one center frequency for the oscilloscope.

<center_freq> Center frequency in NR3 format.

Query `:ANALyze:SIGNal:MMWave:CFRequency?`

The :ANALyze:SIGNal:MMWave:CFRequency? query returns the specified center frequency.

Returned Format `<center_freq><NL>`

See Also

- ":ANALyze:SIGNal:MIXer:CABLeloss" on page 389
- ":ANALyze:SIGNal:MMWave:CALibrate" on page 390
- ":ANALyze:SIGNal:MMWave:CONNect" on page 392
- ":ANALyze:SIGNal:MMWave:LOADdress" on page 393
- ":ANALyze:SIGNal:MMWave:MBANdwidth" on page 394

History New in version 5.60.

:ANALyze:SIGNal:MMWave:CONNect

Command

```
:ANALyze:SIGNal:MMWave:CONNect <source>,{{0 | OFF} | {1 | ON}}
```

The :ANALyze:SIGNal:MMWave:CONNect command sets the connection status for the mixer and LO (local oscillator) assigned to the specified channel. Use OFF to disconnect.

<source> {CHANnel<N>}

Query

```
:ANALyze:SIGNal:MMWave:CONNect? <source>
```

The :ANALyze:SIGNal:MMWave:CONNect? query returns the mixer and LO connection status.

Returned Format

```
<status><NL>

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

See Also

- ":ANALyze:SIGNal:MIXer:CABLeloss" on page 389
- ":ANALyze:SIGNal:MMWave:CALibrate" on page 390
- ":ANALyze:SIGNal:MMWave:CFRequency" on page 391
- ":ANALyze:SIGNal:MMWave:LOADdress" on page 393
- ":ANALyze:SIGNal:MMWave:MBANdwidth" on page 394

History New in version 5.60.

:ANALyze:SIGNal:MMWave:LOADdress

Command

```
:ANALyze:SIGNal:MMWave:LOADdress <string>
```

The :ANALyze:SIGNal:MMWave:LOADdress command sets the LO's (local oscillator's) VISA address.

The VISA address of the LO can be found in the Keysight Connection Expert.

<string> Quoted VISA address of LO.

Query

```
:ANALyze:SIGNal:MMWave:LOADdress?
```

The :ANALyze:SIGNal:MMWave:LOADdress? query returns the specified LO VISA address.

Returned Format

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

See Also

- ":ANALyze:SIGNal:MIXer:CABLeloss" on page 389
- ":ANALyze:SIGNal:MMWave:CALibrate" on page 390
- ":ANALyze:SIGNal:MMWave:CFRequency" on page 391
- ":ANALyze:SIGNal:MMWave:CONNect" on page 392
- ":ANALyze:SIGNal:MMWave:MBANdwidth" on page 394

History New in version 5.60.

:ANALyze:SIGNal:MMWave:MBANdwidth

Command

```
:ANALyze:SIGNal:MMWave:MBANdwidth <meas_bandwidth>
```

The :ANALyze:SIGNal:MMWave:MBANdwidth command sets the measurement bandwidth (in Hz).

The specified bandwidth becomes the center frequency for the oscilloscope's FFT math function.

Because all channels using mmWave will share a LO (local oscillator), there is only one measurement bandwidth for the oscilloscope.

<meas_bandwidth> Bandwidth in NR3 format.

Query

```
:ANALyze:SIGNal:MMWave:MBANdwidth?
```

The :ANALyze:SIGNal:MMWave:MBANdwidth? query returns the specified measurement bandwidth.

Returned Format

```
<meas_bandwidth><NL>
```

See Also

- ":ANALyze:SIGNal:MIXer:CABLeloss" on page 389
- ":ANALyze:SIGNal:MMWave:CALibrate" on page 390
- ":ANALyze:SIGNal:MMWave:CFRequency" on page 391
- ":ANALyze:SIGNal:MMWave:CONNect" on page 392
- ":ANALyze:SIGNal:MMWave:LOADdress" on page 393

History New in version 5.60.

:ANALyze:SIGNal:PATTern:CLEar

Command

```
:ANALyze:SIGNal:PATTern:CLEar <source>
```

For NRZ and PAM4 signal types (see :ANALyze:SIGNal:TYPE), the :ANALyze:SIGNal:PATTern:CLEar command clears a loaded pattern file and goes back to an automatic or manual pattern length setting.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:INVert" on page 396
- ":ANALyze:SIGNal:PATTern:REVerse" on page 400
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History New in version 6.20.

:ANALyze:SIGNal:PATTern:INVert

Command

```
:ANALyze:SIGNal:PATTern:INVert {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SIGNal:PATTern:INVert command enables or disables inverting a PRBS or loaded pattern.

Query

```
:ANALyze:SIGNal:PATTern:INVert?
```

The :ANALyze:SIGNal:PATTern:INVert? query returns whether the PRBS or loaded pattern invert setting is on or off.

Returned Format

```
<setting><NL>

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

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:REVerse" on page 400
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History

New in version 11.15.

:ANALyze:SIGNal:PATTern:LOAD

Command `:ANALyze:SIGNal:PATTern:LOAD <source>,"<pattern_file_path>"`

For NRZ and PAM4 signal types (see :ANALyze:SIGNal:TYPE), the :ANALyze:SIGNal:PATTern:LOAD command loads a pattern file from which pattern lengths and patterns are determined.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<pattern_file_path> Quoted string that is the path of the pattern file.

The Infiniium oscilloscope software includes some NRZ PRBS pattern files that are also included with BERTs.

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:SIGNal:PATTern:INVert" on page 396
- ":ANALyze:SIGNal:PATTern:REVerse" on page 400
- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History New in version 6.20.

:ANALyze:SIGNal:PATTern:PLENgth

Command

```
:ANALyze:SIGNal:PATTern:PLENgth <source>,{AUTO | <pattern_length>
    | P5M1 | P6M1 | P7M1 | P8M1 | P9M1 | P10M1 | P11M1 | P12M1 | P13M1
    | P14M1 | P15M1}
```

For NRZ and PAM4 signal types (see :ANALyze:SIGNal:TYPE), the :ANALyze:SIGNal:PATTern:PLENgth command specifies that the oscilloscope determine the pattern length automatically, manually specifies a pattern length, or specifies a PRBS pattern.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

AUTO The oscilloscope automatically determines the pattern length and pattern by looking for at least two error-free copies of an identical repeating bit pattern in acquisition memory.

<pattern_length> Manually specifies the pattern length as an integer number of symbols from 2 to 2^23. In this case, to determine the pattern, the oscilloscope looks in acquisition memory for at least two error-free copies of an identical repeating bit pattern of the specified length.

PRBS Pattern These options specify PRBS patterns:

- P5M1 — PRBS 2^5-1
- P6M1 — PRBS 2^6-1
- P7M1 — PRBS 2^7-1
- P8M1 — PRBS 2^8-1
- P9M1 — PRBS 2^9-1
- P10M1 — PRBS 2^10-1
- P11M1 — PRBS 2^11-1
- P12M1 — PRBS 2^12-1
- P13M1 — PRBS 2^13-1
- P14M1 — PRBS 2^14-1
- P15M1 — PRBS 2^15-1

Query

```
:ANALyze:SIGNal:PATTern:PLENgth? <source>
```

The :ANALyze:SIGNal:PATTern:PLENgth? query returns the specified pattern length or PRBS pattern setting.

Returned Format

```
<pattern_length><NL>

<pattern_length> ::= {AUTO | 2 to 2^23 | P5M1 | P6M1 | P7M1 | P8M1
    | P9M1 | P10M1 | P11M1 | P12M1 | P13M1 | P14M1 | P15M1}
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:SIGNal:PATTern:INVert" on page 396
- ":ANALyze:SIGNal:PATTern:REVerse" on page 400
- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History New in version 6.20.

Version 10.25: There are now additional P5M1, P6M1, P7M1, P8M1, P9M1, P10M1, P11M1, P12M1, P13M1, P14M1, and P15M1 options for specifying PRBS patterns.

:ANALyze:SIGNal:PATTern:REVerse

Command

```
:ANALyze:SIGNal:PATTern:REVerse {{0 | OFF} | {1 | ON}}
```

The :ANALyze:SIGNal:PATTern:REVerse command enables or disables reversing a PRBS or loaded pattern.

Query

```
:ANALyze:SIGNal:PATTern:REVerse?
```

The :ANALyze:SIGNal:PATTern:REVerse? query returns whether the PRBS or loaded pattern reverse setting is on or off.

Returned Format

```
<setting><NL>
```

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

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:INVert" on page 396
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History New in version 11.15.

:ANALyze:SIGNal:PATTern:SMAP

Command

```
:ANALyze:SIGNal:PATTern:SMAP <source>,{UNCoded | GRAYcoded}
```

For PAM4 signal types (see :ANALyze:SIGNal:TYPE), the :ANALyze:SIGNal:PATTern:SMAP command specifies whether the symbol map is gray-coded or uncoded.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

Query

```
:ANALyze:SIGNal:PATTern:SMAP? <source>
```

The :ANALyze:SIGNal:PATTern:SMAP? query returns the symbol map setting.

Returned Format

```
{UNC | GRAY}<NL>
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History

New in version 6.20.

:ANALyze:SIGNal:SYMBolrate

Command

```
:ANALyze:SIGNal:SYMBolrate <source>,<symbol_rate>
```

When the source signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), this command specifies the edge rate of the signal. The inverse of this rate is the unit interval (UI).

With PAM-4, the data rate is twice the symbol rate because each voltage level represents two bits of data. Changing the symbol rate also changes the data rate (see :ANALyze:SIGNal:DATarate) and vice-versa.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

<symbol_rate> Baud in NR3 format.

Query

```
:ANALyze:SIGNal:SYMBolrate? <source>
```

The :ANALyze:SIGNal:SYMBolrate? query returns the symbol rate for the specified source signal.

Returned Format

```
[:ANALyze:SIGNal:SYMBolrate] <symbol_rate><NL>

<symbol_rate> ::= baud in NR3 format.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

:ANALyze:SIGNal:TYPE

Command

```
:ANALyze:SIGNal:TYPE <source>,{UNSPecified | NRZ | PAM4 | PAM3 | PAM6
    | PAM8 | CPHY, <source_B-C>, <source_C-A> | FEXTension | SPECtral}
```

The :ANALyze:SIGNal:TYPE command lets you specify whether a channel, function, or waveform memory is a special type of signal, like a PAM-4 signal for example.

- UNSPecified — When a signal type is unspecified, the oscilloscope's digital signal analysis and measurement features assume a NRZ signal with two levels (high and low).
- NRZ — With this selection, you are able to specify pattern lengths and patterns so that you can make BER measurements on NRZ signals.
- PAM4 — When a signal is specified as a PAM-4 (4-level Pulse-Amplitude Modulation) signal:
  - It changes how the oscilloscope determines voltage levels. Three thresholds are used to distinguish between the four voltage levels.
  - It changes how the oscilloscope represents the data rate. Two bits of data are represented by each voltage level.
  - The term *symbol rate* is used to describe the clock edge rate of the signal. The inverse of this rate is the unit interval (UI).
  - There are multiple edges to consider in clock recovery.
  - It changes how eye measurements, voltage level measurements, and rise/fall time measurements are presented because of the multiple eyes, levels, and edges.
- PAM3 — When a signal is specified as a PAM-3 (3-level Pulse-Amplitude Modulation) signal:
  - It changes how the oscilloscope determines voltage levels. Two thresholds are used to distinguish between the three voltage levels.
  - It changes how the oscilloscope represents the data rate. Two bits of data are represented by each voltage level (00, 01, or 10).
  - The term *symbol rate* is used to describe the clock edge rate of the signal. The inverse of this rate is the unit interval (UI).
  - There are multiple edges to consider in clock recovery.
  - It changes how eye measurements, voltage level measurements, and rise/fall time measurements are presented because of the multiple eyes, levels, and edges.
- PAM6 — When a signal is specified as a PAM-6 (6-level Pulse-Amplitude Modulation) signal:
  - It changes how the oscilloscope determines voltage levels. Five thresholds are used to distinguish between the six voltage levels.

  - It changes how the oscilloscope represents the data rate. Three bits of data are represented by each voltage level (000, 001, 010, 100, 101, or 110).
  - The term *symbol rate* is used to describe the clock edge rate of the signal. The inverse of this rate is the unit interval (UI).
  - There are multiple edges to consider in clock recovery.
  - It changes how eye measurements, voltage level measurements, and rise/fall time measurements are presented because of the multiple eyes, levels, and edges.
- PAM8 — When a signal is specified as a PAM-8 (8-level Pulse-Amplitude Modulation) signal:
  - It changes how the oscilloscope determines voltage levels. Seven thresholds are used to distinguish between the eight voltage levels.
  - It changes how the oscilloscope represents the data rate. Three bits of data are represented by each voltage level (000, 001, 010, 011, 100, 101, 110, or 111).
  - The term *symbol rate* is used to describe the clock edge rate of the signal. The inverse of this rate is the unit interval (UI).
  - There are multiple edges to consider in clock recovery.
  - It changes how eye measurements, voltage level measurements, and rise/fall time measurements are presented because of the multiple eyes, levels, and edges.
- CPHY — Supports decode and analysis of MIPI C-PHY signals.

  When CPHY is selected, the first <source> option is the A-B source.
- FEXTension — (Available in MXR-Series oscilloscopes only) Enables the Frequency Extension option that lets you use a 2 GHz span of oscilloscope bandwidth at frequencies up to the physical limit of the oscilloscope hardware.
- SPECtral — (Available in MXR-Series oscilloscopes only) Enables the Spectrum Analysis (DDC) option that switches the oscilloscope into a frequency domain hardware digital down-conversion (DDC) mode of operation that produces complex sample data for use in spectral analysis applications like Keysight's PathWave Vector Signal Analysis (89600 VSA) software.

<source>, <source_B-C>, <source_C-A> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Sources for Analyze Commands" on page 331.

Query

```
:ANALyze:SIGNal:TYPE? <source>
```

The :ANALyze:SIGNal:TYPE? query returns a channel, function, or waveform memory's signal type.

Returned Format

```
[:ANALyze:SIGNal:TYPE] <type><NL>
```

```
<type> ::= {UNSP | NRZ | PAM4 | PAM3 | PAM6 | PAM8 | CPHY | FEXT | SPEC}
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176
- ":CHANnel<N>:SPECtral:CFRequency" on page 504
- ":CHANnel<N>:SPECtral:SPAN (MXR-Series)" on page 506
- ":CHANnel<N>:ISIM:BPASs:CFRequency" on page 454
- ":CHANnel<N>:ISIM:BPASs:SPAN?" on page 455

History

New in version 5.50.

Version 5.60: Added the MMWave signal type.

Version 6.10: Added the CPHY signal type.

Version 10.10: Added the PAM3 and FEXTension signal types.

Version 10.12: Added the SPECtral signal type.

Version 11.20: Added the PAM6 and PAM8 signal types.

:ANALyze:VIEW

Command

```
:ANALyze:VIEW <data>
```

```
<data> ::= {ALL | MAIN}
```

The :ANALyze:VIEW command specify whether to use the data on screen or the entire acquisition for measurements, functions, and analysis.

This command maps to the "Window All Data" control in the user interface's Measurement Setup dialog box.

When all edges in an acquisition are used for horizontal measurements (see :ANALyze:AEDGes), this command's setting becomes "ALL" and the entire acquisition is used for measurements, functions, and analysis.

Query

```
:ANALyze:VIEW?
```

The :ANALyze:VIEW? query returns the value that is currently set.

Returned Format

```
[:ANALyze:VIEW] <data><NL>
```

```
<data> ::= {ALL | MAIN}
```

See Also

- ":ANALyze:AEDGes" on page 332

History

New in version 5.30.

16 :BUS Commands

NOTE The :BUS:B<N>:TYPE command applies to oscilloscopes with protocol decode licenses installed. The other :BUS<B> commands apply to mixed-signal oscilloscopes (MSOs).

:BUS:B<N>:TYPE

Command

```
:BUS:B<N>:TYPE {<protocol> | <hs_protocol>}
```

NOTE This BUS command only applies to oscilloscopes with protocol decode licenses installed.

The :BUS:B<N>:TYPE command sets the type of protocol being analyzed for a serial bus waveform.

<protocol> {A429 | CAN | CPHY | DDR | E10BASET | E10GBASEKR | E100GBASEKRCR | EPSI | FLEXray | GENRaw | I3C | IIC | JTAG | LIN | MAN | M1553 | MIPI | RFFE | SENT | SPI | SPMI | SPW | SVID | UART | USB2 | EUSB2}

<hs_protocol> {BRR | CSI3 | DIGRf | DPAUX | DVI | E100BASETX | E1000BASET1 | FIBRechannel | {GEN8B10B | GENeric} | INFiniband | JESD204B | LLI | PCI3 | PCI4 | PCI5 | PCIExpress | QSPI | SAS | SATA | SSIC | UFS | UNIPro | USB3 | USB31 | USB32 | USB4 | USB4LS | USB4TUSB32 | USB4TPCI31 | USBPD | XAUI}

<N> An integer, 1-4.

Example This example sets the serial bus waveform number one protocol type to FLEXray.

```
myScope.WriteString ":BUS:B1:TYPE FLEXray"
```

Query

```
:BUS:B<N>:TYPE?
```

The :BUS:B<N>:TYPE? query returns the name of the protocol being used for the serial bus.

Returned Format

```
[:BUS:B<N>:TYPE] {<protocol> | <hs_protocol>}<NL>
```

<protocol> {A429 | CAN | CPHY | DDR | E10BASET | E10GBASEKR | E100GBASEKRCR | ESPI | FLEX | GENR | I3C | IIC | JTAG | LIN | MAN | M1553 | MIPI | RFFE | SENT | SPI | SPMI | SPW | SVID | UART | USB2 | EUSB2}

<hs_protocol> {BRR | CSI3 | DIGR | DPAUX | DVI | E100BASETX | E1000BASET1 | FC | {USER | USER} | INF | JESD204B | LLI | PCI3 | PCI4 | PCI5 | PCIE | QSPI | SAS | SATA | SSIC | UFS | UNIP | USB3 | USB31 | USB32 | USB4 | USB4LS | USB4TUSB32 | USB4TPCI31 | USBPD | XAUI}

See Also

- ":SBUS<N>:HS Commands" on page 1460

History Legacy command (existed before version 3.10).

Version 3.11: Added the MPHY protocol type for the MIPI M-PHY serial decode selection.

Version 5.00: Added support for new protocols.

Version 5.51: Added support for new ESPI, USB31, and USBPD protocols.

Version 5.70: Added support for new A429, M1553, and I3C protocols.

Version 6.00: Added support for new GENRaw and SPMI protocols.

Version 6.10: Added support for new CPHY (MIPI C-PHY), BRR (BroadR-Reach), SPW (SpaceWire), and MAN (Manchester) protocols.

Version 6.20: Added support for new PCI4 (PCI Express Gen4) and SENT (Single Edge Nibble Transmission) protocols.

Version 10.10: Added support for new QSPI (Quad SPI) and USB32 (USB 3.2) protocols.

Version 10.20: Added support for new EUSB2 (eUSB2) protocol.

Version 11.10: Added support for new PCI5 (PCI Express Gen 1-5), USB4, USB4LS, USB4TUSB32 (USB4 Tunnel USB 3.2), and USB4TPCI31 (USB4 Tunnel PCI Express 3.1) protocols.

Version 11.15: Added support for new E1000BASET1 (Automotive Ethernet 1000BaseT1) and DPAUX (DisplayPort Aux) protocols.

:BUS<B>:BIT<M>

Command

```
:BUS<B>:BIT<M> {{ON | 1} | {OFF | 0}}
```

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<B>: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) then the bit is included in the definition. If the parameter is a 0 (OFF) then the bit is excluded from the definition. The digital subsystem must be enabled for this command will work. See ENABle command in the root subsystem.

<B> An integer, 1-4.

<M> An integer, 0-15.

Example This example includes bit 1 as part of the bus 1 definition.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":BUS1:BIT1 ON"
```

Query

```
:BUS<B>:BIT<M>?
```

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

Returned Format

```
[:BUS<B>:BIT<M>] {1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:BITS

Command
```
:BUS<B>:BITS <channel_list>,{ON | OFF| 1 | 0}
```

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<B>: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. The digital subsystem must be enabled for this command will work. See ENABle command in the root subsystem.

<B> An integer, 1-4.

<channel_list> The channel range is from 0 to 15 in the following format.

(@1,5,7,9)	channels 1, 5, 7, and 9 are turned on.
(@1:15)	channels 1 through 15 are turned on.
(@1:5,8,14)	channels 1 through 5, channel 8, and channel 14 are turned on.

NOTE The parenthesizes are part of the expression and are necessary.

Example This example includes bits 1, 2, 4, 5, 6, 7, 8, and 9 as part of the bus 1 definition.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":BUS1:BITS (@1,2,4:9),ON"
```

Query
```
:BUS<B>:BITS?
```

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

Returned Format
```
[:BUS<B>:BITS] <channel_list>,{1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:CLEar

Command `:BUS<B>:CLEar`

NOTE **The BUS commands only apply to the MSO oscilloscopes.**

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

<B> An integer, 1-4.

Example This example excludes all the digital channels from the bus 1 definition.

```
myScope.WriteString ":BUS1:CLEar"
```

History Legacy command (existed before version 3.10).

:BUS<B>:CLOCk

Command
```
:BUS<B>:CLOCk {CHANnel<N> | DIGital<M> | NONE}
```

NOTE **The BUS commands only apply to the MSO oscilloscopes.**

The :BUS<B>:CLOCk command sets the digital or analog channel used as the clock for decoding the bus values.

<B> An integer, 1-4.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15.

Example This example sets the clock to channel 1 for bus 1.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":BUS1:CLOCk CHANnel1"
```

Query
```
:BUS<B>:CLOCK?
```

The :BUS<B>:CLOCk query returns the channel being used for the specified bus.

Returned Format
```
[:BUS<B>:CLOCk] {CHAN<N> | DIG<M> | NONE}<NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:CLOCk:SLOPe

Command

```
:BUS<B>:CLOCk:SLOPe {RISing | FALLing | EITHer}
```

NOTE **The BUS commands only apply to the MSO oscilloscopes.**

The :BUS<B>:CLOCk:SLOPe command sets the clock edge used for decoding the bus values.

<B> An integer, 1-4.

Example This example sets the clock edge to falling for bus 1.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":BUS1:CLOCk:SLOPE FALLING"
```

Query

```
:BUS<B>:CLOCK:SLOPe?
```

The :BUS<B>:CLOCk:SLOPe query returns the clock edge being used for the specified bus.

Returned Format

```
[:BUS<B>:CLOCk:SLOPe] {RISing | FALLing | EITHer}<NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:DISPlay

Command

```
:BUS<B>[:DISPlay] {ON | OFF | 1 | 0}
```

NOTE **The BUS commands only apply to the MSO oscilloscopes.**

The :BUS<B>:DISPlay command enables or disables the view of the selected bus. The digital subsystem must be enabled before this command will work. See the ENABle command in the root subsystem.

<B> An integer, 1-4.

Example This example enables the viewing of bus 1.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":BUS1 ON"
```

Query

```
:BUS<B>[:DISPlay]?
```

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

Returned Format

```
[:BUS<B>] {1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:LABel

Command

```
:BUS<B>:LABel <quoted_string>
```

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<B>: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 16 characters or less, and may contain any commonly used ASCII characters. Labels with more than 16 characters are truncated to 16 characters.

<B> An integer, 1-4.

<quoted_string> A series of 6 or less characters as a quoted ASCII string.

Example This example sets the bus 1 label to Data.

```
myScope.WriteString ":BUS1:LABel ""Data"""
```

Query

```
:BUS<B>:LABel?
```

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

Returned Format

```
[:BUS<B>:LABel] <quoted_string><NL>
```

History Legacy command (existed before version 3.10).

:BUS<B>:READout

Command `:BUS<B>:READout {DECimal | HEX | SIGNed | SYMBol}`

NOTE **The BUS commands only apply to the MSO oscilloscopes.**

The :BUS<B>:READout command changes the format of the numbers displayed in the bus waveform.

<B> An integer, 1-4.

Example This example sets the bus read out to decimal.

```
myScope.WriteString ":BUS1:READout DECimal"
```

Query `:BUS<B>:READout?`

The :BUS<B>:READout? query returns the format of the readout control.

Returned Format `[:BUS<B>:READout] {DECimal | HEX | SIGNed | SYMBol}<NL>`

History Legacy command (existed before version 3.10).

17 :CALibrate (Calibration) Commands

This chapter briefly explains the calibration of the oscilloscope. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used.

The commands in the CALibration subsystem allow you to change the output of the front-panel Aux Out connector, adjust the skew of channels, and check the status of calibration.

These CALibration commands and queries are implemented in the Infiniium oscilloscopes:

This chapter briefly explains the calibration of the oscilloscope. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used.

Oscilloscope Calibration

Oscilloscope calibration establishes calibration factors for the oscilloscope. These factors are stored on the oscilloscope's hard disk.

- Initiate the calibration from the "Utilities Calibration" menu.

You should calibrate the oscilloscope periodically (at least annually), or if the ambient temperature since the last calibration has changed more than ±5 °C. The temperature change since the last calibration is shown on the calibration status screen which is found under the "Utilities Calibration" dialog. It is the line labeled "Calibration Δ Temp: _ °C."

See also the oscilloscope's *User's Guide* has more details about the calibration.

Probe Calibration Probe calibration establishes the gain and offset of a probe that is connected to a channel of the oscilloscope, and applies these factors to the calibration of that channel.

- Initiate probe calibration from the "Setup > Channel > Probes > Calibrate Probe" menu.

  To achieve the specified accuracy (±2%) with a probe connected to a channel, make sure the oscilloscope is calibrated.

  - For probes that the oscilloscope can identify through the probe power connector, like the 1158A, the oscilloscope automatically adjusts the vertical scale factors for that channel even if a probe calibration is not performed.
  - For nonidentified probes, the oscilloscope adjusts the vertical scale factors only if a probe calibration is performed.
- If you do not perform a probe calibration but want to use an unidentified probe, enter the attenuation factor in the "Setup > Channel > Probes > Configure Probing System > User Defined Probe" menu.
  - If the probe being calibrated has an attenuation factor that allows the oscilloscope to adjust the gain (in hardware) to produce even steps in the vertical scale factors, the oscilloscope will do so.
  - If the probe being calibrated has an unusual attenuation, like 3.75, the oscilloscope may have to adjust the vertical scale factors to an unusual number, like 3.75 V/div.

  Typically, probes have standard attenuation factors such as divide by 10, divide by 20, or divide by 100.

:CALibrate:DATE?

Query `:CALibrate:DATE?`

The :CALibrate:DATE? query returns two calibration dates and times:

- The date and time of the last regular user calibration.
- The date and time of the last time scale calibration.

Returned Format `[:CALibrate:DATE] <user_date_time>,<ts_date_time><NL>`

The string returned is formatted like "<day> <month> <year> <hours>:<minutes>:<seconds>,<day> <month> <year> <hours>:<minutes>:<seconds>", for example "31 MAY 2013 12:52:45,4 DEC 2012 10:59:52".

History Legacy command (existed before version 3.10).

:CALibrate:FREQ

Query `:CALibrate:FREQ?`

The :CALibrate:FREQ? query returns the frequency of the AUX OUT signal.

Returned Format `<frequency><NL>`

`<frequency> ::= frequency of AUX OUT signal in Hz`

See Also

- ":CALibrate:OUTPut" on page 425

History New in version 10.00.

:CALibrate:OUTPut

Command

```
:CALibrate:OUTPut {DC,<dc_value> | {AC | PROBecomp} | ZERO | ONE
    | DPULse | NDPulse | FSYNth | TBCLK | CALMSO??? | PRBS23???}
```

The :CALibrate:OUTPut command specifies the signal that is output on the front panel AUX OUT and CAL OUT connectors. The same signal is output on both connectors; however, there is no DC offset on CAL OUT while there is negative DC offset on AUX OUT.

Signal outputs are enabled or disabled using the :CALibrate:OUTPut:AUX and :CALibrate:OUTPut:CAL commands.

The available output signals are:

- DC,<dc_value> — this outputs a DC level. The <dc_value> is a real number from -2.5 V to +2.5 V.
- AC | PROBecomp — outputs a probe compensation square wave (approximately 715 Hz).
- ZERO — outputs a "zero" or low DC level:
  - On AUX OUT this is: -1 V
- ONE — outputs a "one" or high DC level:
  - On AUX OUT this is: 0 V
- DPULse — A 4 ns positive pulse that occurs every 650 ns (1.53 MHz rate).
- NDPULse — The inverse of DPULse. That is, a 4 ns negative pulse that occurs every 650 ns (1.53 MHz rate).
- FSYNth — A 1.077 GHz sine wave.
- TBCLK — The 100 MHz reference clock output.
- CALMSO — The 100 MHz reference clock output appears on AUX OUT, and 4 GHz clock signals are internally routed to the D1 and D7 digital channels.
- PRBS23 — Pseudo-Random Binary Sequence with maximum length 2^{23}-1 bits.

Example This example puts a DC voltage of 1.0 volts on the oscilloscope front-panel AUX OUT connector.

```
myScope.WriteString ":CALibrate:OUTPut DC,1.0"
```

Query

```
:CALibrate:OUTPut?
```

The :CALibrate:OUTPut? query returns the current setup.

Returned Format

```
[:CALibrate:OUTPut] {DC,<dc_value> | AC | ZERO | ONE | DPUL
    | NDP | FSYN | TBCLK | CALMSO | PRBS2^23}
```

Example This example places the current selection into the string variable, strSelection, then prints the variable.

```
Dim strSelection As String   'Dimension variable
myScope.WriteString ":CALibrate:OUTPut?"
```

```
strSelection = myScope.ReadString
Debug.Print strSelection
```

See Also

- ":CALibrate:OUTPut:AUX" on page 427
- ":CALibrate:OUTPut:CAL" on page 429

History

Legacy command (existed before version 3.10).

Version 11.00: The MXR/EXR-Series oscilloscopes have updated output options.

:CALibrate:OUTPut:AUX

Command

```
:CALibrate:OUTPut:AUX {0 | OFF}
```

The :CALibrate:OUTPut:AUX command enables or disables signal output on the oscilloscope's AUX OUT connector.

Because there is only an AUX OUT connector on MXR/EXR-Series oscilloscopes, OFF is the only valid value for this command.

Query

```
:CALibrate:OUTPut:AUX?
```

The :CALibrate:OUTPut:AUX? query returns the AUX OUT enable/disable setting.

Returned Format

```
<setting><NL>

<setting> ::= 0
```

See Also

- ":CALibrate:OUTPut" on page 425
- ":CALibrate:OUTPut:AUX:RTIMe" on page 428
- ":CALibrate:OUTPut:CAL" on page 429

History

New in version 6.00.

Version 11.00: Because there is only an AUX OUT connector on MXR/EXR-Series oscilloscopes, OFF is the only valid value for this command.

:CALibrate:OUTPut:AUX:RTIMe

Command

```
:CALibrate:OUTPut:AUX:RTIMe <risetime>
```

The :CALibrate:OUTPut:AUX:RTIMe command specifies the speed of the Aux Out signal's rise time.

<risetime> {FAST | SLOW}

Query

```
:CALibrate:OUTPut:AUX:RTIMe?
```

The :CALibrate:OUTPut:AUX:RTIMe? query returns the Aux Out rise time setting.

Returned Format

```
<risetime><NL>

<risetime> ::= {FAST | SLOW}
```

See Also

- ":CALibrate:OUTPut" on page 425
- ":CALibrate:OUTPut:AUX" on page 427
- ":CALibrate:OUTPut:CAL" on page 429

History New in version 6.00.

:CALibrate:OUTPut:CAL

Command

```
:CALibrate:OUTPut:CAL {0 | OFF}
```

The :CALibrate:OUTPut:CAL command enables or disables signal output on the oscilloscope's CAL OUT connector.

Because there is no CAL OUT connector on MXR/EXR-Series oscilloscopes, OFF is the only valid value for this command.

Query

```
:CALibrate:OUTPut:CAL?
```

The :CALibrate:OUTPut:CAL? query returns the CAL OUT enable/disable setting.

Returned Format

```
<setting><NL>
```

```
<setting> ::= 0
```

See Also

- ":CALibrate:OUTPut" on page 425
- ":CALibrate:OUTPut:AUX" on page 427
- ":CALibrate:OUTPut:AUX:RTIMe" on page 428

History

New in version 6.00.

Version 11.00: Because there is no CAL OUT connector on MXR/EXR-Series oscilloscopes, OFF is the only valid value for this command.

:CALibrate:SKEW

Command

```
:CALibrate:SKEW <source>,<skew_value>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C>}
```

The :CALibrate:SKEW command sets the channel-to-channel skew factor for a channel. The numeric argument is a real number in seconds, which is added to the current time base position to shift the position of the channel's data in time. Use this command to compensate for differences in the electrical lengths of input paths due to cabling and probes.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<skew_value> A real number, in seconds.

Example This example sets the oscilloscope channel 1 skew to 1 µs.

```
myScope.WriteString ":CALibrate:SKEW CHANnel1,1E-6"
```

Query

```
:CALibrate:SKEW? <source>
```

The :CALibrate:SKEW? query returns the current skew value.

Returned Format

```
[:CALibrate:SKEW] <skew_value><NL>
```

History Legacy command (existed before version 3.10).

:CALibrate:STATus?

Query `:CALibrate:STATus?`

The :CALibrate:STATus? query returns the calibration status of the oscilloscope. These are 11 (or 15 on 8-channel oscilloscopes), comma-separated integers, with 1, 0, or -1:

- "1" = pass
- "0" = fail
- "-1" = unused

This matches the status in the Calibration dialog box in the Utilities menu.

Returned Format `[:CALibrate:STATus] <status>`

<status> On 4-channel oscilloscopes:

<Oscilloscope Frame Status>, <Channel1 Vertical>, <Channel1 Trigger>, <Channel2 Vertical>, <Channel2 Trigger>, <Channel3 Vertical>, <Channel3 Trigger>, <Channel4 Vertical>, <Channel4 Trigger>, <Aux Trigger>, <Timebase Calibration>

On 8-channel oscilloscopes:

<Oscilloscope Frame Status>, <Channel1 Vertical>, <Channel1 Trigger>, <Channel2 Vertical>, <Channel2 Trigger>, <Channel3 Vertical>, <Channel3 Trigger>, <Channel4 Vertical>, <Channel4 Trigger>, <Channel5 Vertical>, <Channel5 Trigger>, <Channel6 Vertical>, <Channel6 Trigger>, <Channel7 Vertical>, <Channel7 Trigger>, <Channel8 Vertical>, <Channel8 Trigger>, <Aux Trigger>, <Timebase Calibration>

History Legacy command (existed before version 3.10).

:CALibrate:TEMP?

Query `:CALibrate:TEMP?`

The :CALibrate:TEMP? query returns two delta temperature values in Celsius:

- Between the current temp and the temp of the last time scale calibration.
- Between the current temp and the temp of the last regular user calibration.

Returned Format `[:CALibrate:TEMP] <ts_delta_temp>,<user_delta_temp><NL>`

For example, the string returned could be "-1,0". A difference in the two delta values of one degree is not uncommon.

History Legacy command (existed before version 3.10).

18 :CHANnel<N> Commands

KEYSIGHT TECHNOLOGIES

The CHANnel subsystem commands control all vertical (Y axis) functions of the oscilloscope. You may toggle the channel displays on and off with the root level commands :VIEW and :BLANk, or with :CHANnel:DISPlay.

NOTE

In this section, you can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

:CHANnel<N>:ADC:CLIPped

Query `:CHANnel<N>:ADC:CLIPped?`

The :CHANnel<N>:ADC:CLIPped? query returns the channel's clipped status since the last time this command was issued or since the clipping status was cleared.

- 0 — Channel did not clip.
- 1 — Channel did clip low.
- 2 — Channel did clip high.
- 3 — Channel did clip both low and high.

The clipped status remains set until the it is read (by this query) or cleared.

Returned Format `<clipped_status><NL>`

`<clipped_status> ::= {0 | 1 | 2 | 3}`

See Also

- ":ACQuire:ADC:CLIPped:CLEar" on page 291
- ":DISPlay:CLIPped" on page 564

History New in version 10.10.

:CHANnel<N>:BWLimit

Command

```
:CHANnel<N>:BWLimit {{OFF | 0} | 20e6 | 200e6}
```

The :CHANnel<N>:BWLimit command controls the input channel's low-pass filter. When ON (or set to a value), the bandwidth of the specified channel is limited. The bandwidth filter can be used with either AC or DC coupling.

- On Infiniium MXR/EXR-Series oscilloscopes, 20 MHz or 200 MHz low pass filters can be turned on (without a passive probe being connected).

You can enable custom bandwidth limits using the :CHANnel<N>:ISIM:BWLimit command. You can specify the custom bandwidth limit value with the :CHANnel<N>:ISIM:BANDwidth command.

<N> An integer, 1-4.

Example This example sets the internal low-pass filter to "ON" for channel 1.

```
myScope.WriteString ":CHANnel1:BWLimit ON"
```

Query

```
:CHANnel<N>:BWLimit?
```

The :CHANnel<N>:BWLimit? query returns the current state of the low-pass filter for the specified channel.

Returned Format

```
[:CHANnel<N>:BWLimit] {0 | 20e6 | 200e6}<NL>
```

Example This example places the current setting of the low-pass filter in the variable varLimit, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:BWLimit?"
varLimit = myScope.ReadNumber
Debug.Print FormatNumber(varLimit, 0)
```

See Also

- ":CHANnel<N>:INPut" on page 450
- ":CHANnel<N>:ISIM:BWLimit" on page 456
- ":CHANnel<N>:ISIM:BANDwidth" on page 453
- ":ACQuire:BANDwidth" on page 296

History Legacy command (existed before version 3.10).

:CHANnel<N>:CLIPped?

Query `:CHANnel<N>:CLIPped?`

The oscilloscope's analog-to-digital converter (ADC) has clip detectors to detect and report ADC clipping. This alerts you to situations where digital-signal-processor-corrected signals appear to be scaled correctly on screen but are actually being clipped by the ADC.

The :CHANnel<N>:CLIPped? query returns the analog-to-digital converter (ADC) clipping status of the channel, where 0 = not clipped and 1 = clipped at least once.

The ADC clipping status for each channel is checked every 200 ms. If a channel has clipped any time during the 200 ms, the clipped (1) status is set. This clipped (1) status remains set until the clipped status is read (by this query) or reset. The clipped status is reset when the respective channel's vertical scale or offset is changed.

The :CHANnel<N>:CLIPped remote command works even when the "show analog-to-digital converter (ADC) clipping" option is disabled (with the ":DISPlay:CLIPped OFF" command).

Returned Format `<setting><NL>`

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

See Also

- ":DISPlay:CLIPped" on page 564
- ":CHANnel<N>:SCALe" on page 503
- ":CHANnel<N>:RANGe" on page 502
- ":CHANnel<N>:OFFSet" on page 468

History New in version 10.00.

:CHANnel<N>:COMMonmode

Command

```
:CHANnel<N>:COMMonmode {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:COMMonmode command turns on/off common mode for the channel. Channels 2 and 4 may form a common mode channel and Channels 1 and 3 may form a common mode channel.

<N> An integer, 1 to the number of analog input channels.

Example This example turns channel 1 common mode channel on (channel 1 + channel 3).

```
myScope.WriteString ":CHANnel1:COMMonmode ON"
```

Query

```
:CHANnel<N>:COMMonmode?
```

The :CHANnel<N>:COMMonmode? query returns whether the channel is in commonmode or not.

Returned Format

```
[:CHANnel<N>:COMMonmode] {1 | 0}<NL>
```

Example This example places the current common mode setting of the channel 1 display in the variable varComm, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:COMMonmode?"
varComm = myScope.ReadNumber
Debug.Print FormatNumber(varComm, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:DIFFerential

Command
```
:CHANnel<N>:DIFFerential {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:DIFFerential command turns on/off differential mode for the channel. Channels 1 and 3 may form a differential channel and Channels 2 and 4 may form a differential channel.

When differential channel modes are turned on, you can specify the display vertical scale, range, and offsets for the differential or common mode signals (using the :CHANnel<N>:DISPlay:SCALe, :CHANnel<N>:DISPlay:RANGe, or :CHANnel<N>:DISPlay:OFFSet commands), or you can set the display vertical scale, range, and offsets to track the acquisition vertical scale and offset (using the :CHANnel<N>:DISPlay:AUTO command).

<N> An integer, 1 to the number of analog input channels.

Example This example turns channel 1 differential on (channel 1 - channel 3).

```
myScope.WriteString ":CHANnel1:DIFFerential ON"
```

Query
```
:CHANnel<N>:DIFFerential?
```

The :CHANnel<N>:DIFFerential? query returns whether the channel is in differential mode or not.

Returned Format
```
[:CHANnel<N>:DIFFerential] {1 | 0}<NL>
```

Example This example places the current differential setting of the channel 1 display in the variable varDiff, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DIFFerential?"
varDiff = myScope.ReadNumber
Debug.Print FormatNumber(varDiff, 0)
```

See Also

- ":CHANnel<N>:DISPlay:AUTO" on page 443
- ":CHANnel<N>:DISPlay:OFFSet" on page 445
- ":CHANnel<N>:DISPlay:RANGe" on page 446
- ":CHANnel<N>:DISPlay:SCALe" on page 447

History Legacy command (existed before version 3.10).

:CHANnel<N>:DIFFerential:SKEW

Command `:CHANnel<N>:DIFFerential:SKEW <skew>`

The :CHANnel<N>:DIFFerential:SKEW <skew> command sets the skew that is applied to the differential or common mode pair of channels.

<skew> A real number for the skew value

Example This example sets the skew applied to the channel 1 - channel 3 differential channel to 10 µs.

```
myScope.WriteString ":CHANnel1:DIFFerential:SKEW 10E-6"
```

Query `:CHANnel<N>:DIFFerential:SKEW?`

The :CHANnel<N>:DIFFerential:SKEW? query returns the skew that is applied to the differential or common mode pair of channels.

Returned Format `[:CHANnel<N>:DIFFerential:SKEW] <skew_value><NL>`

Example This example places the current skew setting of the channel 1 - channel 3 differential channel in the variable varSkew, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DIFFerential:SKEW?"
varSkew = myScope.ReadNumber
Debug.Print FormatNumber(varSkew, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay

Command

```
:CHANnel<N>:DISPlay {{ON | 1} | {OFF | 0}}
```

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

NOTE

If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example sets channel 1 display to on.

```
myScope.WriteString ":CHANnel1:DISPlay ON"
```

Query

```
:CHANnel<N>:DISPlay?
```

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

Returned Format

```
[:CHANnel<N>:DISPlay] {1 | 0}<NL>
```

Example This example places the current setting of the channel 1 display in the variable varDisplay, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DISPlay?"
varDisplay = myScope.ReadNumber
Debug.Print FormatNumber(varDisplay, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay:AUTO

Command

```
:CHANnel<N>:DISPlay:AUTO {{ON | 1} | {OFF | 0}}
```

NOTE This command only works when differential channel modes are turned on (using the :CHANnel<N>:DIFFerential command).

When differential channel modes are turned on:

- ON — sets the differential and common mode display vertical scale and offset to track the acquisition vertical scale and offset.

  In this case, the differential and common mode display vertical scale and offsets are set using the normal :CHANnel<N>:SCALe, :CHANnel<N>:RANGe, or :CHANnel<N>:OFFSet commands.
- OFF — the differential and common mode display vertical scale and offset are set using the the :CHANnel<N>:DISPlay:SCALe, :CHANnel<N>:DISPlay:RANGe, or :CHANnel<N>:DISPlay:OFFSet commands.

<N> An integer, 1 to the number of analog input channels.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

Example This example sets the channel 1 - channel 3 differential channel display scale and offset to track the acquisition scale and offset.

```
myScope.WriteString ":CHANnel1:DISPlay:AUTO ON"
```

Query

```
:CHANnel<N>:DISPlay:AUTO?
```

The :CHANnel<N>:DISPlay:AUTO? query returns whether or not the differential or common mode display scale and offset are tracking the acquisition scale and offset.

Returned Format

```
[:CHANnel<N>:DISPlay:AUTO] {1 | 0}<NL>
```

Example This example places whether or not the channel 1 - channel 3 differential channel display scale and offset is tracking the acquisition scale and offset in the variable varAuto, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DISPlay:AUTO?"
varAuto = myScope.ReadNumber
Debug.Print FormatNumber(varAuto, 0)
```

See Also

- ":CHANnel<N>:DIFFerential" on page 440

- ":CHANNel<N>:DISPlay:OFFSet" on page 445
- ":CHANNel<N>:DISPlay:RANGe" on page 446
- ":CHANNel<N>:DISPlay:SCALe" on page 447
- ":CHANNel<N>:OFFSet" on page 468
- ":CHANNel<N>:RANGe" on page 502
- ":CHANNel<N>:SCALe" on page 503

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay:OFFSet

Command

```
:CHANnel<N>:DISPlay:OFFSet <value>
```

NOTE **This command only works when differential channel modes are turned on (using the :CHANnel<N>:DIFFerential command).**

When differential channel modes are turned on, the :CHANnel<N>:DISPlay:OFFSet command sets the display vertical offset of the selected channel.

If the differential and common mode display vertical scale and offset is set to track the acquisition vertical scale and offset (:CHANnel<N>:DISPlay:AUTO ON), using the :CHANnel<N>:DISPlay:OFFSet command turns OFF auto tracking, and the :CHANnel<N>:DISPlay:SCALe, :CHANnel<N>:DISPlay:RANGe, and :CHANnel<N>:DISPlay:OFFSet commands are used to specify the display vertical scale, range, and offsets for the differential or common mode signals.

<value> A real number for the value variable

NOTE **If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.**

Example This example sets the displayed offset of channel 1 to 10 mV.

```
myScope.WriteString ":CHANnel1:DISPlay:OFFSet 10e-3"
```

Query

```
:CHANnel<N>:DISPlay:OFFSet?
```

The :CHANnel<N>:DISPlay:OFFSet? query returns the displayed offset for the selected channel.

Returned Format

```
[:CHANnel<N>:DISPlay:OFFSet] <value><NL>
```

Example This example places the displayed offset of channel 1 in the variable varOffset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DISPlay:OFFSet?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, 0)
```

See Also

- ":CHANnel<N>:DIFFerential" on page 440
- ":CHANnel<N>:DISPlay:AUTO" on page 443
- ":CHANnel<N>:DISPlay:RANGe" on page 446
- ":CHANnel<N>:DISPlay:SCALe" on page 447

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay:RANGe

Command

```
:CHANnel<N>:DISPlay:RANGe <range>
```

NOTE This command only works when differential channel modes are turned on (using the :CHANnel<N>:DIFFerential command).

When differential channel modes are turned on, the :CHANnel<N>:DISPlay:RANGe command sets the display full scale vertical range of the selected channel.

If the differential and common mode display vertical scale and offset is set to track the acquisition vertical scale and offset (:CHANnel<N>:DISPlay:AUTO ON), using the :CHANnel<N>:DISPlay:RANGe command turns OFF auto tracking, and the :CHANnel<N>:DISPlay:SCALe, :CHANnel<N>:DISPlay:RANGe, and :CHANnel<N>:DISPlay:OFFSet commands are used to specify the display vertical scale, range, and offsets for the differential or common mode signals.

<range> A real number for the range value

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

Example This example sets the display range of channel 1 to 800 mV.

```
myScope.WriteString ":CHANnel1:DISPlay:RANGe 800e-3"
```

Query

```
:CHANnel<N>:DISPlay:RANGe?
```

The :CHANnel<N>:DISPlay:RANGe? query returns the full scale vertical range of the display for the selected channel.

Returned Format

```
[:CHANnel<N>:DISPlay:RANGe] <range><NL>
```

Example This example places the range of channel 1 in the variable varRange, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DISPlay:RANGe?"
varRange = myScope.ReadNumber
Debug.Print FormatNumber(varRange, 0)
```

See Also

- ":CHANnel<N>:DIFFerential" on page 440
- ":CHANnel<N>:DISPlay:AUTO" on page 443
- ":CHANnel<N>:DISPlay:OFFSet" on page 445
- ":CHANnel<N>:DISPlay:SCALe" on page 447

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay:SCALe

Command

```
:CHANnel<N>:DISPlay:SCALe <scale>
```

NOTE This command only works when differential channel modes are turned on (using the :CHANnel<N>:DIFFerential command).

When differential channel modes are turned on, the :CHANnel<N>:DISPlay:SCALe command sets the display vertical scale (units per division) of the selected channel.

If the differential and common mode display vertical scale and offset is set to track the acquisition vertical scale and offset (:CHANnel<N>:DISPlay:AUTO ON), using the :CHANnel<N>:DISPlay:SCALe command turns OFF auto tracking, and the :CHANnel<N>:DISPlay:SCALe, :CHANnel<N>:DISPlay:RANGe, and :CHANnel<N>:DISPlay:OFFSet commands are used to specify the display vertical scale, range, and offsets for the differential or common mode signals.

<scale> A real number for the scale value

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

Example This example sets the display scale of channel 1 to 100 mV per division.

```
myScope.WriteString ":CHANnel1:DISPlay:SCALe 100e-3"
```

Query

```
:CHANnel<N>:DISPlay:SCALe?
```

The :CHANnel<N>:DISPlay:SCALe? query returns the displayed scale of the selected channel per division.

Returned Format

```
[:CHANnel<N>:DISPlay:SCALe] <scale><NL>
```

Example This example places the display scale of channel 1 in the variable varScale, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:DISPlay:SCALe?"
varScale = myScope.ReadNumber
Debug.Print FormatNumber(varScale, 0)
```

See Also

- ":CHANnel<N>:DIFFerential" on page 440
- ":CHANnel<N>:DISPlay:AUTO" on page 443
- ":CHANnel<N>:DISPlay:OFFSet" on page 445
- ":CHANnel<N>:DISPlay:RANGe" on page 446

History Legacy command (existed before version 3.10).

:CHANnel<N>:DISPlay:TESTLIMITS?

Query

```
:CHANnel<N>:DISPlay:TESTLIMITS?
```

The :CHANnel<N>:DISPlay:TESTLIMITS? query returns whether a channel can be turned on or not.

Returned Format

```
<num_parms>,<<type>><valid_settings><NL>
```

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<enum>" for this query.

<valid_settings> The valid settings for the :CHANnel<N>:DISPlay command, vertical bar separated.

Example This example shows the Interactive IO session history for a couple channels.

```
-> :CHANnel4:DISPlay:TESTLIMITS?
<- 1,<enum>0|1
-> :CHANnel5:DISPlay:TESTLIMITS?
<- 1,<enum>0
```

In the query results, you can see that the channel 4 display can be 0 (OFF) or 1 (ON) and that channel 5 can be 0 (OFF) only.

See Also

- ":CHANnel<N>:DISPlay" on page 442
- ":ACQuire:BANDwidth:TESTLIMITS?" on page 298
- ":ACQuire:POINts:TESTLIMITS?" on page 314
- ":ACQuire:SRATe:TESTLIMITS?" on page 327

History New in version 10.00.

:CHANNel<N>:INPut

Command

```
:CHANNel<N>:INPut <parameter>
```

The :CHANNel<N>:INPut command selects the input coupling, impedance, and LF/HF reject for the specified channel.

<N> An integer, 1-4 in a single oscilloscope, 1-40 in a MultiScope system.

<parameter> On MXR/EXR-Series oscilloscopes:

- DC — DC coupling, 1 MΩ impedance.
- DC50 | DCFifty — DC coupling, 50Ω impedance.
- AC — AC coupling, 1 MΩ impedance.
- LFR1 | LFR2 — AC 1 MΩ input impedance.

  When no probe is attached, the coupling for each channel can be AC, DC, DC50, or DCFifty.

  If you have an 1153A probe attached, the valid parameters are DC, LFR1, and LFR2 (low-frequency reject).

Example This example sets the channel 1 input to DC50.

```
myScope.WriteString ":CHANnel1:INPut DC50"
```

Query

```
:CHANNel<N>:INPut?
```

The :CHANNel<N>:INPut? query returns the selected channel input parameter.

Returned Format

```
[CHANNel<N>:INPut] <parameter><NL>
```

Example This example puts the current input for channel 1 in the string variable, strInput. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:INPut?"
strInput = myScope.ReadString
Debug.Print strInput
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:INVert

Command

```
:CHANnel<N>:INVert {{0 | OFF} | {1 | ON}}
```

The :CHANnel<N>:INVert command enables or disables the invert setting for a channel.

Inverting a channel causes the waveform to be reflected about the 0 V reference point. Inverting a channel can cause the oscilloscope to stop triggering (depending on the trigger level setting) and it it can affect math function waveforms that have the channel as an input source.

Query

```
:CHANnel<N>:INVert?
```

The :CHANnel<N>:INVert? query returns the channel invert setting.

Returned Format

```
<setting><NL>
```

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

History New in version 5.70.

:CHANnel<N>:ISIM:APPLy

Command

```
:CHANnel<N>:ISIM:APPLy "<transfer_funct_file>"
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:APPLy command applies a pre-computed transfer function to the waveform. If InfiniiSim is in 2 port mode, the file must be a .tf2 file. If in 4 port mode, the file must be a .tf4 file. Use the ISIM:STATe command to enable InfiniiSim before issuing the APPLy command.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<transfer_funct_file> The full path to the .tf2 file name (if in 2 port mode) or the .tf4 file (if in 4 port mode).

Example This example applies the example.tf4 file to the waveform on channel 1.

```
myScope.WriteString _
 ":CHANnel1:ISIM:APPLy " + _
 """C:\Users\Public\Documents\Infiniium\Filters\example.tf4"""
```

Query

```
:CHANnel<N>:ISIM:APPLy?
```

The :CHANnel<N>:ISIM:APPLy? query returns the currently selected function file name when 2 port or 4 port mode is enabled.

Returned Format

```
[CHANnel<N>:ISIM:APPLy] <file_name><NL>
```

Example This example puts the current transfer function file name in the variable strFile. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:APPLy?"
strFile = myScope.ReadString
Debug.Print strFile
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:BANDwidth

Command

```
:CHANnel<N>:ISIM:BANDwidth <bw_value>
```

The :CHANnel<N>:ISIM:BANDwidth command lets you set the custom bandwidth limit (cutoff frequency) value. The :CHANnel<N>:ISIM:BWLimit command lets you enable or disable the custom bandwidth limit.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<bw_value> The maximum value is the sample rate / 2. The minimum value is 1000 Hz.

Example This example sets the channel 1 input bandwidth limit cutoff frequency to 2 GHz.

```
myScope.WriteString ":CHANnel1:ISIM:BANDwidth 2e9"
```

Query

```
:CHANnel<N>:ISIM:BANDwidth?
```

The :CHANnel<N>:ISIM:BANDwidth? query returns the selected channel input's bandwidth limit cutoff frequency.

Returned Format

```
[CHANnel<N>:ISIM:BANDwidth] <parameter><NL>
```

Example This example puts the current input for channel 1 in the string variable, varBwLimit. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:BANDwidth?"
varBwLimit = myScope.ReadNumber
Debug.Print FormatNumber(varBwLimit, 0)
```

See Also

- ":CHANnel<N>:ISIM:BWLimit" on page 456
- ":ACQuire:BANDwidth" on page 296

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:BPASs:CFRequency

Command

```
:CHANnel<N>:ISIM:BPASs:CFRequency <cf_value>
```

For the BANDpass, BP5 and BP10 bandwidth limit types (see :CHANnel<N>:ISIM:BWLimit:TYPE), the :CHANnel<N>:ISIM:BPASs:CFRequency command sets the center frequency of the bandpass filter.

For the BP5 and BP10 bandwidth limit types, only the center frequency set by the :CHANnel<N>:SPECtral:CFRequency command can be set.

<cf_value> Center frequency value in NR3 format.

Query

```
:CHANnel<N>:ISIM:BPASs:CFRequency?
```

For the BANDpass, BP5 and BP10 bandwidth limit types (see :CHANnel<N>:ISIM:BWLimit:TYPE), the :CHANnel<N>:ISIM:BPASs:CFRequency? query returns the center frequency setting.

Returned Format

```
<cf_value><NL>

<cf_value> ::= center frequency value in NR3 format.
```

See Also

- ":CHANnel<N>:ISIM:BWLimit:TYPE" on page 457
- ":CHANnel<N>:ISIM:BPASs:SPAN?" on page 455
- ":CHANnel<N>:SPECtral:CFRequency" on page 504

History New in version 10.10.

:CHANnel<N>:ISIM:BPASs:SPAN?

Query

```
:CHANnel<N>:ISIM:BPASs:SPAN?
```

For the BANDpass, BP5 and BP10 bandwidth limit types (see :CHANnel<N>:ISIM:BWLimit:TYPE), the :CHANnel<N>:ISIM:BPASs:SPAN? query returns the frequency span setting of the bandpass filter.

Returned Format

```
<span_value><NL>

<span_value> ::= span value in NR3 format.
```

See Also

- ":CHANnel<N>:ISIM:BWLimit:TYPE" on page 457
- ":CHANnel<N>:ISIM:BPASs:CFRequency" on page 454

History New in version 10.10.

:CHANnel<N>:ISIM:BWLimit

Command

```
:CHANnel<N>:ISIM:BWLimit {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:ISIM:BWLimit command lets you enable or disable the custom bandwidth limit (cutoff frequency). The CHANnel<N>:ISIM:BANDwidth command sets the value to be used when the custom bandwidth limit is enabled.

NOTE

If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example turns on the bandwidth limit feature for channel 1.

```
myScope.WriteString ":CHANnel1:ISIM:BWLimit ON"
```

Query

```
:CHANnel<N>:ISIM:BWLimit?
```

The :CHANnel<N>:ISIM:BWLimit? query returns the current state of the corresponding channel's bandwidth limiting feature.

Returned Format

```
[CHANnel<N>:ISIM:BWLimit] {1 | 0}<NL>
```

Example This example puts the current bandwidth limit state for channel 1 in the string variable, varLimit. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:BWLimit?"
varLimit = myScope.ReadNumber
Debug.Print FormatNumber(varLimit, 0)
```

See Also

- ":CHANnel<N>:ISIM:BWLimit:TYPE" on page 457
- ":CHANnel<N>:ISIM:BANDwidth" on page 453
- ":ACQuire:BANDwidth" on page 296

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:BWLimit:TYPE

Command

```
:CHANnel<N>:ISIM:BWLimit:TYPE <filter_type>

<filter_type> ::= {WALL | BESSEL4 | BANDpass | BP5 | BP10 | BUTTerworth}
```

The :CHANnel<N>:ISIM:BWLimit:TYPE command specifies a channel's bandwidth limit filter response:

- WALL — Specifies a Brick-Wall response for the bandwidth limit filter. This response has a sharp roll-off.
- BESSEL4 — Specifies a 4th order Bessel response for the bandwidth limit filter. This response has a more gradual roll-off.

NOTE To achieve the 4th order Bessel response, the maximum bandwidth you can specify is about 2/3 of the maximum bandwidth you could specify with the Brick-Wall filter. Also with the 4th Order Bessel filter, the brick-wall response takes over at what would be the brick-wall filter's maximum bandwidth. Therefore, you can have a combination of roll-off responses with this selection.

- BUTTerworth — Specifies a 4th order Butterworth response for the bandwidth limit filter. This response has a roll-off that is between the Brick Wall and Bessel responses.

NOTE To achieve the 4th order Butterworth response, the maximum bandwidth you can specify is about 80% of the maximum bandwidth you could specify with the Brick-Wall filter. Also with the Butterworth filter, the brick-wall response takes over at what would be the brick-wall filter's maximum bandwidth. Therefore, you can have a combination of roll-off responses with this selection.

- BANDpass — This option is used with the Phase Noise analysis application (see ":MEASure:PN:STATe" on page 1068). Because you do not have control over the slope of the transition from the pass band to the rejected bands, this option has limited usefulness as a general-purpose bandpass filter.

NOTE When the Frequency Extension option is turned on for a channel (by the ":ANALyze:SIGNal:TYPE CHANnel<N>,FEXTension" command), you will not be able to change the channel's bandwidth limit filter response. In this case, the channel's bandwidth limit is fixed to the bandpass filter set by the Frequency Extension.

Query

```
:CHANnel<N>:ISIM:BWLimit:TYPE?
```

The :CHANnel<N>:ISIM:BWLimit:TYPE? query returns the bandwidth limit filter response setting.

When the Frequency Extension option is turned on for a channel (by the ":ANALyze:SIGNal:TYPE CHANnel<N>,FEXTension" command), this query will return one of the following settings, depending on the option that is installed:

- BP5 — Returned when the Frequency Extension 5 GHz BW Span option is installed and enabled on the channel.
- BP10 — Returned when the Frequency Extension 10 GHz BW Span Upgrade option is installed and enabled on the channel.

Returned Format

```
<filter_type><NL>

<filter_type> ::= {WALL | BESSEL4 | BAND | BP5 | BP10 | BUTT}
```

See Also

- ":CHANnel<N>:ISIM:BWLimit" on page 456
- ":ANALyze:SIGNal:TYPE" on page 404
- ":CHANnel<N>:ISIM:BPASs:CFRequency" on page 454
- ":CHANnel<N>:ISIM:BPASs:SPAN?" on page 455

History

New in version 5.70.

Version 10.10: The BANDpass option has been added to support the Phase Noise analysis application, and the query can return BP5 or BP10 when the Frequency Extension option is turned on.

Version 11.10: The BUTTerworth option has been added.

:CHANnel<N>:ISIM:CONVolve

Command

```
:CHANnel<N>:ISIM:CONVolve "<s_parameter_file>", {OFF | ON}
```

NOTE This CHANnel command is available when you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:CONVolve command sets the InfiniiSim 2 Port state (:CHANnel<N>:ISIM:STATe PORT2), automates the creation of a transfer function, and applies the transfer function to the channel waveform. The transfer function is created using the **Add insertion loss of a fixture or cable** application preset's simple one-block circuit model. The S-parameter file defines the simulation circuit of the single block and the measurement circuit is a **Thru**. When the generated transfer function is applied, it performs an embed operation.

This command uses the S-parameter file's S_{21} insertion loss only. If a .s4p file is specified, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files.

Optionally, include ON to flip the port numbering when reading the S-parameter file.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<s_parameter_file> The quoted name of the S-parameter file.

Example This example convolves the S-parameter file example.s2p with the waveform on channel 1.

```
myScope.WriteString ":CHANnel1:ISIM:CONVolve example.s2p"
```

See Also

- ":CHANnel<N>:ISIM:STATe" on page 466
- ":CHANnel<N>:ISIM:APPLy" on page 452
- ":CHANnel<N>:ISIM:DEConvolve" on page 461

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:CORRection

Command

```
:CHANnel<N>:ISIM:CORRection <percent>
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:CORRection command sets the amount of linearly scaled correction applied to the non-DC frequency components of the measured signal. This lets you trade off the amount of correction to apply via the transformation function versus the increase in noise it may create at higher frequencies. In other words, you can fine-tune the amount of high-frequency noise versus the sharpness of the step response edge.

<N> An integer, 1 to the number of analog input channels.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<percent> If you are making averaged mode measurements or applying a transfer function that does not magnify the noise, use the full correction by setting this field to 100%.

However, if you are working with eye diagrams or making jitter measurements and the transfer function is magnifying the noise, you may want to limit the correction by selecting a lower percentage.

Example This example sets the channel 1 InfiniiSim correction factor to 80%.

```
myScope.WriteString ":CHANnel1:ISIM:CORRection 80"
```

Query

```
:CHANnel<N>:ISIM:CORRection?
```

The :CHANnel<N>:ISIM:CORRection? query returns the selected input channel's percent correction factor.

Returned Format

```
[CHANnel<N>:ISIM:CORRection] <percent><NL>
```

Example This example gets the current channel 1 InfiniiSim correction percentage and places it in the numeric variable, varIsimCorrection. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:CORRection?"
varIsimCorrection = myScope.ReadNumber
Debug.Print FormatNumber(varBwLimit, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:DEConvolve

Command

```
:CHANnel<N>:ISIM:DEConvolve "<s_parameter_file>", {OFF | ON}
```

NOTE This CHANnel command is available when you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:DEConvolve command sets the InfiniiSim 2 Port state (:CHANnel<N>:ISIM:STATe PORT2), automates the creation of a transfer function, and applies the transfer function to the channel waveform. The transfer function is created using the **Remove insertion loss of a fixture or cable** application preset's simple one-block circuit model. The S-parameter file defines the measurement circuit of the single block and the simulation circuit is a **Thru**. When the generated transfer function is applied, it performs an de-embed operation.

This command uses the S-parameter file's S_{21} insertion loss only. If a .s4p file is specified, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files.

Optionally, include ON to flip the port numbering when reading the S-parameter file.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<s_parameter_file> The quoted name of the S-parameter file.

Example This example deconvolves the S-parameter file example.s2p with the waveform on channel 1.

```
myScope.WriteString ":CHANnel1:ISIM:DEConvolve example.s2p"
```

See Also

- ":CHANnel<N>:ISIM:STATe" on page 466
- ":CHANnel<N>:ISIM:APPLy" on page 452
- ":CHANnel<N>:ISIM:CONVolve" on page 459

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:DELay

Command

```
:CHANnel<N>:ISIM:DElay {OFF | ON | TRIG}
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:DELay command specifies the transfer function filter delay option:

- ON — Includes filter delay.
- OFF — Removes filter delay.
- TRIG — Includes trigger-corrected delay.

Consult the InfiniiSim User's Guide in the Manuals section of the GUI help system for more information.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example applies the transfer function delay in the resultant waveform.

```
myScope.WriteString ":CHANnel1:ISIM:DELay ON"
```

Query

```
:CHANnel<N>:ISIM:DELay?
```

The :CHANnel<N>:ISIM:DELay? query returns the current state of the transfer function delay feature on the corresponding input channel.

Returned Format

```
[CHANnel<N>:ISIM:DELay] {OFF | ON | TRIG}<NL>
```

Example This example puts whether or not the transfer function delay is included in the resultant waveform for channel 1 in the string variable, strDelay. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:DELay?"
strDelay = myScope.ReadString
Debug.Print strDelay
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:NORMalize

Command

```
:CHANnel<N>:ISIM:NORMalize {{ON | 1} | {OFF | 0}}
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:NORMalize command activates or deactivates the "Normalize Gain" option. The InfiniiSim normalize gain option removes any DC gain of the transfer function and can be used when modeling probes.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example turns on the InfiniiSim normalize gain option for channel 1.

```
myScope.WriteString ":CHANnel1:ISIM:NORMalize ON"
```

Query

```
:CHANnel<N>:ISIM:NORMalize?
```

The :CHANnel<N>:ISIM:NORMalize? query returns the current state of the corresponding channel's InfiniiSim normalize gain option.

Returned Format

```
[CHANnel<N>:ISIM:NORMalize] {1 | 0}<NL>
```

Example This example puts the current InfiniiSim normalize gain state for channel 1 in the string variable, varNormalizeGain. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:NORMalize?"
varNormalizeGain = myScope.ReadNumber
Debug.Print FormatNumber(varLimit, 0)
```

See Also

- ":CHANnel<N>:ISIM:DELay" on page 462

History New in version 4.60.

:CHANnel<N>:ISIM:PEXTraction

Command

```
:CHANnel<N>:ISIM:PEXTraction {P12 | P32 | P34 | P14 | DIFFerential
                               | COMMonmode}
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:PEXTraction command selects the InfiniiSim port extraction. The selections are:

- P12 — Use ports 1 -> 2.
- P32 — Use ports 3 -> 2.
- P34 — Use ports 3 -> 4.
- P14 — Use ports 1 -> 4.
- DIFFerential — valid for all channels.
- COMMonmode — valid for all channels.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example selects the channel 1 InfiniiSim differential port extraction.

```
myScope.WriteString ":CHANnel1:ISIM:PEXTraction DIFFerential"
```

Query

```
:CHANnel<N>:ISIM:PEXTraction?
```

The :CHANnel<N>:ISIM:PEXTraction? query returns the current InfiniiSim port extraction selection.

Returned Format

```
[CHANnel<N>:ISIM:PEXTraction] {P12 | P32 | P34 | P14 | DIFF | COMM}<NL>
```

Example This example puts the current InfiniiSim port extraction selection for channel 1 in the string variable, strMode. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:PEXTraction?"
strMode = myScope.ReadString
Debug.Print strMode
```

History New in version 3.11.

:CHANnel<N>:ISIM:SPAN

Command
```
:CHANnel<N>:ISIM:SPAN <max_time_span>
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:SPAN command sets the maximum time span control in the InfiniiSim Setup dialog box.

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<max_time_span> A real number.

Example This example sets the maximum time span control to 100e-9.

```
myScope.WriteString ":CHANnel1:ISIM:SPAN 100e-9"
```

Query
```
:CHANnel<N>:ISIM:SPAN?
```

The :CHANnel<N>:ISIM:SPAN? query returns the current InfiniiSim filter maximum time span on the corresponding input channel.

Returned Format
```
[CHANnel<N>:ISIM:SPAN] <max_time_span><NL>
```

Example This example puts the InfiniiSim filter's maximum time span value in the variable varTspan. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:SPAN?"
varTspan = myScope.ReadNumber
Debug.Print FormatNumber(varTspan, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:ISIM:STATe

Command

```
:CHANnel<N>:ISIM:STATe {OFF | PORT2 | PORT4 | PORT41}
```

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:STATe command turns InfiniiSim on or off and sets whether 2 port, 4 port (Channels 1&3), or 4 port (Channel 1) mode is being used (if it is turned on).

NOTE If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

Example This example turns on InfiniiSim for channel 1 and puts it in 2 port mode.

```
myScope.WriteString ":CHANnel1:ISIM:STATe PORT2"
```

Query

```
:CHANnel<N>:ISIM:STATe?
```

The :CHANnel<N>:ISIM:STATe? query returns the current state of InfiniiSim on the corresponding input channel.

Returned Format

```
[CHANnel<N>:ISIM:STATe] {OFF | PORT2 | PORT4 | PORT41}<NL>
```

Example This example puts the current InfiniiSim state for channel 1 in the string variable, strMode. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:STATe?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also

- ":DISPlay:ISIM:DGRaphs" on page 575
- ":DISPlay:ISIM:GDCouple" on page 577

History Legacy command (existed before version 3.10).

:CHANnel<N>:LABel

Command

```
:CHANnel<N>:LABel <string>
```

The :CHANnel<N>:LABel command sets the channel label to the quoted string.

Labels can be enabled with the :DISPlay:LABel command.

NOTE

If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands), CHANnel<N> refers to a differential or common mode waveform as described in ":ACQuire:DIFFerential:PARTner" on page 302.

<N> An integer, 1 to the number of analog input channels.

<string> A series of 16 or less characters as a quoted ASCII string

Example This example sets the channel 1 label to Data.

```
myScope.WriteString ":CHANnel1:LABel ""Data"""
```

Query

```
:CHANnel<N>:LABel?
```

The :CHANnel<N>:LABel? query returns the label of the specified channel.

Returned Format

```
[CHANnel<N>:LABel] <string><NL>
```

See Also

- ":DISPlay:LABel" on page 584
- ":FUNCtion<F>:LABel" on page 659
- ":WMEMory<R>:LABel" on page 1772

History Legacy command (existed before version 3.10).

:CHANnel<N>:OFFSet

Command

```
:CHANnel<N>:OFFSet <offset_value>
```

The :CHANnel<N>:OFFSet command sets the vertical value that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

<N> An integer, 1 to the number of analog input channels.

<offset_value> A real number for the offset value at center screen. Usually expressed in volts, but it can also be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:UNITs command or the CHANnel<N>:PROBe:EXTernal:UNITs command.

Example This example sets the offset for channel 1 to 0.125 in the current measurement units:

```
myScope.WriteString ":CHANnel1:OFFSet 125E-3"
```

Query

```
:CHANnel<N>:OFFSet?
```

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

Returned Format

```
[CHANnel<N>:OFFSet] <offset_value><NL>
```

Example This example places the offset value of the specified channel in the variable, varOffset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:OFFSet?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, "Scientific")
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe

Command

```
:CHANnel<N>:PROBe <attenuation_factor>[,{RATio | DECibel}]
```

The :CHANnel<N>:PROBe command sets the probe attenuation factor and the units (ratio or decibels) for the probe attenuation factor for a user-defined probe.

The DECibel and RATio parameters also set the "mode" for the probe attenuation. These parameters, along with attenuation factor, determine the scaling of the display and affect automatic measurements and trigger levels.

This mode also determines the units (ratio or decibels) that may be used for a subsequent command.

<N> An integer, 1 to the number of analog input channels.

<attenuation_factor> A real number from 0.0001 to 1000 for the RATio attenuation units or from -80 dB to 60 dB for the DECibel attenuation units.

Example This example sets the probe attenuation factor for a 10:1 probe on channel 1 in ratio units.

```
myScope.WriteString ":CHANnel1:PROBe 10,RAT"
```

Query

```
:CHANnel<N>:PROBe?
```

The :CHANnel<N>:PROBe? query returns the current probe attenuation setting and units for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe] <attenuation>,{RATio | DECibel}<NL>
```

Example This example places the current attenuation setting for channel 1 in the string variable, strAtten, then the program prints the contents.

```
Dim strAtten As String   ' Dimension variable.
myScope.WriteString ":CHANnel1:PROBe?"
strAtten = myScope.ReadString
Debug.Print strAtten
```

If you use a string variable, the query returns the attenuation value and the factor (decibel or ratio). If you use an integer variable, the query returns the attenuation value. You must then read the attenuation units into a string variable.

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:ACCAL

Command

```
:CHANnel<N>:PROBe:ACCAL {AUTO | OFF | PRECprobe}
```

The :CHANnel<N>:PROBe:ACCAL command sets the type of AC response probe calibration to use:

- OFF — no AC response probe calibration is used.
- AUTO — the AC response probe calibration is based on the type of probe being used and its general characteristics. For example, these probes support automatic probe correction:
  - 1130/1/2/4B
  - 1168/9B
  - N2800/1/2/3A
  - N2830/1/2A
  - N7000/1/2/3A
  - N7010A
  - N7020A
  - N7024A
  - MX0011A/12A/13A/20A/21A/22A/24A/25A
  - MX0023A
- PRECprobe — PrecisionProbe or PrecisionCable probe calibration is used.

NOTE You are not able to start a PrecisionProbe or PrecisionCable calibration using remote SCPI commands. However, you can enter SCPI commands to use the results of calibrations performed using the front panel wizards.

<N> An integer, 1 to the number of analog input channels.

Example This example chooses the PrecisionProbe or PrecisionCable AC response calibration for the probe on channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:ACCAL PRECprobe"
```

Query

```
:CHANnel<N>:PROBe:ACCAL?
```

The :CHANnel<N>:PROBe:ACCAL? query returns the AC response probe calibration setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:ACCAL] {AUTO | OFF | PREC}<NL>
```

See Also

- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 494
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 492
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 495
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 491

History New in version 3.10.

:CHANnel<N>:PROBe:ATTenuation

Command

```
:CHANnel<N>:PROBe:ATTenuation {DIV1 | DIV10}
```

NOTE This command is only valid for the 1154A probe.

The :CHANnel<N>:PROBe:ATTenuation command sets the 1154A probe's input amplifier attenuation. If the 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the probe attenuation for channel 1 to divide by 10.

```
myScope.WriteString ":CHANnel1:PROBe:ATTenuation DIV10"
```

Query

```
:CHANnel<N>:PROBe:ATTenuation?
```

The :CHANnel<N>:PROBe:ATTenuation? query returns the current 1154A probe input amplifier attenuation setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:ATTenuation] {DIV1 | DIV10}<NL>
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:AUTozero

Command

```
:CHANnel<N>:PROBe:AUTozero
```

NOTE This command is currently only valid for the N2893A probe.

The :CHANnel<N>:PROBe:AUTozero command initiates the N2893A probe's auto degauss/ offset cal.

If the N2893A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 to the number of analog input channels.

Example This example performs an auto zero operation for the probe on channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:AUTozero"
```

History New in version 3.50.

:CHANnel<N>:PROBe:COUPling

Command

```
:CHANnel<N>:PROBe:COUPling {DC | AC}
```

The :CHANnel<N>:PROBe:COUPling command sets the probe coupling to either AC or DC.

NOTE This command is for probes only. To set the input channel coupling, see :CHANnel<N>:INPut.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the probe coupling for channel 1 to AC.

```
myScope.WriteString ":CHANnel1:PROBe:COUPling AC"
```

Query

```
:CHANnel<N>:PROBe:COUPling?
```

The :CHANnel<N>:PROBe:COUPling? query returns the current probe coupling setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:COUPling] {DC | AC}<NL>
```

See Also

- ":CHANnel<N>:INPut" on page 450

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:EADapter

Command

```
:CHANnel<N>:PROBe:EADapter {NONE | DIV10 | DIV20 | DIV100 | DIV1000
    | CUR0002VA | CUR001VA | CUR002VA | CUR01VA | CUR1VA | CUR10VA
    | CURN2893 | CUR1147 | HIVOLTN2790}
```

NOTE This command is valid only for the 1153A, 1154A, and 1159A probes and the E2697A and N5449A high impedance adapters.

The :CHANnel<N>:PROBe:EADapter command sets the probe external adapter control. The 1153A, 1154A, and 1159A probes and the E2697A and N5449A high impedance adapters have external adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the EADapter command to set the external adapter control to match the adapter connected to your probe as follows.

If an 1153A, 1154A, or 1159A probe or E2697A or N5449A high impedance adapter is not connected to the channel you will get a settings conflict error.

With the 1153A, 1154A, and 1159A probes:

Parameter	Description
NONE	Use this setting when there is no adapter connected to the end of your probe.
DIV10	Use this setting when you have a divide by 10 adapter connected to the end of your probe.
DIV20	Use this setting when you have a divide by 20 adapter connected to the end of your probe. (1159A)
DIV100	Use this setting when you have a divide by 100 adapter connected to the end of your probe. (1153A only)

With the E2697A and N5449A high impedance adapters: When the :CHANnel<N>:PROBe:EADapter command is used with either the E2697A or N5449A high impedance adapter, the behavior is the same as with the 115x probes; however, there are more parameters available to choose from. The following table describes which probes are available with which adapters and what the parameter string is:

Probe	Parameter	Compatability	
		E2697A adapter	N5449A adapter
1:1 probe	NONE	X	X
10:1 probe	DIV10	X	X
20:1 probe	DIV20	X	X
100:1 probe	DIV100	X	X

Probe	Parameter	Compatability	
		E2697A adapter	N5449A adapter
100:1 probe	DIV1000		X
0.002 V/A current probe	CUR0002VA	X	X
0.01 V/A current probe	CUR001VA	X	X
0.02 V/A current probe	CUR002VA	X	X
0.1 V/A current probe	CUR01VA	X	X
1 V/A current probe	CUR1VA	X	X
10 V/A current probe	CUR10VA	X	X
N2893A current probe	CURN2893		X
1147A/B current probe	CUR1147		X
N2790A high-voltage differential probe	HIVOLTN2790		X

Lastly, the N5449A adapter has the ability to automatically detect supported probes. If a supported probe (most passive probes) is attached to the N5449A adapter, the :CHANnel<N>:PROBe:EADapter command will not have any effect until the attached probe is removed. If, on the other hand, an unsupported probe (BNC cable, etc.) is attached to the N5449A adapter, the :CHANnel<N>:PROBe:EADapter command will work as normal. The :CHANnel<N>:PROBe:EADapter? query will work in either case.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the external adapter for channel 1 to divide by 10:

```
myScope.WriteString ":CHANnel1:PROBe:EADapter DIV10"
```

Query

```
:CHANnel<N>:PROBe:EADapter?
```

The :CHANnel<N>:PROBe:EADapter? query returns the current external adapter value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:EDApter] {NONE | DIV10 | DIV20 | DIV100 | DIV1000
   | CUR0002VA | CUR001VA | CUR002VA | CUR01VA | CUR1VA | CUR10VA
   | CURN2893 | CUR1147 | HIVOLTN2790}<NL>
```

Example This example places the external adapter value of the specified channel in the string variable, strAdapter, then prints the contents of the variable to the computer's screen.

```
Dim strAdapter As String   'Dimension variable
myScope.WriteString ":CHANnel1:PROBe:EADapter?"
strAdapter = myScope.ReadString
Debug.Print strAdapter
```

History Legacy command (existed before version 3.10).

Version 5.20: Added the DIV1000, CUR001VA, CUR01VA, CUR1VA, CUR10VA, CURN2893, CUR1147, HIVOLTN2790 options for use with the E2697A or N5449A high impedance adapters.

Version 6.10: Added the CUR0002VA and CUR002VA options for use with the E2697A or N5449A high impedance adapters.

:CHANnel<N>:PROBe:ECOupling

Command

```
:CHANnel<N>:PROBe:ECOupling {NONE | AC}
```

NOTE This command is valid only for the 1153A, 1154A, and 1159A probes.

The :CHANnel<N>:PROBe:ECOupling command sets the probe external coupling adapter control. The 1154A and 1159A probes have external coupling adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the ECOupling command to set the external coupling adapter control to match the adapter connected to your probe as follows.

Parameter	Description
NONE	Use this setting when there is no adapter connected to the end of your probe.
AC	Use this setting when you have an ac coupling adapter connected to the end of your probe.

If an 1153A, 1154A, or 1159A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the external coupling adapter for channel 1 to ac:

```
myScope.WriteString ":CHANnel1:PROBe:ECOupling AC"
```

Query

```
:CHANnel<N>:PROBe:ECOupling?
```

The :CHANnel<N>:PROBe:ECoupling? query returns the current external adapter coupling value for the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:ECOupling] {NONE | AC}<NL>
```

Example This example places the external coupling adapter value of the specified channel in the string variable, strAdapter, then prints the contents of the variable to the computer's screen.

```
Dim strAdapter As String   ' Dimension variable.
myScope.WriteString ":CHANnel1:PROBe:ECOupling?"
strAdapter = myScope.ReadString
Debug.Print strAdapter
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:EXTernal

Command

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

The :CHANnel<N>:PROBe:EXTernal command sets the external probe mode to on or off.

<N> An integer, 1 to the number of analog input channels.

Example This example sets channel 1 external probe mode to on.

```
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
```

Query

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

The :CHANnel<N>:PROBe:EXTernal? query returns the current external probe mode for the specified channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal] {1 | 0}<NL>
```

Example This example places the current setting of the external probe mode on channel 1 in the variable varMode, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:PROBe:EXTernal?"
varMode = myScope.ReadNumber
Debug.Print FormatNumber(varMode, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:EXTernal:GAIN

Command

```
:CHANnel<N>:PROBe:EXTernal:GAIN <gain_factor>[,{RATio | DECibel}]
```

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:GAIN command sets the probe external scaling gain factor and, optionally, the units for the probe gain factor. The reference factors that are used for scaling the display are changed with this command, and affect automatic measurements and trigger levels.

The RATio or DECibel also sets the mode for the probe attenuation and also determines the units that may be used for a subsequent command. For example, if you select RATio mode, then the attenuation factor must be given in ratio gain units. In DECibel mode, you can specify the units for the argument as "dB".

<N> An integer, 1 to the number of analog input channels.

<gain_factor> A real number from 0.0001 to 1000 for the RATio gain units, or from -80 dB to 60 dB for the DECibel gain units.

Example This example sets the probe external scaling gain factor for channel 1 to 10.

```
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:GAIN 10,RATio"
```

Query

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

The :CHANnel<N>:PROBe:EXTernal:GAIN? query returns the probe external gain setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal:GAIN] <gain_factor>,{RAT | DEC}<NL>
```

Example This example places the external gain value of the probe on the specified channel in the variable, varGain, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:GAIN?"
varGain = myScope.ReadString
Debug.Print varGain
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:EXTernal:OFFSet

Command

```
:CHANnel<N>:PROBe:EXTernal:OFFSet <offset_value>
```

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:OFFSet command sets the external vertical value for the probe that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

When using the 113xA series probes, the CHANnel<N>:PROBe:STYPe command determines how the offset is applied. When CHANnel<N>:PROBe:STYPe SINGle is selected, the :CHANnel<N>:PROBe:EXTernal:OFFset command changes the offset value of the probe amplifier. When CHANnel<N>:PROBe:STYPe DIFFerential is selected, the :CHANnel<N>:PROBe:EXTernal:OFFSet command changes the offset value of the channel amplifier.

<N> An integer, 1 to the number of analog input channels.

<offset_value> A real number for the offset value at center screen. Usually expressed in volts, but can be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:PROBe:EXTernal:UNITs command.

Example This example sets the external offset for the probe on channel 1 to 0.125 in the current measurement units:

```
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:OFFSet 125E-3"
```

Query

```
:CHANnel<N>:PROBe:EXTernal:OFFSet?
```

The :CHANnel<N>:PROBe:EXTernal:OFFSet? query returns the current external offset value for the probe on the specified channel.

Returned Format

```
[CHANnel<N>:PROBe:EXTernal:OFFSet] <offset_value><NL>
```

Example This example places the external offset value of the probe on the specified channel in the variable, Offset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:OFFSet?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:EXTernal:UNITs

Command

```
:CHANnel<N>:PROBe:EXTernal:UNITs {VOLT | AMPere | WATT | UNKNown}
```

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect. UNITs can also be set using the CHANnel<N>:UNITs command.

The :CHANnel<N>:PROBe:EXTernal:UNITs command sets the probe external vertical units on the specified channel. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel probe external commands and channel commands (such as :CHANnel<N>:PROBe:EXTernal:OFFSet and :CHANnel<N>:RANGe). See the Probe Setup dialog box for more information.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the external units for the probe on channel 1 to amperes.

```
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:UNITs AMPERE"
```

Query

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

The :CHANnel<N>:PROBe:EXTernal:UNITs? query returns the current external units setting for the probe on the specified channel.

Returned Format

```
[:CHANnel<N>:PROBe:EXTernal:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>
```

Example This example places the external vertical units for the probe on the specified channel in the string variable, strUnits, then prints the contents of the variable to the computer's screen.

```
Dim strUnits As String
myScope.WriteString ":CHANnel1:PROBe:EXTernal ON"
myScope.WriteString ":CHANnel1:PROBe:EXTernal:UNITs?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

History Legacy command (existed before version 3.10).

:CHANNel<N>:PROBe:GAIN

Command

```
:CHANNel<N>:PROBe:GAIN {X1 | X10}
```

NOTE This command is valid only for the 1154A probe.

The :CHANNel<N>:PROBe:GAIN command sets the 1154A probe input amplifier gain.

If an 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the probe gain for channel 1 to times 10.

```
myScope.WriteString ":CHANnel1:PROBe:GAIN X10"
```

Query

```
:CHANNel<N>:PROBe:GAIN?
```

The :CHANNel<N>:PROBe:GAIN? query returns the current probe gain setting for the selected channel.

Returned Format

```
[:CHANNel<N>:PROBe:GAIN] {X1 | X10}<NL>
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:HEAD:ADD

Command

```
:CHANnel<N>:PROBe:HEAD:ADD "head"[,"label"]
```

The :CHANnel<N>:PROBe:HEAD:ADD command adds an entry to the list of probe heads.

<N> An integer, 1 to the number of analog input channels.

"head" A quoted string matching the probe head model such as "N5381A", "E2678A", etc.

"label" An optional quoted string for the head label.

Example This example adds the probe head N5381A to the list of probe heads for channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:ADD ""N5381A"""
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:HEAD:DELete ALL

Command

```
:CHANnel<N>:PROBe:HEAD:DELete ALL
```

The :CHANnel<N>:PROBe:HEAD:DELete ALL command deletes all the nodes in the list of probe heads except for one default probe head which remains after this command is executed.

<N> An integer, 1 to the number of analog input channels.

Example This example deletes the entire list of probe heads for channel 1 except for the default head.

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:DELete ALL"
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:HEAD:SELect

Command

```
:CHANnel<N>:PROBe:HEAD:SELect {<int> | <quoted_label_string>}
```

The :CHANnel<N>:PROBe:HEAD:SELect command selects the probe head being used from a list of possible probe head choices. You can select by the position number in the list of probe heads, or you can select by the label given when the probe head was added.

<N> An integer, 1 to the number of analog input channels.

<int> Specifies the number of the head (or position) in the configure list. The entry at the top of the list starts at 1.

<quoted_label_string> Specifies the label of the probe head given with the :CHANnel<N>:PROBe:HEAD:ADD command.

Example This example add a couple of probe heads to the list then selects the probe head using a number and a label.

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:ADD 'N5445A:B1.5-2.5S'"
myScope.WriteString ":CHANnel1:PROBe:HEAD:ADD 'N5444A:2.92','foo'"
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect 1"
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect 'foo'"
```

Query

```
:CHANnel<N>:PROBe:HEAD:SELect? [{MODel | LABel}]
```

The :CHANnel<N>:PROBe:HEAD:SELect? query returns a SCPI formatted string of the selected probe head. Optional parameters are:

- MODel — Returns the model of the probe head.
- LABel — Returns the label of the probe head. This is the same label given with the :CHANnel<N>:PROBe:HEAD:ADD command and that can also be used with the SELect command.

If no parameter is specified, the MODel format is returned.

Example This example shows a few queries of the channel 1 probe head selection.

```
Dim strProbeHead As String
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect?"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "N5444A:2.92".
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect? LABel"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "foo".
myScope.WriteString ":CHANnel2:PROBe:HEAD:SELect? MODel"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "N5444A:2.92".
```

See Also

- ":CHANnel<N>:PROBe:HEAD:ADD" on page 484

History Legacy command (existed before version 3.10).

Version 3.50: Added the MPHY protocol type for the MIPI M-PHY serial decode selection.

:CHANnel<N>:PROBe:HEAD:VTERm

Command

```
:CHANnel<N>:PROBe:HEAD:VTERm {FLOating | EXTernal
   | {INTernal,<voltage>}}
```

The :CHANnel<N>:PROBe:HEAD:VTERm command sets the termination voltage for:

- The N5444A probe head.
- The N7010A active termination adapter, regardless of the attachment/head.
- The MX0020A/21A/22A/24A/25A probe amplifiers.

<N> An integer, 1 to the number of analog input channels.

<voltage> A real number for the internal termination voltage setting.

Example To set an internal termination voltage of -1.0 V:

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:VTERm INTernal,-1.0"
```

Query

```
:CHANnel<N>:PROBe:HEAD:VTERm?
```

The :CHANnel<N>:PROBe:HEAD:VTERm? query returns the termination voltage setting.

Returned Format

```
[:CHANnel<N>:PROBe:HEAD:VTERm] {FLO | EXT | {INT,<voltage>}}<NL>
```

History New in version 3.50.

Version 5.50: Supports the N7010A active termination adapter (regardless of the attachment/head) as well as the N5444A probe head.

Version 11.20: Supports the MX0020A/21A/22A/24A/25A probe amplifiers.

:CHANnel<N>:PROBe:ID?

Query `:CHANnel<N>:PROBe:ID?`

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

<N> An integer, 1 to the number of analog input channels.

Returned Format [:CHANnel<N>:PROBe:ID] <probe_id>

<probe_id> A string of alphanumeric characters. Some of the possible returned values are:

1131A	1132A	1134A
1152A	1154A	1156A
1157A	1158A	1159A
1163A	1168A	1169A
AutoProbe	E2621A	E2622A
E2695A	E2697A	N5380A
N5381A	N5382A	E2695A
No Probe	Unknown	User Defined Probe

Example This example reports the probe type connected to channel 1, if one is connected.

```
myScope.WriteString ":CHANnel1:PROBe:ID?"
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:INFO?

Query

```
:CHANnel<N>:PROBe:INFO?
```

The :CHANnel<N>:PROBe:INFO? query returns a comma-separated list of probe information.

Returned Format

```
[:CHANnel<N>:PROBe:INFO] <info_list><NL>
```

<info_list> A comma-delimited list of probe information that includes:

- Model number.
- Serial number.
- Probe head model number or "No Head".
- Attenuation calibration date and time (or "1 JAN 1999 00:00:00" if uncalibrated).
- Skew calibration date and time (or "1 JAN 1999 00:00:00" if uncalibrated).
- Specifies whether default attenuation "Default Atten" or calibrated attenuation "Cal Atten" is being used.
- Specifies whether default skew "Default Skew" or calibrated skew "Cal Skew" is being used.
- The first part of the attenuation ratio (<first>:<second>).
- The second part of the attenuation ratio (<first>:<second>).
- The probe bandwidth.

Example This is an example probe information string.

```
N7005A, US59410012, No Head, 1 JAN 1999 00:00:00, 1 JAN 1999 00:00:00,
Cal Atten, Default Skew, 1.1057E-02, 1.0000E+00, 6.0000E+10
```

See Also

- ":CHANnel<N>:PROBe:ID?" on page 488

History New in version 5.70.

Version 10.25: Added a field at the end of the query results for probe bandwidth.

:CHANnel<N>:PROBe:MODE

Command `:CHANnel<N>:PROBe:MODE {DIFF | SEA | SEB | CM | SEST}`

NOTE

This command is valid for InfiniiMode probes, for example:

- N2750/1/2A
- N2830/1/2A
- N7000/1/2/3A
- MX0011AA/12A/13A/20A/21A/22A/24A/25A

The :CHANnel<N>:PROBe:MODE command sets the probe's InfiniiMode configuration.

If the probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the probe InfiniiMode for channel 1 to common mode.

```
myScope.WriteString ":CHANnel1:PROBe:MODE CM"
```

Query `:CHANnel<N>:PROBe:MODE?`

The :CHANnel<N>:PROBe:MODE? query returns the probe's InfiniiMode setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:MODE] {DIFF | SEA | SEB | CM | SEST | NONE}<NL>`

NONE will be returned for unsupported probes.

History New in version 3.50.

Version 11.20: The SEST (source estimate differential) mode has been added for the MX0020A/21A/22A/24A/25A probe amplifiers.

:CHANnel<N>:PROBe:PRECprobe:BANDwidth

Command

```
:CHANnel<N>:PROBe:PRECprobe:BANDwidth {AUTO | {MANual, <bandwidth>}
                                        | {BOOSt, <boost_dB>}}
```

The :CHANnel<N>:PROBe:PRECprobe:BANDwidth command specifies how the limit of PrecisionProbe or PrecisionCable correction/boosting is determined.

<N> An integer, 1 to the number of analog input channels.

AUTO PrecisionProbe or PrecisionCable normally sets the bandwidth to a value that has a small amount of boosting in the frequency response.

MANual, <bandwidth> Let you manually specify a bandwidth limit at which to stop applying correction.

BOOSt, <boost_dB> Lets you specify a dB limit at which to stop applying correction.

Example This example specifies that, for PrecisionProbe or PrecisionCable on channel 1, correction/boosting should stop being applied at a 3 dB limit.

```
myScope.WriteString ":CHANnel1:PROBe:PRECprobe:BANDwidth BOOSt, 3"
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:BANDwidth?
```

The :CHANnel<N>:PROBe:PRECprobe:BANDwidth? query returns the current PrecisionProbe or PrecisionCable corrected bandwidth setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:BANDwidth] {AUTO | {MANual, <bandwidth>}
                                          | {BOOSt, <boost_dB>}}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 470
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 494
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 492
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 495
- ":CHANnel<N>:PROBe:PRECprobe:DELay" on page 493

History New in version 3.10.

:CHANnel<N>:PROBe:PRECprobe:CALibration

Command

```
:CHANnel<N>:PROBe:PRECprobe:CALibration <cal_string>[,<cal_string2>]
```

The :CHANnel<N>:PROBe:PRECprobe:CALibration command specifies the name of the PrecisionProbe or PrecisionCable calibration to use for the specified channel and probe.

<N> An integer, 1 to the number of analog input channels.

<cal_string>[,<cal_string2>] A quoted string that is the name of the PrecisionProbe or Precision Cable calibration. The SMA probe heads can use two independent calibration files.

Example This example says to use the PrecisionProbe or PrecisionCable calibration named "2-8-2" for channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:PRECprobe:CALibration "2-8-2""
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:CALibration?
```

The :CHANnel<N>:PROBe:PRECprobe:CALibration? query returns the currently specified name for the selected channel's PrecisionProbe or PrecisionCable calibration.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:CALibration] <cal_string>[,<cal_string2>]<N
L>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 470
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 494
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 495
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 491
- ":CHANnel<N>:PROBe:PRECprobe:DELay" on page 493

History New in version 3.10.

:CHANnel<N>:PROBe:PRECprobe:DELay

Command

```
:CHANnel<N>:PROBe:PRECprobe:DELay {{ON | 1} | {OFF | 0}}
```

The :CHANnel<N>:PROBe:PRECprobe:DELay command specifies whether to include cable delay in a PrecisionCable AC response probe calibration.

<N> An integer, 1 to the number of analog input channels.

Example This example specifies to include cable delay in the calibration.

```
myScope.WriteString ":CHANnel1:PROBe:PRECprobe:DELay ON"
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:DELay?
```

The :CHANnel<N>:PROBe:PRECprobe:DELay? query returns the current "include cable delay" selection.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:DELay] {1 | 0}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 470
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 494
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 492
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 495
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 491

History New in version 4.20.

:CHANnel<N>:PROBe:PRECprobe:MODE

Command

```
:CHANnel<N>:PROBe:PRECprobe:MODE {PROBe | CABLe}
```

The :CHANnel<N>:PROBe:PRECprobe:MODE command chooses between PrecisionProbe or PrecisionCable AC response probe calibration.

<N> An integer, 1 to the number of analog input channels.

Example This example chooses PrecisionProbe calibration for the probe on channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:PRECprobe:MODE PROBe"
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:MODE?
```

The :CHANnel<N>:PROBe:PRECprobe:MODE? query returns the current PrecisionProbe/PrecisionCable selection for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:MODE] {PROBe | CABLe}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 470
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 492
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 495
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 491
- ":CHANnel<N>:PROBe:PRECprobe:DELay" on page 493

History New in version 3.10.

:CHANnel<N>:PROBe:PRECprobe:ZSRC

Command

```
:CHANnel<N>:PROBe:PRECprobe:ZSRC {VIN | {VSRC, <impedance>}
                                  | {VSRC, <file_string>}}
```

The :CHANnel<N>:PROBe:PRECprobe:ZSRC command specifies how PrecisionProbe characterizes the time domain and frequency domain response.

<N> An integer, 1 to the number of analog input channels.

VIN Selects the VOut/Vin probe transfer function (which characterizes the output of the probe as a function of the input at the probe tips).

Defining the response this way lets you evaluate the probe's accuracy in reproducing the actual signal present in your system with the probe attached. This correction is what you would see with a real band limited probe that has finite input impedance. PrecisionProbe corrects the "VOut/Vin" response to be flat with frequency and phase to your defined bandwidth limit. It does not de-embed the loading effects of the probe. (Keysight's probe corrections are typically defined using Vout/Vin.)

VSRC, <impedance> Selects the VOut/VSrc estimate of probed system response (which corrects the probe as "what would be there if the probe were not present"), and specifies a constant (Zo/2) value (in ohms) as the system source impedance.

One drawback of defining the probe's response in this manner is that if the probe's loading causes your circuit to lose some timing or amplitude margin, you probably want to know that when you make a measurement. VOut/VSource compensation will hide these effects from you. However, this method can be effective if probing at the transmitter.

VSRC, <file_string> Selects the VOut/VSrc estimate of probed system response (which corrects the probe as "what would be there if the probe were not present"), and names an S-parameter file whose S11 is used to specify the system source impedance.

Example This example, for channel 1, tells PrecisionProbe to use the VOut/VSrc characterization and to get the system source impedance from S11 in the "foo.s2p" S-parameter file.

```
myScope.WriteString ":CHANnel1:PROBe:PRECprobe:ZSRC VSRC, "foo.s2p""
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:ZSRC?
```

The :CHANnel<N>:PROBe:PRECprobe:ZSRC? query returns the current settings for PrecisionProbe time domain and frequency domain response characterization.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:ZSRC] {VIN | {VSRC, <impedance>}
                                   | {VSRC, <file_string>}}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 470
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 494
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 492

- ":CHANNel<N>:PROBe:PRECprobe:BANDwidth" on page 491
- ":CHANNel<N>:PROBe:PRECprobe:DELay" on page 493

History New in version 3.10.

:CHANnel<N>:PROBe:PRIMary

Command

```
:CHANnel<N>:PROBe:PRIMary {ZIN | ZOUT}
```

For the N2820A/N2821A high-sensitivity current probes only, the :CHANnel<N>:PROBe:PRIMary command configures the input channel as a zoomed-in amplified channel (ZIN) or zoomed-out channel (ZOUT). With N2820A probes, the secondary channel will have the other waveform.

<N> An integer, 1 to the number of analog input channels.

Query

```
:CHANnel<N>:PROBe:PRIMary?
```

The :CHANnel<N>:PROBe:PRIMary? query returns the primary channel output setting.

Returned Format

```
[:CHANnel<N>:PROBe:PRIMary] {ZIN | ZOUT}<NL>
```

See Also

- ":MEASure:CHARge" on page 902

History New in version 5.60.

:CHANnel<N>:PROBe:RESPonsivity

Command `:CHANnel<N>:PROBe:RESPonsivity <value>`

NOTE This command is valid only for the N7004A Optical-to-Electrical Converter probe.

When a user-defined wavelength is selected (by using the ":CHANnel<N>:PROBe:WAVelength WUSer" command), the :CHANnel<N>:PROBe:RESPonsivity command sets the responsivity value that has been determined using an optical power meter.

You can set the responsivity only when the wavelength is set to WUSer.

<N> An integer, 1-4.

<value> Responsivity value in Volts/Watt.

Query `:CHANnel<N>:PROBe:RESPonsivity?`

The :CHANnel<N>:PROBe:RESPonsivity? query returns responsivity value setting.

Returned Format `<value><NL>`

```
<value> ::= V/W responsivity value in NR3 format
```

See Also • ":CHANnel<N>:PROBe:WAVelength" on page 501

History New in version 10.20.

:CHANnel<N>:PROBe:SKEW

Command

```
:CHANnel<N>:PROBe:SKEW <skew_value>
```

The :CHANnel<N>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. You can use the oscilloscope's probe skew control to remove timing differences between probes or cables on different channels.

<N> An integer, 1 to the number of analog input channels.

<skew_value> A real number for the skew value, in the range -1 ms to +1 ms.

Example This example sets the probe skew for channel 1 to 10 µs.

```
myScope.WriteString ":CHANnel1:PROBe:SKEW 10E-6"
```

Query

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

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

Returned Format

```
[:CHANnel<N>:PROBe:SKEW] <skew_value><NL>
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:STYPe

Command

```
:CHANnel<N>:PROBe:STYPe {DIFFerential | SINGle}
```

NOTE This command is valid only for the 113xA series probes, 1168A probe, and 1169A probe.

The :CHANnel<N>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA series probes, 1168A probe, and 1169A probe. This setting determines how offset is applied.

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

<N> An integer, 1 to the number of analog input channels.

Example This example sets the probe mode to single-ended.

```
myScope.WriteString ":CHANnel1:PROBe:STYPe SINGle"
```

Query

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

The :CHANnel<N>:PROBe:STYPe? query returns the current probe mode setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:STYPe] {DIFFerential | SINGle}<NL>
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:PROBe:WAVelength

Command

```
:CHANnel<N>:PROBe:WAVelength <wavelength>
```

NOTE This command is valid only for the N7004A Optical-to-Electrical Converter probe.

The :CHANnel<N>:PROBe:WAVelength command lets you specify the wavelength as 850 nm, 1310 nm, 1550 nm, or a user-defined value.

When WUSer is selected, use the :CHANnel<N>:PROBe:RESPonsivity command to enter the responsivity value determined by using an optical power meter.

<N> An integer, 1-4.

<wavelength> {W850 | W1310 | W1550 | WUSer}

Query

```
:CHANnel<N>:PROBe:WAVelength?
```

The :CHANnel<N>:PROBe:WAVelength? query returns the wavelength setting.

Returned Format

```
<wavelength><NL>

<wavelength> ::= {W850 | W1310 | W1550 | WUS}
```

See Also

- ":CHANnel<N>:PROBe:RESPonsivity" on page 498

History New in version 10.20.

:CHANnel<N>:RANGe

Command

```
:CHANnel<N>:RANGe <range_value>
```

The :CHANnel<N>:RANGe command defines the full-scale vertical axis of the selected channel. It sets up acquisition and display hardware to display the waveform at a given range scale. The values represent the full-scale deflection factor of the vertical axis in volts. These values change as the probe attenuation factor is changed.

<N> An integer, 1 to the number of analog input channels.

<range_value> A real number for the full-scale voltage of the specified channel number.

Example This example sets the full-scale range for channel 1 to 500 mV.

```
myScope.WriteString ":CHANnel1:RANGe 500E-3"
```

Query

```
:CHANnel<N>:RANGe?
```

The :CHANnel<N>:RANGe? query returns the current full-scale vertical axis setting for the selected channel.

Returned Format

```
[:CHANnel<N>:RANGe]<range_value><NL>
```

Example This example places the current range value in the number variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":CHANnel1:RANGe?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:SCALe

Command

```
:CHANnel<N>:SCALe <scale_value>
```

The :CHANnel<N>:SCALe command sets the vertical scale, or units per division, of the selected channel. This command is the same as the front-panel channel scale.

<N> An integer, 1 to the number of analog input channels.

<scale_value> A real number for the vertical scale of the channel in units per division.

Example This example sets the scale value for channel 1 to 500 mV/div.

```
myScope.WriteString ":CHANnel1:SCALe 500E-3"
```

Query

```
:CHANnel<N>:SCALe?
```

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

Returned Format

```
[:CHANnel<N>:SCALe] <scale_value><NL>
```

Example This example places the current scale value in the number variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":CHANnel1:SCALe?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:CHANnel<N>:SPECtral:CFRequency

Command

```
:CHANnel<N>:SPECtral:CFRequency <cf_value>
```

The :CHANnel<N>:SPECtral:CFRequency command sets the center frequency when:

- FEXTension is selected as the :ANALyze:SIGNal:TYPE (for the Frequency Extension feature).
- SPECtral is selected as the :ANALyze:SIGNal:TYPE (for the Spectral Analysis (DDC) feature).
- RTSA is selected as the :SYSTem:MODE (for the Real-Time Spectrum Analysis (RTSA) mode).

  RTSA is available in MXR-Series oscilloscopes only.

<cf_value> Center frequency value in NR3 format.

Query

```
:CHANnel<N>:SPECtral:CFRequency?
```

The :CHANnel<N>:SPECtral:CFRequency? query returns the center frequency setting.

Returned Format

```
<cf_value><NL>

<cf_value> ::= center frequency value in NR3 format.
```

See Also

- ":CHANnel<N>:SPECtral:SPAN (MXR-Series)" on page 506
- ":ANALyze:SIGNal:TYPE" on page 404
- ":SYSTem:MODE (MXR-Series)" on page 1534

History New in version 10.10.

Version 10.12: This command now also applies when SPECtral is selected as the signal type.

Version 11.05: This command now also applies when the RTSA mode is selected.

:CHANnel<N>:SPECtral:CFRequency:TESTLIMITS

Query `:CHANnel<N>:SPECtral:CFRequency:TESTLIMITS?`

The :CHANnel<N>:SPECtral:CFRequency:TESTLIMITS? query returns the center frequency minimum and maximum limits.

Returned Format `<num_parms>,<<type>><min>:<max><NL>`

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<numeric>" for this query.

<min> Lower center frequency limit value.

<max> Upper center frequency limit value.

See Also

- ":CHANnel<N>:SPECtral:CFRequency" on page 504
- ":CHANnel<N>:SPECtral:SPAN:TESTLIMITS" on page 507

History New in version 11.10.

:CHANnel<N>:SPECtral:SPAN (MXR-Series)

Command

```
:CHANnel<N>:SPECtral:SPAN <span_value>
```

The :CHANnel<N>:SPECtral:SPAN command sets the frequency span when:

- FEXTension is selected as the :ANALyze:SIGNal:TYPE (for the Frequency Extension feature).
- SPECtral is selected as the :ANALyze:SIGNal:TYPE (for the Spectral Analysis (DDC) feature).
- RTSA is selected as the :SYSTem:MODE (for the Real-Time Spectrum Analysis (RTSA) mode).

These selections are available in MXR-Series oscilloscopes only.

<span_value> The span value in NR3 format.

When FEXTension is selected as the signal type, depending on the Frequency Extension option that is installed, the span value can be 2 GHz (2E9) only.

Query

```
:CHANnel<N>:SPECtral:SPAN?
```

The :CHANnel<N>:SPECtral:SPAN? query returns the frequency span setting.

Returned Format

```
<span_value><NL>

<span_value> ::= span value in NR3 format.
```

See Also

- ":CHANnel<N>:SPECtral:CFRequency" on page 504
- ":ANALyze:SIGNal:TYPE" on page 404
- ":SYSTem:MODE (MXR-Series)" on page 1534

History New in version 10.10.

Version 10.12: This command now also applies when SPECtral is selected as the signal type.

Version 11.05: This command now also applies when the RTSA mode is selected.

:CHANnel<N>:SPECtral:SPAN:TESTLIMITS

Query `:CHANnel<N>:SPECtral:SPAN:TESTLIMITS?`

The :CHANnel<N>:SPECtral:SPAN:TESTLIMITS? query returns the frequency span minimum and maximum limits.

Returned Format `<num_parms>,<<type>><min>:<max><NL>`

<num_parms> Number of parameters, always 1 for this query.

<type> Type of values returned, always "<numeric>" for this query.

<min> Lower frequency span limit value.

<max> Upper frequency span limit value.

See Also

- ":CHANnel<N>:SPECtral:SPAN (MXR-Series)" on page 506
- ":CHANnel<N>:SPECtral:CFRequency:TESTLIMITS" on page 505

History New in version 11.10.

:CHANnel<N>:UNITs

Command
```
:CHANnel<N>:UNITs {VOLT | AMPere | WATT | UNKNown}
```

NOTE UNITs can also be set using the CHANnel<N>:PROBe:EXTernal:UNITs command when CHANnel<N>:PROBe:EXTernal command has been set to ON.

The :CHANnel<N>:UNITs command sets the vertical units. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel commands (such as :CHANnel<N>:RANGe and :CHANnel<N>:OFFSet). See the Probe Setup dialog box for more information.

<N> An integer, 1 to the number of analog input channels.

Example This example sets the units for channel 1 to amperes.

```
myScope.WriteString ":CHANnel1:UNITs AMPere"
```

Query
```
:CHANnel<N>:UNITs?
```

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

Returned Format
```
[:CHANnel<N>:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>
```

Example This example places the vertical units for the specified channel in the string variable, strUnits, then prints the contents of the variable to the computer's screen.

```
Dim strUnits As String
myScope.WriteString ":CHANnel1:UNITs?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

History Legacy command (existed before version 3.10).

19 :COUNter Commands

The :COUNter subsystem provides commands to control the counter feature.

:COUNter:RATio

Command `:COUNter:RATio {{0 | OFF} | {1 | ON}}`

When Counter A and Counter B are both on, the :COUNter:RATio command enables or disables the "show Counter A to Counter B ratio" feature.

When Counter A or Counter B are off, the only valid setting for the :COUNter:RATio command is {0 | OFF}.

Query `:COUNter:RATio?`

The :COUNter:RATio? query returns whether the "show Counter A to Counter B ratio" feature is enabled or disabled.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":COUNter:RATio:CURRent?" on page 511
- ":COUNter<N>:ENABle" on page 513

History New in version 11.00.

:COUNter:RATio:CURRent?

Query `:COUNter:RATio:CURRent?`

The :COUNter:RATio:CURRent? query returns the current Counter A to Counter B ratio value.

Return Format `<value><NL>`

`<value> ::= current counter value in NR3 format`

See Also

- ":COUNter:RATio" on page 510

History New in version 11.00.

:COUNter<N>:CURRent?

Query

```
:COUNter<N>:CURRent?
```

The :COUNter<N>:CURRent? query returns the current counter value.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

Return Format

```
<value><NL>

<value> ::= current counter value in NR3 format
```

See Also

- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:NDIGits" on page 515
- ":COUNter<N>:SOURce" on page 516

History New in version 11.00.

:COUNter<N>:ENABle

Command

```
:COUNter<N>:ENABle {{0 | OFF} | {1 | ON}}
```

The :COUNter<N>:ENABle command enables or disables the counter feature.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

Query

```
:COUNter<N>:ENABle?
```

The :COUNter<N>:ENABle? query returns whether the counter is enabled or disabled.

Return Format

```
<off_on><NL>

{0 | 1}
```

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:NDIGits" on page 515
- ":COUNter<N>:SOURce" on page 516

History New in version 11.00.

:COUNter<N>:MODE

Command

```
:COUNter<N>:MODE {FREQuency | PERiod | TOTalize}
```

The :COUNter<N>: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.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

Query

```
:COUNter<N>:MODE?
```

The :COUNter<N>:MODE? query returns the counter mode setting.

Return Format

```
<mode><NL>

<mode> ::= {FREQ | PER | TOT}
```

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:NDIGits" on page 515
- ":COUNter<N>:SOURce" on page 516
- ":COUNter<N>:TOTalize:CLEar" on page 517
- ":COUNter<N>:TOTalize:SLOPe" on page 518

History New in version 11.00.

:COUNter<N>:NDIGits

Command

```
:COUNter<N>:NDIGits <value>
```

```
<value> ::= 5 to 10 in NR1 format for Counter A/B, 5 to 8 for Counter C
```

When the counter mode is FREQuency or PERiod (see :COUNter<n>:MODE), the :COUNter<N>: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.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

Query

```
:COUNter<N>:NDIGits?
```

The :COUNter<N>:NDIGits? query returns the currently set number of digits of resolution.

Return Format

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

```
<value> ::= 5 to 10 in NR1 format for Counter A/B, 5 to 8 for Counter C
```

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:SOURce" on page 516
- ":COUNter<N>:TOTalize:CLEar" on page 517
- ":COUNter<N>:TOTalize:SLOPe" on page 518

History New in version 11.00.

:COUNter<N>:SOURce

Command

```
:COUNter<N>:SOURce {CHANnel<N>}
```

The :COUNter<N>:SOURce command selects the waveform source that the counter measures. You can select one of the analog input channels.

<N> COUNter<N> is an integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

CHANnel <N> is 1 to (# analog channels) in NR1 format.

Query

```
:COUNter<N>:SOURce?
```

The :COUNter<N>:SOURce? query returns the currently set counter source channel.

Return Format

```
<source><NL>
```

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:NDIGits" on page 515

History New in version 11.00.

:COUNter<N>:TOTalize:CLEar

Command

```
:COUNter<N>:TOTalize:CLEar
```

When the counter mode is TOTalize (see :COUNter<n>:MODE), the :COUNter<N>:TOTalize:CLEar command zeros the edge event counter.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:NDIGits" on page 515
- ":COUNter<N>:SOURce" on page 516
- ":COUNter<N>:TOTalize:SLOPe" on page 518

History New in version 11.00.

:COUNter<N>:TOTalize:SLOPe

Command

```
:COUNter<N>:TOTalize:SLOPe {{NEGative | FALLing} | {POSitive | RISing}}
```

When the counter mode is TOTalize (see :COUNter<n>:MODE), the :COUNter<N>:TOTalize:SLOPe command specifies whether positive or negative edges are counted.

<N> An integer, 1, 2, or 3 where:

- 1 = Counter A
- 2 = Counter B
- 3 = Counter C - Trigger Qualified

  The trigger qualified event counter is available when the trigger mode is not EDGE. It lets you 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 TRIG 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 25 ns; therefore, the maximum trigger qualified event frequency that can be counted is 40 MHz.

Query

```
:COUNter<N>:TOTalize:SLOPe?
```

The :COUNter<N>:TOTalize:SLOPe? query returns the currently set slope specification.

Return Format

```
<slope><NL>

<slope> ::= {FALL | RIS}
```

See Also

- ":COUNter<N>:CURRent?" on page 512
- ":COUNter<N>:ENABle" on page 513
- ":COUNter<N>:MODE" on page 514
- ":COUNter<N>:NDIGits" on page 515
- ":COUNter<N>:SOURce" on page 516
- ":COUNter<N>:TOTalize:CLEar" on page 517

History New in version 11.00.

20 :DIGital<N> Commands

NOTE The DIGital commands only apply to the MSO oscilloscopes.

:DIGital<N>:DISPlay

Command

NOTE **The DIGital commands only apply to the MSO oscilloscopes.**

```
:DIGital<N>[:DISPlay] {ON | OFF | 1 | 0}
```

The :DIGital<N>:DISPlay command enables or disables the view for the selected digital channel.

<N> An integer, 0-15.

Example This example turns on the display of bit 5 for the digital channels.

```
myScope.WriteString ":DIGital5:DISPlay ON"
```

Query

```
:DIGital<N>[:DISPlay]?
```

The :DIGital<N>:DISPlay? query returns the value of the display setting for the selected digital channel.

Returned Format

```
[:DIGital<N>:DISPlay] {1 | 0}<NL>
```

See Also

- ":DIGital<N>:LABel" on page 521
- ":DIGital<N>:POSition" on page 522
- ":DIGital<N>:SIZE" on page 523
- ":DIGital<N>:THReshold" on page 524

History Legacy command (existed before version 3.10).

:DIGital<N>:LABel

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
:DIGital<N>:LABel <quoted_string>
```

The :DIGital<N>:LABel command sets the digital channel label to the quoted string. Setting a label for a digital channel will also result in the name being added to the label list.

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

<N> An integer, 1-2.

<quoted_string> A series of 16 or less characters as a quoted ASCII string.

Example This example sets the label for bit 7 to Clock.

```
myScope.WriteString ":DIGital7:LABel ""Clock"""
```

Query

```
:DIGital<N>:LABel?
```

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

Returned Format

```
[:DIGital<N>:LABel] <quoted_string><NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 520
- ":DIGital<N>:POSition" on page 522
- ":DIGital<N>:SIZE" on page 523
- ":DIGital<N>:THReshold" on page 524

History Legacy command (existed before version 3.10).

:DIGital<N>:POSition

Command

```
:DIGital<N>:POSition <position>
```

The :DIGital<N>:POSition command sets the vertical position of a digital waveform within its waveform area grid.

<integer> Integer value from 0 to 24 in NR1 format.

Query

```
:DIGital<N>:POSition?
```

The :DIGital<N>:POSition? query returns the digital waveform's vertical position.

Returned Format

```
[:DIGital<N>:DISPlay] <integer><NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 520
- ":DIGital<N>:LABel" on page 521
- ":DIGital<N>:SIZE" on page 523
- ":DIGital<N>:THReshold" on page 524

History New in version 11.20.

:DIGital<N>:SIZE

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
DIGital<N>:SIZE {SMALl | MEDium | LARGe}
```

The :DIGital<N>:SIZE command changes the vertical size of all the displayed digital channels. The digital subsystem must be enabled before this command will work. See ENABle command in the root subsystem.

<N> An integer, 0-15.

Example This example changes the size to medium for all displayed digital channels or buses.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":DIGital5:SIZE MEDium"
```

Query

```
:DIGital<N>:SIZE?
```

The :DIGital:CHANnel:SIZE? query returns the size of the displayed digital channels.

Returned Format

```
[:DIGital<N>:SIZE] {SMALl | MEDium | LARGe}<NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 520
- ":DIGital<N>:LABel" on page 521
- ":DIGital<N>:POSition" on page 522
- ":DIGital<N>:THReshold" on page 524

History Legacy command (existed before version 3.10).

:DIGital<N>:THReshold

Command

```
:DIGital<N>:THReshold {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
    | DIFFerential | <value>}
```

The :DIGital<N>:THReshold command sets the logic threshold value for a pod. Setting the threshold for digital channels 0 through 7 sets the threshold for pod 1 while setting the threshold for digital channels 8 through 15 sets the threshold for pod 2. This command is equivalent to the POD<N>:THReshold command.

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:

- CMOS50 = 2.5 V
- CMOS33 = 1.65 V
- CMOS25 = 1.25 V
- ECL = -1.3 V
- PECL = 3.7 V
- TTL = 1.4 V
- DIFFerential = 0 V

<N> An integer, 0-15.

<value> A real number representing the voltage value which distinguishes a 1 logic level from a 0 logic level. Waveform voltages greater than the threshold are 1 logic levels while waveform voltages less than the threshold are 0 logic levels.

On MXR/EXR-Series oscilloscopes, the range of the threshold voltage is from -8 volts to 8 volts.

Example This example sets the threshold to 1.8 volts for bits D15 through D8.

```
myScope.WriteString ":DIGital8:THReshold 1.8"
```

Query

```
:DIGital<N>:THReshold?
```

The :DIGital<N>:THReshold? query returns the threshold value for the specified pod.

Returned Format

```
[:DIGital<N>:THReshold] {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
    | DIFF | <value>}<NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 520
- ":DIGital<N>:LABel" on page 521
- ":DIGital<N>:POSition" on page 522
- ":DIGital<N>:SIZE" on page 523

History Legacy command (existed before version 3.10).

Version 4.50: Added the DIFFerential parameter for specifying the threshold voltage.

21 :DISK Commands

The DISK subsystem commands perform the disk operations as defined in the File menu. This allows saving and loading of waveforms and setups, as well as saving screen images to bitmap files.

NOTE

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

NOTE

Filenames are Not Case Sensitive.

The filename that you use is not case sensitive.

:DISK:CDIRectory

Command `:DISK:CDIRectory "<directory>"`

The :DISK:CDIRectory command changes the present working directory to the designated directory name. An error occurs when the requested directory does not exist. You can then view the error with the :SYSTem:ERRor? [{NUMBer | STRing}] query.

<directory> A character-quoted ASCII string, which can include the subdirectory designation. You must separate the directory name and any subdirectories with a backslash (\).

Example This example sets the present working directory to C:\Users\Public\Documents\Infiniium.

```
myScope.WriteString ":DISK:CDIRectory ""C:\Users\Public\Documents\
Infiniium"""
```

History Legacy command (existed before version 3.10).

:DISK:COPY

Command

```
:DISK:COPY "<source_file>","<dest_file>"
```

The :DISK:COPY command copies a source file from the disk to a destination file on the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. Use full directory paths for the source and destination files.

<source_file> <dest_file> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example copies C:\Users\Public\Documents\Infiniium\File1.wfm to C:\Temp\File1b.wfm on the disk.

```
myScope.WriteString ":DISK:COPY ""C:\Users\Public\Documents\Infiniium\
File1.wfm"",""C:\Temp\File1b.wfm"""
```

History Legacy command (existed before version 3.10).

:DISK:DELete

Command

```
:DISK:DELete "<file_name>"
```

The :DISK:DELete command deletes a file from the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. The default path is C:\Users\Public\Documents\Infiniium.

<file_name> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example deletes FILE1.SET from the disk.

```
myScope.WriteString ":DISK:DELete ""FILE1.SET"""
```

History Legacy command (existed before version 3.10).

:DISK:DIRectory?

Query `:DISK:DIRectory? ["<directory>"]`

The :DISK:DIRectory? query returns the requested directory listing. Each entry is 63 bytes long, including a carriage return and line feed. The default path is C:\Users\Public\Documents\Infiniium.

<directory> The list of filenames and directories.

Returned Format `[:DISK:DIRectory]<n><NL><directory>`

<n> The specifier that is returned before the directory listing, indicating the number of lines in the listing.

<directory> The list of filenames and directories. Each line is separated by a <NL>.

Example This example displays a number, then displays a list of files and directories in the current directory. The number indicates the number of lines in the listing.

```
Dim varResults As Variant
Dim lngI As Long

myScope.WriteString ":DISK:DIR?"
varResults = myScope.ReadList(ASCIIType_BSTR, vbLf)
Debug.Print FormatNumber(varResults(0), 0)

For lngI = 1 To (varResults(0) - 2)
  Debug.Print CStr(varResults(lngI))
Next lngI
```

History Legacy command (existed before version 3.10).

:DISK:LOAD

Command

```
:DISK:LOAD "<file_name>"[,<destination>,<interp>]
```

The :DISK:LOAD command restores from the disk a setup file, composite file, or a waveform file into a waveform memory destination. The type of file is determined by the filename suffix if one is present, or by the destination field if one is not present. You can load .WFM, .CSV, .TSV, .TXT, .BIN, .H5, .SET, and .OSC file types. The destination is only used when loading a waveform memory.

CAUTION

Setups saved from Infiniium software versions prior to 2.00 may not load correctly in software versions 4.30 and greater.

You can remedy this by re-saving any pre-2.00 setups using any version of software from version 2.00 to version 4.20.

Setups saved from software versions between 2.00 and 4.20 should load correctly into version 4.30 and greater.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. You can use either .WFM, .CSV, .TSV, .TXT, .BIN, .H5, .SET, or .OSC as a suffix after the filename. If no file suffix is specified, the default is .wfm.

The present working directory is assumed, or you can specify the entire path. For example, you can load the standard setup file "SETUP0.SET" using the command:

:DISK:LOAD "C:\Users\Public\Documents\Infiniium\Setups\SETUP0.SET"

Or, you can use :DISK:CDIRectory to change the present working directory to C:\Users\Public\Documents\Infiniium\Setups, then just use the file name ("SETUP0.SET", for example). The default path is C:\Users\Public\Documents\Infiniium\Setups.

When setup files are loaded, touch screen settings are ignored.

<destination> WMEMory<R>.

Where <R> is an integer from 1-4.

If a destination is not specified, waveform memory 1 is used.

<interp> {OFF | INT1 | INT2 | INT4 | INT8 | INT16}

When loading waveform data into a waveform memory, you can specify the Sin(x)/x interpolation ratio that is used. OFF means no interpolation. You can also specify the 1, 2, 4, 8, or 16 point Sin(x)/x interpolation ratios using INT1, INT2, INT4, INT8, or INT16.

Example This example restores the waveform in FILE1.WFM to waveform memory 1 with no Sin(x)/x interpolation.

```
myScope.WriteString ":DISK:LOAD ""FILE1.WFM"",WMEM1,OFF"
```

History Legacy command (existed before version 3.10).

Version 6.20: The <interp> option has been added.

:DISK:MDIRectory

Command

```
:DISK:MDIRectory "<directory>"
```

The :DISK:MDIRectory command creates a directory in the present working directory which has been set by the :DISK:CDIRectory command. If the present working directory has not been set by the :DISK:CDIRectory command, you must specify the full path in the <directory> parameter as shown in Example 1 below.

An error is displayed if the requested subdirectory does not exist.

<directory> A quoted ASCII string which can include subdirectories. You must separate the directory name and any subdirectories with a backslash (\).

Example 1 This example creates the directory CPROGRAMS in the C:\Users\Public\Documents\Infiniium directory.

```
myScope.WriteString _
  ":DISK:MDIRectory ""C:\Users\Public\Documents\Infiniium\CPROGRAMS"""
```

Example 2 This example creates the directory CPROGRAMS in the present working directory set by the :DISK:CDIRectory command.

```
myScope.WriteString ":DISK:MDIRectory ""CPROGRAMS"""
```

You can check your path with the :DISK:DIRectory? query.

History Legacy command (existed before version 3.10).

:DISK:PUTFILE

Command

```
:DISK:PUTFILE "<file_name>",<binary_block>
```

The :DISK:PUTFILE command writes bytes to a full-path location on the oscilloscope.

<file_name> A quoted ASCII full path name of the file to be written.

<binary_block> A binary block of data, consisting of the bytes of the file to be written.

This is the same binary block format used when writing setup files to the oscilloscope. For examples of sending binary blocks to the oscilloscope, see the :SYSTem:SETup command examples.

Example This example shows how to write a transfer function file to the oscilloscope.

```
// Make a directory on the oscilloscope.
INF_USER_DATA_DIR = "C:\Users\Public\Documents\Infiniium\"
:DISK:MDIRectory INF_USER_DATA_DIR + "Apps\InfiniiSim\"

// Check to see if the above step was successful.
:DISK:CDIRectory INF_USER_DATA_DIR + "Apps\InfiniiSim\"

// Tell Infiniium to receive the <binary_block> bytes stream and
// write it to the "My.tf2" file.
:DISK:PUTFILE INF_USER_DATA_DIR + "Apps\InfiniiSim\
My.tf2",<binary_block>

// Tell Infiniium to load the transfer function from the newly
// written file.
:CHANnel1:ISIM:APPLy CHANnel1,INF_USER_DATA_DIR + "Apps\InfiniiSim\
My.tf2"
```

See Also

- ":DISK:MDIRectory" on page 534
- ":DISK:CDIRectory" on page 528
- ":SYSTem:SETup" on page 1538
- "Definite-Length Block Response Data" on page 123

History Version 11.30: This is not a new command but its description is now being included.

:DISK:PWD?

Query `:DISK:PWD?`

The :DISK:PWD? query returns the name of the present working directory (including the full path). If the default path (C:\Users\Public\Documents\Infiniium) has not been changed by the :DISK:CDIRectory command, the :DISK:PWD? query will return an empty string.

Returned Format `:DISK:PWD? <present_working_directory><NL>`

Example This example places the present working directory in the string variable strWdir, then prints the contents of the variable to the computer's screen.

```
Dim strWdir As String
myScope.WriteString ":DISK:PWD?"
str Wdir = myScope.ReadString
Debug.Print strWdir
```

History Legacy command (existed before version 3.10).

:DISK:SAVE:COMPosite

Command `:DISK:SAVE:COMPosite "<file_name>"`

The :DISK:SAVE:COMPosite command lets you save oscilloscope composite files to Infiniium's hard disk or to a network drive. Composite files contain setups and waveform data.

The file will have an .osc extension.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

The filename assumes the present working directory if a path does not precede the file name.

Example This example saves the oscilloscope's setup and waveform data to a composite file named "C:\Scope\Setup\Comp001.osc".

```
myScope.WriteString ":DISK:SAVE:COMPosite ""C:\Scope\Setup\Comp001"""
```

History New in version 3.50.

:DISK:SAVE:IMAGe

Command

```
:DISK:SAVE:IMAGe "<file_name>" [,<format>
[,{SCReen | GRATicule}
[,{ON | 1} | {OFF | 0}
[,{NORMal | INVert}
[,{ON | 1} | {OFF | 0}]]]]]
```

The DISK:SAVE:IMAGe command saves a screen image. The default path is C:\Users\Public\Documents\Infiniium.

<format> The image format can be: BMP, GIF, TIF, PNG, or JPEG. The extension is supplied by the oscilloscope depending on the selected file format.

If you do not include the format in the command, the file is saved in the format shown in the Save Screen dialog box.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

(First) ON | OFF ON means that compression is on for the bitmap format (BMP). OFF means compression is off.

(Second) ON | OFF The second ON/OFF selection indicates to save the setup information in the image or not.

<format> {BMP | GIF | TIF | JPEG | PNG}

NOTE Error 273 can occur after this command to warn you about a known issue when Remote Desktop is being used to control the oscilloscope. See "List of Error Messages" on page 2020.

Examples

```
myScope.WriteString ":DISK:SAVE:IMAGe ""FILE1"",BMP,SCR,ON,INVERT"
```

or:

```
myScope.WriteString ":DISK:SAVE:IMAGe ""FILE1"",TIF,GRAT,ON"
```

or:

```
myScope.WriteString ":DISK:SAVE:IMAGe ""FILE1"""
```

History Legacy command (existed before version 3.10).

:DISK:SAVE:JITTer

Command `:DISK:SAVE:JITTer "<file_name>"`

The DISK:SAVE:JITTer command saves the jitter measurements shown in the RJDJ tab at the bottom of the oscilloscope screen along with the RJDJ graph data in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Users\Public\Documents\Infiniium.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
myScope.WriteString ":DISK:SAVE:JITTer ""FILE1"""
```

History Legacy command (existed before version 3.10).

:DISK:SAVE:LISTing

Command `:DISK:SAVE:LISTing [<source>,] "<file_name>" [,<format>[,<type>]]`

The DISK:SAVE:LISTing command saves the contents of the bus listing window to a file in either a .csv or .txt format. The default path is C:\Users\Public\Documents\Infiniium.

<source> `{SERial<N>}` — The default serial bus is the one currently displayed in the listing window.

<N> An integer 1 - 4.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

<format> {CSV | TXT}

<type> {PACKets | SYMBols}

Specifies which display window to save.

Example

```
myScope.WriteString ":DISK:SAVE:LISTing SERial3, ""FILE1"", CSV"
```

History Legacy command (existed before version 3.10).

Version 5.00: Added the <type> parameter for specifying which display window to save.

:DISK:SAVE:MEASurements

Command

```
:DISK:SAVE:MEASurements "<file_name>"[,<legacy_save_mode>]
```

The DISK:SAVE:MEASurements command saves the measurements shown in the measurements tab at the bottom of the oscilloscope screen in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Users\Public\Documents\Infiniium.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

<legacy_save_mode> {{ON | 1} | {OFF | 0}}

The <legacy_save_mode> option specifies whether to save measurement results in the format used prior to Infiniium version 5.00. If this option is not used, it is OFF.

Example

```
myScope.WriteString ":DISK:SAVE:MEASurements ""FILE1"""
```

History Legacy command (existed before version 3.10).

Version 5.50: Added the <legacy_save_mode> option to save measurement results in the format used prior to Infiniium version 5.00.

:DISK:SAVE:MREPort

Command `:DISK:SAVE:MREPort "<filename>"`

The :DISK:SAVE:MREPort command lets you save information about the oscilloscope setup, a waveform screen capture, and measurement results to a PDF or MHTML (*.mht) format file.

MHTML is a web page archive format that includes, in one file, the HTML, images, and other resources used to display the page.

<filename> A quoted ASCII string file name with either a ".pdf" or ".mht" extension to specify the format.

The file name can be a full path name or just the file name, in which case the report is saved to the default Infiniium data directory.

If the ".pdf" or ".mht" extension is not specified, the command will return a SCPI file path error.

History New in version 6.60.

:DISK:SAVE:NOISe

Command `:DISK:SAVE:NOISe "<filename>"`

The :DISK:SAVE:NOISe command saves the noise measurements shown in the Noise Results tab at the bottom of the oscilloscope screen along with the Noise graph data in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Users\Public\Documents\Infiniium.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

See Also

- ":DISK:SAVE:JITTer" on page 539

History New in version 6.60.

:DISK:SAVE:PRECprobe

Command `:DISK:SAVE:PRECprobe "<file_name>.csv", {CHAN1 | CHAN2 | CHAN3 | CHAN4}`

The DISK:SAVE:PRECprobe command saves PrecisionProbe/Cable data in a comma separated variables (CSV) file format. The default path is C:\Users\Public\Documents\Infiniium.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
myScope.WriteString ":DISK:SAVE:PRECprobe ""PPch1data.csv""", CHAN1
```

History New in version 4.00.

:DISK:SAVE:SETup

Command

```
:DISK:SAVE:SETup "<file_name>"
```

The :DISK:SAVE:SETup command saves the current oscilloscope setup to a disk. The file will have a .set extension.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\Users\Public\Documents\Infiniium\setups.

Example This example saves the channel 1 waveform to SETUP1 on the disk.

```
myScope.WriteString ":DISK:SAVE:SETup ""SETUP1"""
```

History Legacy command (existed before version 3.10).

:DISK:SAVE:WAVeform

Command

```
:DISK:SAVE:WAVeform <source>,"<file_name>" [,<format>[,<header>]]
```

The :DISK:SAVE:WAVeform command saves a waveform to a disk. If the source is ALL, all of the currently displayed waveforms are saved to the file. If you use a file extension as shown below in the <format> variable, then the type of file saved defaults to the extension type. If no format is specified and no extension is used, the file is saved in the INTernal format.

NOTE See the ":WAVeform:VIEW" on page 1715 command to determine how much data is saved.

NOTE When an acquisition is made on multiple channels, the data for each channel has the same X origin and the same number of points.

<source> {ALL | CHANnel<N> | CLOCk | FUNCtion<F> | HISTogram | MTRend | MSPectrum | EQUalized<L> | WMEMory<R> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR | BUS<B> | PODALL | POD1 | POD2}

The BUS<B> sources can be used with the BIN, CSV, TSV, and TXT formats.

For more information on waveform sources, see Chapter 50, "Waveform Sources," starting on page 2011.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\Users\Public\Documents\Infiniium.

<format> {BIN | CSV | INTernal | TSV | TXT | H5 | H5INt | MATlab}

The following file name extensions are used for the different formats:

- BIN = file_name.bin
- CSV (comma separated values) = file_name.csv
- INTernal = file_name.wfm
- TSV (tab separated values) = file_name.tsv
- TXT = file_name.txt
- H5 (HDF5) = file_name.h5

  In the H5 format, data is saved as floats. In this case, the data values are actual vertical values and do not need to be multiplied by the Y increment value.
- H5INt (HDF5) = file_name.h5

In the H5INt format, data is saved as integers. In this case, data values are quantization values and need to be multiplied by the Y increment value and added to the Y origin value to get the actual vertical values.

- MATlab (MATLAB data format) = file_name.mat

<header> {{ON | 1} | {OFF | 0}}

Example This example saves the channel 1 waveform to FILE1 on the disk in the CSV format with header on.

```
myScope.WriteString ":DISK:SAVE:WAVeform CHANnel1,""FILE1"",CSV,ON"
```

History Legacy command (existed before version 3.10).

Version 4.50: Added the H5INt format parameter which saves waveform data as integers within the H5 file.

Version 6.10: Added the MATlab format for saving waveforms to MATLAB (.mat) data format files.

:DISK:SEGMented

Command

```
:DISK:SEGMented {ALL | CURRent}
```

The :DISK:SEGMented command sets whether all segments or just the current segment are saved to a file when the :DISK:SAVE:WAVeform command is issued and the source is a channel but not a waveform memory or function. Before segments can be saved, the :ACQuire:MODE must be set to the SEGMented mode and segments must be acquired.

Example This example sets the disk segmented memory store method to CURRent.

```
myScope.WriteString ":DISK:SEGMented CURRent"
```

Query

```
:DISK:SEGMented?
```

The :DISK:SEGMented? query returns disk segmented memory store method value.

Returned Format

```
[:DISK:SEGMented] {ALL | CURRent}<NL>
```

Example This example places the disk store method in the string variable strMethod, then prints the contents of the variable to the computer's screen.

```
Dim strMethod As String
myScope.WriteString ":DISK:SEGMented?"
strMethod = myScope.ReadString
Debug.Print strMethod
```

History Legacy command (existed before version 3.10).

22 :DISPlay Commands

KEYSIGHT TECHNOLOGIES

The DISPlay subsystem controls the display of data, text, and graticules, and the use of color.

:DISPlay:BOOKmark:DELete:ALL

Command `:DISPlay:BOOKmark:DELete:ALL`

The :DISPlay:BOOKmark:DELete:ALL command deletes all bookmarks.

See Also

- ":DISPlay:BOOKmark<N>:DELete" on page 552
- ":DISPlay:BOOKmark<N>:SET" on page 553
- ":DISPlay:BOOKmark<N>:VERTical?" on page 554
- ":DISPlay:BOOKmark<N>:XPOSition" on page 555
- ":DISPlay:BOOKmark<N>:YPOSition" on page 556

History New in version 11.20.

:DISPlay:BOOKmark<N>:DELete

Command

```
:DISPlay:BOOKmark<N>:DELete
```

The :DISPlay:BOOKmark<N>:DELete command deletes a bookmark.

<N> An integer, 1-100.

See Also

- ":DISPlay:BOOKmark:DELete:ALL" on page 551
- ":DISPlay:BOOKmark<N>:SET" on page 553
- ":DISPlay:BOOKmark<N>:VERTical?" on page 554
- ":DISPlay:BOOKmark<N>:XPOSition" on page 555
- ":DISPlay:BOOKmark<N>:YPOSition" on page 556

History New in version 5.00.

:DISPlay:BOOKmark<N>:SET

Command

```
:DISPlay:BOOKmark<N>:SET NONE,"label"[,<color>[,"content"]]
```

```
:DISPlay:BOOKmark<N>:SET <source>,"label"[,"content"[,<time>]]
```

The :DISPlay:BOOKmark<N>:SET command sets a bookmark.

<N> An integer, 1-100.

"label" A quoted ASCII string. This is the text that appears in the bookmark callout box.

<color> Display element color name (see the color names in ":DISPlay:SCOLor" on page 598). You can set the color only for bookmarks that are not associated with a waveform (that is, when <source> is NONE).

"content" A quoted ASCII string. This is the text that pops up when you mouse over a bookmark callout box.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR | DIGital<M>}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

<time> A real number time position. Time values are appropriate only for bookmarks associated with waveforms.

See Also

- ":DISPlay:BOOKmark:DELete:ALL" on page 551
- ":DISPlay:BOOKmark<N>:DELete" on page 552
- ":DISPlay:BOOKmark<N>:VERTical?" on page 554
- ":DISPlay:BOOKmark<N>:XPOSition" on page 555
- ":DISPlay:BOOKmark<N>:YPOSition" on page 556

History New in version 5.00.

:DISPlay:BOOKmark<N>:VERTical?

Query `:DISPlay:BOOKmark<N>:VERTical?`

The :DISPlay:BOOKmark<N>:VERTical? query returns a waveform's vertical value at a bookmark's horizontal position.

<N> An integer, 1-100.

Returned Format `<vertical_value><NL>`

<vertical_value> A real number value.

See Also

- ":DISPlay:BOOKmark:DELete:ALL" on page 551
- ":DISPlay:BOOKmark<N>:DELete" on page 552
- ":DISPlay:BOOKmark<N>:SET" on page 553
- ":DISPlay:BOOKmark<N>:XPOSition" on page 555
- ":DISPlay:BOOKmark<N>:YPOSition" on page 556

History New in version 5.00.

:DISPlay:BOOKmark<N>:XPOSition

Command `:DISPlay:BOOKmark<N>:XPOSition <x_pos>`

The :DISPlay:BOOKmark<N>:XPOSition command sets the horizontal grid position of a bookmark's callout box.

<N> An integer, 1-100.

<x_pos> A real number between 0.0 and 1.0 that represents a percentage of the grid width.

Query `:DISPlay:BOOKmark<N>:XPOSition?`

The :DISPlay:BOOKmark<N>:XPOSition? query returns the horizontal grid position of a bookmark's callout box.

Returned Format `[:DISPlay:BOOKmark<N>:XPOSition] <x_pos><NL>`

See Also

- ":DISPlay:BOOKmark:DELete:ALL" on page 551
- ":DISPlay:BOOKmark<N>:DELete" on page 552
- ":DISPlay:BOOKmark<N>:SET" on page 553
- ":DISPlay:BOOKmark<N>:VERTical?" on page 554
- ":DISPlay:BOOKmark<N>:YPOSition" on page 556

History New in version 5.00.

:DISPlay:BOOKmark<N>:YPOSition

Command

```
:DISPlay:BOOKmark<N>:YPOSition <y_pos>
```

The :DISPlay:BOOKmark<N>:YPOSition command sets the vertical grid position of a bookmark's callout box.

<N> An integer, 1-100.

<y_pos> A real number between 0.0 and 1.0 that represents a percentage of the grid height.

Query

```
:DISPlay:BOOKmark<N>:YPOSition?
```

The :DISPlay:BOOKmark<N>:YPOSition? query returns the vertical grid position of a bookmark's callout box.

Returned Format

```
[:DISPlay:BOOKmark<N>:YPOSition] <y_pos><NL>
```

See Also

- ":DISPlay:BOOKmark:DELete:ALL" on page 551
- ":DISPlay:BOOKmark<N>:DELete" on page 552
- ":DISPlay:BOOKmark<N>:SET" on page 553
- ":DISPlay:BOOKmark<N>:VERTical?" on page 554
- ":DISPlay:BOOKmark<N>:XPOSition" on page 555

History New in version 5.00.

:DISPlay:CGRade

Command

```
:DISPlay:CGRade {{ON | 1} | {OFF | 0}}[,<source>]
```

The :DISPlay:CGRade command sets the color grade persistence on or off.

When in the color grade persistence mode, all waveforms are mapped into a database and shown with different colors representing varying number of hits in a pixel. "Connected dots" display mode (:DISPlay:CONNect) is disabled when the color grade persistence is on.

The oscilloscope has three features that use a specific database. This database uses a different memory area than the waveform record for each channel. The three features that use the database are:

- Histograms.
- Mask testing.
- Color grade persistence.

When any one of these three features is turned on, the oscilloscope starts building the database. The database is the size of the graticule area and varies in size. Behind each pixel is a 53-bit counter. Each counter is incremented each time a pixel is hit by data from a channel or function. The maximum count (saturation) for each counter is 9,007,199,254,740,991. You can check for counter saturation by using the DISPlay:CGRade:LEVels? query.

The color grade persistence uses colors to represent the number of hits on various areas of the display. The default color-grade state is off.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted:

- Color grade is enabled/disabled for all sources which are currently on.
- When enabling color grade, the main waveform view is turned off.
- When disabling color grade, the main waveform view is turned on.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example

This example sets the color grade persistence on.

```
myScope.WriteString ":DISPlay:CGRade ON"
```

Query

```
:DISPlay:CGRade? [<source>]
```

The DISPlay:CGRade query returns the current color-grade state.

If <source> is omitted, the query returns ON (1) if any color grade is enabled.

Returned Format

```
[:DISPlay:CGRade] {1 | 0}<NL>
```

Example

This example returns the current color grade state.

```
Dim strCgrade As String   ' Dimension variable.
myScope.WriteString ":DISPlay:CGRade?"
strCgrade = myScope.ReadString
Debug.Print strCgrade
```

See Also

- ":DISPlay:CGRade:LEVels?" on page 560
- ":DISPlay:CGRade:SCHeme" on page 562

History

Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which color grade should be turned on or off.

Version 5.50: When the <source> parameter is not provided, enabling color grade will turn off the main waveform view, and disabling color grade will turn on the main waveform view.

:DISPlay:CGRade:LEGend

Command `:DISPlay:CGRade:LEGend {{0 | OFF} | {1 | ON}}`

The :DISPlay:CGRade:LEGend command lets you enable or disable the Color Grade legend in the graphical user interface's Results pane.

Query `:DISPlay:CGRade:LEGend?`

The :DISPlay:CGRade:LEGend? query returns the Color Grade legend setting.

Returned Format `<setting><NL>`

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

See Also

- ":DISPlay:CGRade" on page 557

History New in version 6.60.

:DISPlay:CGRade:LEVels?

Query `:DISPlay:CGRade:LEVels?`

The :DISPlay:CGRade:LEVels? query returns the range of hits represented by each color. Fourteen values are returned, representing the minimum and maximum count for each of seven colors. In the CLASsic color grade scheme, the values are returned in the following order:

- Green minimum value
- Green maximum value
- Blue minimum value
- Blue maximum value
- Pink minimum value
- Pink maximum value
- Red minimum value
- Red maximum value
- Orange minimum value
- Orange maximum value
- Yellow minimum value
- Yellow maximum value
- White minimum value
- White maximum value

Returned Format `[DISPlay:CGRade:LEVels] <color format><NL>`

<color format> <intensity color min/max> is an integer value from 0 to 9,007,199,254,740,991

Example This example gets the range of hits represented by each color and prints it on the computer screen:

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:CGRade:LEVels?"
strCgrade = myScope.ReadString
Debug.Print strCgrade
```

In the CLASsic color grade scheme, colors start at green minimum, maximum, then blue, pink, red, orange, yellow, white. The format is a string where commas separate minimum and maximum values. The largest number in the string can be 9,007,199,254,740,991

An example of a possible returned string is as follows:

1,414,415,829,830,1658,1659,3316,3317,6633,6634,13267,13268,26535

See Also

- ":DISPlay:CGRade" on page 557
- ":DISPlay:CGRade:SCHeme" on page 562

History Legacy command (existed before version 3.10).

Version 5.00: This query is unchanged but results are returned only when a single color grade view is on.

:DISPlay:CGRade:SCHeme

Command

```
:DISPlay:CGRade:SCHeme {CLASsic | TEMP}
```

The :DISPlay:CGRade:SCHeme command sets the color grade scheme to CLASsic or TEMP.

Color grade persistence is displayed in 255 colors grouped into seven color range blocks. The blocks represent the database counts for each color range. In the CLASsic color grade scheme, the counters with the largest counts are displayed using a white pixel while the counters with the smallest counts are displayed using green pixels.

The following table shows the counter range blocks for each color for both the CLASsic and TEMP color grade schemes.

Color Grade Scheme		Range
Classic	Temperature	
		50% to 100% of Max counter
		25% to 50% of Max counter
		12.5% to 25% of Max counter
		6.25% to 12.5% of Max counter
		3.125% to 6.25% of Max counter
		1.5625% to 3.125% of Max counter
		1 hit to 1.5625% of Max counter

Example This example sets the color grade scheme to "classic".

```
myScope.WriteString ":DISPlay:CGRade:SCHeme CLASsic"
```

Query

```
:DISPlay:CGRade:SCHeme?
```

The :DISPlay:CGRade:SCHeme? query returns the specified color scheme.

Returned Format

```
[DISPlay:CGRade:SCHeme] {CLASsic | TEMP}<NL>
```

Example This example gets the specified color scheme and prints it on the computer screen:

```
Dim strCgradeScheme As String   ' Dimension variable.
myScope.WriteString ":DISPlay:CGRade:SCHeme?"
strCgradeScheme = myScope.ReadString
Debug.Print strCgradeScheme
```

See Also

- ":DISPlay:CGRade" on page 557
- ":DISPlay:CGRade:LEVels?" on page 560

History New in version 4.20.

:DISPlay:CLIPped

Command `:DISPlay:CLIPped {{0 | OFF} | {1 | ON}}`

The :DISPlay:CLIPped command enables or disables the "show analog-to-digital converter (ADC) clipping" option. (This does not enable or disable the ADC clip detect feature.)

The :DISPlay:CLIPped command maps to the **Show ADC Clipping message** control in the User Preferences dialog box of the front panel user interface.

Query `:DISPlay:CLIPped?`

The :DISPlay:CLIPped? query returns the "show analog-to-digital converter (ADC) clipping" option setting.

Returned Format `<setting><NL>`

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

See Also

- ":CHANnel<N>:CLIPped?" on page 438

History New in version 10.00.

:DISPlay:CONNect

Command `:DISPlay:CONNect {{ON | 1} | {OFF | 0}}[,<source>]`

When enabled, :DISPlay:CONNect draws a line between consecutive waveform data points. This is also known as linear interpolation.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, connected dots is enabled for all sources.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example turns on the connect-the-dots feature.

```
myScope.WriteString ":DISPlay:CONNect ON"
```

Query `:DISPlay:CONNect? [<source>]`

The :DISPlay:CONNect? query returns the status of the connect-the-dots feature.

If <source> is omitted, the query returns ON (1) if connect the dots is enabled on channel 1.

Returned Format `[:DISPlay:CONNect] {1 | 0}<NL>`

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the setting should be made.

:DISPlay:DATA?

Query

```
:DISPlay:DATA? [<type>[,<screen_mode>[,<compression> [,<inversion>]]]]
```

The :DISPlay:DATA? query returns information about the captured data. If no options to the query are specified, the default selections are BMP file type, SCReen mode, compression turned ON, and inversion set to NORMal.

<type> The bitmap type: BMP | JPG | GIF | TIF | PNG.

<screen_mode> The display setting: SCReen | GRATicule. Selecting GRATicule displays a 10-by-8 (unit) display graticule on the screen. See also :DISPlay:GRATicule.

<compression> The file compression feature: ON | OFF.

<inversion> The inversion of the displayed file: NORMal | INVert.

NOTE Error 273 can occur after this query to warn you about a known issue when Remote Desktop is being used to control the oscilloscope. See "List of Error Messages" on page 2020.

Returned Format

```
[:DISPlay:DATA] <binary_block_data><NL>
```

<binary_block _data> Data in the IEEE 488.2 definite block format.

See Also

- "Definite-Length Block Response Data" on page 123

History Legacy command (existed before version 3.10).

:DISPlay:GRATicule

Command

```
:DISPlay:GRATicule {GRID | FRAMe}
```

The :DISPlay:GRATicule command selects the type of graticule that is displayed. Infiniium oscilloscopes have a 10-by-8 (unit) display graticule grid GRID), a grid line is place on each vertical and horizontal division. When it is off (FRAMe), a frame with tic marks surrounds the graticule edges.

Example

This example sets up the oscilloscope's display background with a frame that is separated into major and minor divisions.

```
myScope.WriteString ":DISPlay:GRATicule FRAMe"
```

Query

```
:DISPlay:GRATicule?
```

The :DISPlay:GRATicule? query returns the type of graticule currently displayed.

Returned Format

```
[:DISPlay:GRATicule] {GRID | FRAMe}<NL>
```

Example

This example places the current display graticule setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule:INTensity" on page 572
- ":DISPlay:GRATicule:NUMBer" on page 573
- ":DISPlay:GRATicule:SETGrat" on page 574

History

Legacy command (existed before version 3.10).

:DISPlay:GRATicule:AREA<N>:HSCale

Command

```
:DISPlay:GRATicule:AREA<N>:HSCale {{0 | OFF} | {1 | ON}}
```

The :DISPlay:GRATicule:AREA<N>:HSCale command specifies whether the horizontal scale is displayed.

<N> Integer waveform window number, 1-8.

Query

```
:DISPlay:GRATicule:AREA<N>:HSCale?
```

The :DISPlay:GRATicule:AREA<N>:HSCale? query returns whether the horizontal scale is displayed.

Returned Format

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

See Also

- ":DISPlay:GRATicule:AREA<N>:VSCale" on page 570
- ":DISPlay:GRATicule:AREA<N>:STATe" on page 569

History New in version 11.20.

:DISPlay:GRATicule:AREA<N>:STATe

Command

```
:DISPlay:GRATicule:AREA<N>:STATe {{ON | 1} | {OFF | 0}}
```

The :DISPlay:GRATicule:AREA<N>:STATe command turn a waveform area on or off.

<N> Can be an integer from 1–8. Waveform area 1 is always on and cannot be set to off.

Example This example turns on waveform area 2.

```
myScope.WriteString ":DISPlay:GRATicule:AREA2:STATe ON"
```

Query

```
:DISPlay:GRATicule:AREA<N>:STATe?
```

The :DISPlay:GRATicule:AREA<N>:STATe? query returns whether the waveform area is on or off.

Returned Format

```
[:DISPlay:GRATicule:AREA<N>:STATe] {1 | 0}<NL>
```

Example This example places the status of waveform area 2 in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:AREA2:STATe?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 567
- ":DISPlay:GRATicule:INTensity" on page 572
- ":DISPlay:GRATicule:NUMBer" on page 573
- ":DISPlay:GRATicule:SETGrat" on page 574
- ":DISPlay:GRATicule:AREA<N>:HSCale" on page 568
- ":DISPlay:GRATicule:AREA<N>:VSCale" on page 570

History New in version 5.00.

:DISPlay:GRATicule:AREA<N>:VSCale

Command

```
:DISPlay:GRATicule:AREA<N>:VSCale {OFF | AUTO | MANual}
```

The :DISPlay:GRATicule:AREA<N>:VSCale command specifies whether the vertical scale is displayed with automatic sizing (AUTO), displayed with handles for manual sizing (MANual), or not displayed (OFF).

<N> Integer waveform window number, 1-8.

Query

```
:DISPlay:GRATicule:AREA<N>:VSCale?
```

The :DISPlay:GRATicule:AREA<N>:VSCale? query returns whether the vertical scale is displayed with automatic sizing (AUTO), displayed with handles for manual sizing (MAN), or not displayed (OFF).

Returned Format

```
<setting> ::= Integer waveform window number, 1-8.
```

```
<setting> ::= {OFF | AUTO | MAN}
```

See Also

- ":DISPlay:GRATicule:AREA<N>:HSCale" on page 568
- ":DISPlay:GRATicule:AREA<N>:STATe" on page 569

History New in version 11.20.

:DISPlay:GRATicule:GLAYout

Command

```
:DISPlay:GRATicule:GLAYout <layout>[,<area>]
```

```
<layout> ::= {SVERtical | SHORizontal | TVERtical | THORizontal}
```

The :DISPlay:GRATicule:GLAYout command specifies the layout format to be used in organizing the grids in a waveform area:

- SVERtical — Stacked vertical layout.
- SHORizontal — Stacked horizontal layout.
- TVERtical — Tiled vertical layout.
- THORizontal — Tiled horizontal layout.

When a waveform area is using horizontal Zoom, the :DISPlay:GRATicule:GLAYout command is not applicable and will result in the error -224,"Illegal parameter value".

<area> An integer from 1-8.

If the <area> is omitted, the number of grids will be applied to waveform area 1.

Example This example sets up stacked vertical layout for window 2.

```
myScope.WriteString ":DISPlay:GRATicule:GLAYout SVERtical, 2"
```

Query

```
:DISPlay:GRATicule:GLAYout? [<area>]
```

The :DISPlay:GRATicule:GLAYout? query returns the layout format for the specified area or area 1 if the area is not specified.

Returned Format

```
<layout><NL>
```

```
<layout> ::= {SGV | SGH | TGV | TGH | HZ}
```

- SGV — Stacked vertical layout.
- SGH — Stacked horizontal layout.
- TGV — Tiled vertical layout.
- TGH — Tiled horizontal layout.
- HZ — This is returned If you query while a waveform area is using horizontal Zoom.

See Also

- ":DISPlay:LAYout" on page 585
- ":DISPlay:RESults:LAYout" on page 597

History New in version 6.60.

:DISPlay:GRATicule:INTensity

Command

```
:DISPlay:GRATicule:INTensity <intensity_value>
```

You can dim the grid's intensity or turn the grid off to better view waveforms that might be obscured by the graticule lines using the :DISPlay:GRATicule:INTensity command. Otherwise, you can use the grid to estimate waveform measurements such as amplitude and period.

When printing, the grid intensity control does not affect the hard copy. To remove the grid from a printed hard copy, you must turn off the grid before printing.

<intensity _value> A integer from 0 to 100, indicating the percentage of grid intensity.

Example This example sets the graticule intensity to 50%.

```
myScope.WriteString ":DISPlay:GRATicule:INTensity 50"
```

Query

```
:DISPlay:GRATicule:INTensity?
```

The :DISPlay:GRATicule:INTensity? query returns the intensity.

Returned Format

```
[:DISPlay:GRATicule:INTensity] <value><NL>
```

Example This example places the current graticule intensity setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:INTensity?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 567
- ":DISPlay:GRATicule:NUMBer" on page 573
- ":DISPlay:GRATicule:SETGrat" on page 574

History Legacy command (existed before version 3.10).

:DISPlay:GRATicule:NUMBer

Command

```
:DISPlay:GRATicule:NUMBer <grids>[,<area>]
```

The :DISPlay:GRATicule:NUMBer command specifies the number of grids in a waveform area. Multiple grids let you more easily view multiple waveforms that use the full vertical scale.

<grids> Can be an integer from 1–16.

<area> Can be an integer from 1–8.

If the <area> is omitted, the number of grids will be applied to waveform area 1.

Example This example sets up two viewing areas.

```
myScope.WriteString ":DISPlay:GRATicule:NUMBer 2"
```

Query

```
:DISPlay:GRATicule:NUMBer? [<area>]
```

The :DISPlay:GRATicule:NUMBer? query returns the the number of grids in a waveform area.

Returned Format

```
[:DISPlay:GRATicule:NUMBer] {1-16}<NL>
```

Example This example places the current number of grids in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:NUMBer?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 567
- ":DISPlay:GRATicule:INTensity" on page 572
- ":DISPlay:GRATicule:SETGrat" on page 574
- ":DISPlay:GRATicule:AREA<N>:STATe" on page 569

History Legacy command (existed before version 3.10).

Version 5.0: Number of grids can be any number between 1 and 16 (not just 1, 2, 4, 8, or 16). You can also specify which waveform area the number of grids setting is for.

:DISPlay:GRATicule:SETGrat

Command

```
:DISPlay:GRATicule:SETGrat
   <DispGratChan>,<grid>[,<area>][,{MAIN | CGRade}]
```

The :DISPlay:GRATicule:SETGrat command assigns the corresponding waveform to a specific grid and waveform area.

If {MAIN | CGRade} is omitted, the MAIN view will be placed.

<DispGratChan> Can be:

- CHN<N>
- DIFF1, DIFF2
- COMM3, COMM4
- MEM<N> where N is between 1 and 4
- EQU<N> EQUalized<N> where N is between 1 and 4
- FN<N> where N is between 1 and 16 (function)
- HIST
- INPut (SNDR Input), CORRected (linear fit Pulse Corrected), ERRor (linear fit Error e(k)), and LFPR (Linear Fit Pulse Response p(k)) sources that can be enabled by the Signal to Noise and Distortion Ratio (SNDR) measurements.
- D<M> where M is between 0 and 15 (on MSO models with 16 digital channels)
- BUS<Y> where Y is between 1 and 4 (on MSO models)

<grid> Can be an integer from 1-16; this is the number of the grid you want to assign the waveform to.

<area> Can be an integer from 1–8.

If <area> is omitted, the waveform will be placed in waveform area 1.

Example This example assigns the histogram to grid 2 (in waveform area 1).

```
myScope.WriteString ":DISPlay:GRATicule:SETGrat HIST,2"
```

See Also

- ":DISPlay:GRATicule" on page 567
- ":DISPlay:GRATicule:INTensity" on page 572
- ":DISPlay:GRATicule:NUMBer" on page 573
- ":DISPlay:GRATicule:AREA<N>:STATe" on page 569

History Legacy command (existed before version 3.10).

Version 5.00: In addition to assigning a waveform to a grid, you can now optionally specify which waveform area the grid is in. Also, you can specify whether the MAIN or CGRade (color grade) view of the waveform will be placed.

:DISPlay:ISIM:DGRaphs

Command

```
:DISPlay:ISIM:DGRaphs {{0 | OFF} | {1 | ON}}
```

The :DISPlay:ISIM:DGRaphs command lets you enable or disable the display of InfiniiSim graphs.

You may want to disable the display of InfiniiSim graphs to enable hardware acceleration. (Hardware acceleration is not possible when raw and filtered data are viewed at the same time.)

Query

```
:DISPlay:ISIM:DGRaphs?
```

The :DISPlay:ISIM:DGRaphs? query returns whether InfiniiSim graphs are enabled or disabled.

Returned Format

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

See Also

- ":DISPlay:ISIM:GDCouple" on page 577
- ":CHANnel<N>:ISIM:STATe" on page 466

History New in version 10.10.

:DISPlay:ISIM:GCOunt

Command `:DISPlay:ISIM:GCOunt <count>`

The :DISPlay:ISIM:GCOunt command sets the number of visible graphs in the InfiniiSim plots window area.

<count> An integer in NR1 format.

Up to three plots can be displayed for each channel source (12 total for all sources).

Query `:DISPlay:ISIM:GCOunt?`

The :DISPlay:ISIM:GCOunt? query returns the number of visible graphs in the InfiniiSim plots window area.

Returned Format `[:DISPlay:ISIM:GCOunt] <count><NL>`

See Also

- ":DISPlay:ISIM:SELectgraph" on page 578
- ":DISPlay:ISIM:SOURce" on page 579

History New in version 5.50.

:DISPlay:ISIM:GDCouple

Command

```
:DISPlay:ISIM:GDCouple {{0 | OFF} | {1 | ON}}
```

The :DISPlay:ISIM:GDCouple command specifies whether turning on InfiniiSim (with the :CHANnel<N>:ISIM:STATe command) will automatically display the InfiniiSim graphs.

By default, GDCouple is ON. This allows the oscilloscope to automatically display InfiniiSim graphs when InfiniiSim is turned on, which is how Infiniium oscilloscopes have behaved in the past.

You may want to disable the display of InfiniiSim graphs to enable hardware acceleration. (Hardware acceleration is not possible when raw and filtered data are viewed at the same time.)

Query

```
:DISPlay:ISIM:GDCouple?
```

The :DISPlay:ISIM:GDCouple? query returns the graph display coupling setting.

Returned Format

```
<setting><NL>
```

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

See Also

- ":DISPlay:ISIM:DGRaphs" on page 575
- ":CHANnel<N>:ISIM:STATe" on page 466

History New in version 10.10.

:DISPlay:ISIM:SELectgraph

Command

```
:DISPlay:ISIM:SELectgraph <graph>
```

```
<graph> ::= {SPECtrum | IMPulse | STEP | ALL}
```

The :DISPlay:ISIM:SELectgraph command inserts the specified graph at the first display graph position.

NOTE

Selecting ALL graphs results in all allowed graphs being displayed for the source (see :DISPlay:ISIM:SOURce). If the source is ALL, all graphs will be displayed for all sources. The graph count may change.

See Also

- ":DISPlay:ISIM:GCOunt" on page 576
- ":DISPlay:ISIM:SOURce" on page 579

History New in version 5.50.

:DISPlay:ISIM:SOURce

Command

```
:DISPlay:ISIM:SOURce <source>

<source> ::= {CHANnel<N> | ALL}
```

The :DISPlay:ISIM:SOURce command sets the source for the InfiniiSim plots graph(s).

NOTE Selecting ALL sources causes the selected graph to be applied to all graph sources. If ALL is the selected graph type, all sources are applied to all graphs. The graph count may change.

<N> An integer, 1 to the number of analog input channels.

See Also

- ":DISPlay:ISIM:GCOunt" on page 576
- ":DISPlay:ISIM:SELectgraph" on page 578

History New in version 5.50.

:DISPlay:JITTer:GCOunt

Command `:DISPlay:JITTer:GCOunt <count>`

The :DISPlay:JITTer:GCOunt command sets the number of visible graphs in the Jitter/Noise graphs window area.

<count> An integer in NR1 format.

Query `:DISPlay:JITTer:GCOunt?`

The :DISPlay:JITTer:GCOunt? query returns the number of visible graphs in the Jitter/Noise graphs window area.

Returned Format `[:DISPlay:JITTer:GCOunt] <count><NL>`

See Also

- ":DISPlay:JITTer:SELectgraph" on page 581

History New in version 5.50.

:DISPlay:JITTer:SELectgraph

Command

```
:DISPlay:JITTer:SELectgraph <graph>

<graph> ::= {RPHistogram | TJHistogram | DDJHistogram | JBERbathtub
   | DDJVsbit | JSPectrum | TJComposite | DDJComposite | ISIFilter
   | RJPJsep | JTAilbathtub | NBERbathtub | NSPectrum | RNPHistogram
   | DDIHistogram | TIHistogram | TIComposite | ISIVsbit | NSEParation
   | NISifilter | RNTailhist | NTAilbathtub | ALL}
```

The :DISPlay:JITTer:SELectgraph command inserts the specified graph at the first display graph position.

NOTE Selecting ALL graphs results in all allowed graphs being displayed. The graph count may change.

Graph Option	Description
RPHistogram	RjPj Histogram
TJHistogram	TJ Histogram
DDJHistogram	DDJ Histogram
JBERbathtub	Jitter BER Bathtub
DDJVsbit	DDJ vs. Bit
JSPectrum	Jitter Spectrum
TJComposite	Composite TJ Histogram
DDJComposite	Composite DDJ Histogram
ISIFilter	Jitter ISI Filter
RJPJsep	RJ PJ Separation
JTAilbathtub	Jitter Tail Fit Bathtub
NBERbathtub	Noise BER Bathtub
NSPectrum	Noise Spectrum
RNPHistogram	RN PI Histogram
DDIHistogram	DDI Histogram
TIHistogram	TI Histogram
TIComposite	Composite TI Histogram
ISIVsbit	ISI vs. Bit
NSEParation	RN PI Threshold
NISifilter	Noise ISI Filter

Graph Option	Description
RNTailhist	RN PI Tailfit Histogram
NTAilbathtub	Noise Tailfit Bathtub

See Also

- ":DISPlay:JITTer:GCOunt" on page 580

History New in version 5.50.

:DISPlay:JITTer:THReshold

Command `:DISPlay:JITTer:THReshold <level>`

<level> When the ":MEASure:RJDJ:PAMThreshold ALL" command specifies that all PAM thresholds are measured, the :DISPlay:JITTer:THReshold command specifies whether a certain threshold level or ALL threshold levels should be displayed in the jitter graphs. The <level> syntax is:

- For PAM-3: {T01 | T12 | ALL}
- For PAM-4: {T01 | T12 | T23 | ALL}
- For PAM-6: {T01 | T12 | T23 | T34 | T45 | ALL}
- For PAM-8: {T01 | T12 | T23 | T34 | T45 | T56 | T67 | ALL}

When the ":MEASure:RJDJ:PAMThreshold" command specifies the 0/1, 1/2, or 2/3 PAM-4 thresholds are measured, that level is the one used when displaying jitter graphs.

See Also

- ":MEASure:RJDJ:PAMThreshold" on page 1101

History New in version 6.10.

Version 11.20: Threshold levels for PAM-6 and PAM-8 signal types are now supported.

:DISPlay:LABel

Command

```
:DISPlay:LABel {{ON | 1} | {OFF | 0}}
```

The :DISPlay:LABel command turns on or off the display of analog channel labels. Label names can be up to 6 characters long. The label name is assigned by using the CHANnel<n>:LABel command:

Example This example turns on the display of all labels.

```
myScope.WriteString ":DISPlay:LABel ON"
```

Query

```
:DISPlay:LABel?
```

The :DISPlay:LABel? query returns the current state of the labels.

Returned Format

```
[:DISPlay:LABel] {1 | 0}<NL>
```

Example This example places the current label state into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:LABel?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:DISPlay:LAYout

Command

```
:DISPlay:LAYout <layout>
```

The :DISPlay:LAYout command sets the window layout.

<layout> {{TAB | CUSTom} | SVERtical | SHORizontal}

- TAB (or CUSTom) — Tabbed window layout.
- SVERtical — Stack windows vertically.
- SHORizontal — Stack windows horizontally.

Query

```
:DISPlay:LAYout?
```

The :DISPlay:LAYout? query returns the window layout setting.

Returned Format

```
[DISPlay:LAYout] <layout><NL>

<layout> ::= {TAB | SVER | SHOR}
```

See Also

- ":DISPlay:PROPortion" on page 592

History New in version 5.00.

Version 10.10: The obsolete CUSTom option has been replaced with the new TAB option.

:DISPlay:MAIN

Command

```
:DISPlay:MAIN {{ON | 1} | {OFF | 0}}[,<source>]
```

The :DISPlay:MAIN command turns on or off the main window view for the indicated source.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, the main window view is enabled/disabled for all sources that are currently on, except for digital channel sources.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the main view on.

```
myScope.WriteString ":DISPlay:MAIN ON"
```

Query

```
:DISPlay:MAIN? [<source>]
```

The DISPlay:MAIN? returns whether the main window view for the indicated source is on or off.

If <source> is omitted, the query returns ON (1) if any main window view is enabled.

Returned Format

```
[:DISPlay:MAIN] {1 | 0}<NL>
```

Example This example returns the main window view state.

```
Dim strMain As String   ' Dimension variable.
myScope.WriteString ":DISPlay:MAIN?"
strCgrade = myScope.ReadString
Debug.Print strMain
```

See Also

- ":DISPlay:CGRade" on page 557

History New in version 5.00.

:DISPlay:NOISe:LEVel

Command `:DISPlay:NOISe:LEVel <level>`

The :DISPlay:NOISe:LEVel command specifies which of the noise graphs to display.

<level> {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7}

For a Non Return to Zero (NRZ) signal, you can display the noise graphs for levels 0 or 1.

For a PAM signal, you can you can display the noise graphs for the levels:

- For PAM-3, this may be from 0-2.
- For PAM-4, this may be from 0-3.
- For PAM-6, this may be from 0-5.
- For PAM-8, this may be from 0-7.

See Also

- ":MEASure:NOISe" on page 989
- ":MEASure:NOISe:ALL?" on page 991
- ":MEASure:NOISe:BANDwidth" on page 993
- ":MEASure:NOISe:LOCation" on page 994
- ":MEASure:NOISe:METHod" on page 995
- ":MEASure:NOISe:REPort" on page 996
- ":MEASure:NOISe:RN" on page 997
- ":MEASure:NOISe:SCOPe:RN" on page 999
- ":MEASure:NOISe:STATe" on page 1000
- ":MEASure:NOISe:UNITs" on page 1001

History New in version 6.10.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:DISPlay:PAM:GRAPh

Command

```
:DISPlay:PAM:GRAPh <string_of_choices>
```

The :DISPlay:PAM:GRAPh command selects the PAM-4 12-Edge Jitter graphs to display.

<string_of_choices>

A comma-separated list of graphical user interface graph names.

When a misspelled or unrecognized string is encountered, the command will ignore that selection and move on to the next in the list.

Examples

Here are some examples of how to specify the PAM-4 12-Edge Jitter graphs to display.

```
myScope.WriteString ':DISPlay:PAM:GRAPh "Composite Histogram"'
myScope.WriteString ':DISPlay:PAM:GRAPh "L0 to L1,L1 to L0"'
myScope.WriteString ':DISPlay:PAM:GRAPh "AllItemsChecked"'
myScope.WriteString ':DISPlay:PAM:GRAPh ""'
```

Specifying "AllItemsChecked" for the string of choices will select all graphs.

Specifying an empty string ("") will select none of the graphs.

See Also

- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":DISPlay:PAM:SOURce" on page 589
- ":DISPlay:PAM:TABLe" on page 590

History

New in version 11.30.

:DISPlay:PAM:SOURce

Command `:DISPlay:PAM:SOURce <source_string>`

The :DISPlay:PAM:SOURce command selects the PAM-4 12-Edge Jitter sources whose tables and graphs will be displayed.

<source_string> A comma-separated list of graphical user interface source names.

When a misspelled or unrecognized string is encountered, the command will ignore that selection and move on to the next in the list.

Examples Here are some examples of how to specify the PAM-4 12-Edge Jitter sources.

```
myScope.WriteString ':DISPlay:PAM:SOURce "Memory 1"'
myScope.WriteString ':DISPlay:PAM:SOURce "Channel 1"'
myScope.WriteString ':DISPlay:PAM:SOURce "Memory 1, Memory 2"'
myScope.WriteString ':DISPlay:PAM:SOURce "AllItemsChecked"'
myScope.WriteString ':DISPlay:PAM:SOURce ""'
```

Specifying "AllItemsChecked" for the string of choices will select all available sources (that have PAM-4 12-Edge Jitter enabled).

Specifying an empty string ("") will select the lowest available source (one must remain selected).

See Also

- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":DISPlay:PAM:GRAPh" on page 588
- ":DISPlay:PAM:TABLe" on page 590

History New in version 11.30.

:DISPlay:PAM:TABLe

Command

```
:DISPlay:PAM:TABLe <string_of_choices>
```

The :DISPlay:PAM:TABLe command selects the PAM-4 12-Edge Jitter tables to display.

<string_of_choices>

A comma-separated list of graphical user interface table names.

When a misspelled or unrecognized string is encountered, the command will ignore that selection and move on to the next in the list.

Examples

Here are some examples of how to specify the PAM-4 12-Edge Jitter tables to display.

```
myScope.WriteString ':DISPlay:PAM:TABLe "J3u Table"'
myScope.WriteString ':DISPlay:PAM:TABLe "J3u Table,J4u Table,
   Jrms Table,EOJ Table"'
myScope.WriteString ':DISPlay:PAM:TABLe "AllItemsChecked"'
myScope.WriteString ':DISPlay:PAM:TABLe ""'
```

Specifying "AllItemsChecked" for the string of choices will select all tables.

Specifying an empty string ("") will select none of the tables.

See Also

- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":DISPlay:PAM:GRAPh" on page 588
- ":DISPlay:PAM:SOURce" on page 589

History

New in version 11.30.

:DISPlay:PERSistence

Command

```
:DISPlay:PERSistence {MINimum | INFinite | <time>}[,<source>]
```

```
<time> ::= seconds in in NR3 format from 100E-3 to 200E0
```

The :DISPlay:PERSistence command sets the display persistence. The parameter for this command can be:

- MINimum — indicates zero persistence.
- INFinite — indicates infinite persistence.
- <time> — for variable persistence, that is, you can specify how long acquisitions remain on the screen.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

The <source> option is allowed to support earlier syntax, and there are no errors when you use it; however, persistence is always applied to all waveforms whether the <source> option is included or not.

The Color Grade View, a variation of infinite persistence, can be applied to individual waveforms (using the :DISPlay:CGRade command).

For the WMEMory<R> source, the only valid persistence value is MINimum.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the persistence to infinite.

```
myScope.WriteString ":DISPlay:PERSistence INFinite"
```

Query

```
:DISPlay:PERSistence? [<source>]
```

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

When <source> is omitted, the query returns the persistence mode for channel 1.

Returned Format

```
[:DISPlay:PERSistence] {MINimum | INFinite | <time>}<NL>
```

Example This example places the current persistence setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:PERSistence?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:CGRade" on page 557

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the persistence setting should be made.

:DISPlay:PROPortion

Command

```
:DISPlay:PROPortion <pane>, <float>
```

The :DISPlay:PROPortion command specifies the size of the waveform and plot areas.

If the :DISPlay:LAYout is VERTical, this command sets an area's height.

If the :DISPlay:LAYout is HORizontal, this command sets an area's width.

If the :DISPlay:LAYout is CUSTom, this command is not supported.

<pane> {AREA<N> | SERial<M> | {JITTer | NOISe} | ISIM | PRECprobe | BUS | P4Jitter}

<N> An integer, 1-8.

<M> An integer, 1-4.

<float> A value from 0.0 to 100.0.

Example You should set the proportion of all areas that are displayed such that the sum of the proportions is 100. For example, if you have three areas on: Waveform Area1, Waveform Area 2, and Jitter graphs, you may want to size them as follows:

```
:DISPlay:PROPortion AREA1, 20.0
:DISPlay:PROPortion AREA2, 20.0
:DISPlay:PROPortion JITTer, 60.0
```

If you set the size of one area only, it may not have the intended effect.

See Also

- ":DISPlay:PROPortion:RESults" on page 593
- ":DISPlay:LAYout" on page 585

History New in version 5.00.

Version 10.20: The query portion of this command has been deprecated. The P4Jitter (PAM4 Jitter) pane option has been added.

:DISPlay:PROPortion:RESults

Command `:DISPlay:PROPortion:RESults <float>`

The :DISPlay:PROPortion:RESults command specifies the size of the Results pane in the oscilloscope display.

<float> A value from 0.0 to 100.0.

See Also

- ":DISPlay:PROPortion" on page 592

History New in version 5.70.

Version 6.50: The query portion of this command has been deprecated.

:DISPlay:PRECprobe:GCOunt

Command `:DISPlay:PRECprobe:GCOunt <count>`

The :DISPlay:PRECprobe:GCOunt command sets the number of visible graphs in the PrecisionProbe correction and analysis charts window area.

<count> An integer in NR1 format.

Up to six charts can be displayed for each channel source (24 total for all sources).

Query `:DISPlay:PRECprobe:GCOunt?`

The :DISPlay:PRECprobe:GCOunt? query returns the number of visible graphs in the PrecisionProbe correction and analysis charts window area.

Returned Format `[:DISPlay:PRECprobe:GCOunt] <count><NL>`

See Also

- ":DISPlay:PRECprobe:SELectgraph" on page 595
- ":DISPlay:PRECprobe:SOURce" on page 596

History New in version 5.50.

:DISPlay:PRECprobe:SELectgraph

Command

```
:DISPlay:PRECprobe:SELectgraph <graph>
```

```
<graph> ::= {FRPHase | IMAG | IPHase | FFRMag | FFRPhase | MFRMag | ALL}
```

The :DISPlay:PRECprobe:SELectgraph command inserts the specified graph at the first display graph position.

NOTE

Selecting ALL graphs results in all allowed graphs being displayed for the source (see :DISPlay:PRECprobe:SOURce). If the source is ALL, all graphs will be displayed for all sources. The graph count may change.

See Also

- ":DISPlay:PRECprobe:GCOunt" on page 594
- ":DISPlay:PRECprobe:SOURce" on page 596

History New in version 5.50.

:DISPlay:PRECprobe:SOURce

Command

```
:DISPlay:PRECprobe:SOURce <source>

<source> ::= {CHANnel<N> | ALL}
```

The :DISPlay:PRECprobe:SOURce command sets the source for the PrecisionProbe correction and analysis chart(s).

NOTE

Selecting ALL sources causes the selected graph to be applied to all graph sources. If ALL is the selected graph type, all sources are applied to all graphs. The graph count may change.

<N> An integer, 1 to the number of analog input channels.

See Also

- ":DISPlay:PRECprobe:GCOunt" on page 594
- ":DISPlay:PRECprobe:SELectgraph" on page 595

History New in version 5.50.

:DISPlay:RESults:LAYout

Command `:DISPlay:RESults:LAYout <layout>`

The :DISPlay:RESults:LAYout command sets the Results pane's window layout.

<layout> {{TAB | CUSTom} | SHORizontal}

- TAB (or CUSTom) — Tabbed window layout.
- SHORizontal — Stack windows horizontally.

Query `:DISPlay:RESults:LAYout?`

The :DISPlay:RESults:LAYout? query returns the Results pane window layout setting.

Returned Format `[:DISPlay:RESults:LAYout] <layout><NL>`

`<layout> ::= {TAB | SHOR}`

See Also

- ":DISPlay:LAYout" on page 585

History New in version 10.20.

:DISPlay:SCOLor

Command

```
:DISPlay:SCOLor <color_name>, <hue>, <saturation>, <luminosity>
```

The :DISPlay:SCOLor command sets the color of the specified display element. The display elements are described in Table 14.

<color_name>

```
{BUS | CGLevel1 - CGLevel7 | CHANnel1 - CHANnel4 | DCHannel | DMEMory
   | FUNCtion1 - FUNCtion16 | GRID | HISTogram | MARKers | MTPolygons
   | MMPolygons | TINPuts | WMEMories | WMEMory1 - WMEMory4}
```

Table 14 Color Names

Color Name	Definition
BUS	Buses.
CGLevel1 - CGLevel7	Color Grade Level 1 through Level 7 waveform display elements.
CHANnel1 - CHANnel4	Channel 1 through Channel 4 waveform display elements.
DCHannel	Digital channels.
DMEMory	Digital waveform memory.
FUNCtion1 - FUNCtion16	Function 1 through Function 16 waveform display elements.
GRID	Display element for the grid inside the waveform viewing area.
HISTogram	Histogram bars.
MARKers	Display element for the markers.
MTPolygons	Mask test regions.
MMPolygons	Mask test margin regions.
TINPuts	Display element for line and aux trigger colors.
WMEMories	Display element for waveform memories (same as WMEMory1).
WMEMory1 - WMEMory4	Waveform Memory 1 through Waveform Memory 4 display elements.

<hue> An integer from 0 to 100. The hue control sets the color of the chosen display element. As hue is increased from 0%, the color changes from red, to yellow, to green, to blue, to purple, then back to red again at 100% hue. For color examples, see the sample color settings table in the Infiniium Oscilloscope online help file. Pure red is 100%, pure blue is 67%, and pure green is 33%.

<saturation> An integer from 0 to 100. The saturation control sets the color purity of the chosen display element. The saturation of a color is the purity of a color, or the absence of white. A 100% saturated color has no white component. A 0% saturated color is pure white.

<luminosity> An integer from 0 to 100. The luminosity control sets the color brightness of the chosen display element. A 100% luminosity is the maximum color brightness. A 0% luminosity is pure black.

Example This example sets the hue to 50, the saturation to 70, and the luminosity to 90 for the markers.

```
myScope.WriteString ":DISPlay:SCOLor MARKers,50,70,90"
```

Query

```
:DISPlay:SCOLor? <color_name>
```

The :DISPlay:SCOLor? query returns the hue, saturation, and luminosity for the specified color.

Returned Format

```
[:DISPlay:SCOLor] <color_name>, <hue>, <saturation>, <luminosity><NL>
```

Example This example places the current settings for the graticule color in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:SCOLor? GRID"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 5.60: Removed the ability to set colors for MEASurements, STEXt, TSCale, and WBACkgrnd.

Version 5.70: Added the ability to set colors for MMPolygons.

:DISPlay:STATus:COLumn

Command

```
:DISPlay:STATus:COLumn <column>
```

The :DISPlay:STATus:COLumn command is used to position the real time eye and InfiniiScan Zone Trigger status labels.

This and the :DISPlay:STATus:ROW commands specify the upper left corner of the box relative to the screen.

<column> A value of 0 to 1 may be given for the column where 0 is the far left and 1 the far right.

Example For example, a column of 0.5 will place the upper left of the status label at the center screen.

```
myScope.WriteString ":DISPlay:STATus:COLumn 0.5"
```

Query

```
:DISPlay:STATus:COLumn?
```

The :DISPlay:STATus:COLumn? query returns the current value of the status label column location.

Returned Format

```
[:DISPlay:STATus:COLumn] <column><NL>
```

Example This example places the current value for the status label column location in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:STATus:COLumn?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History New in version 3.10.

:DISPlay:STATus:ROW

Command

```
:DISPlay:STATus:ROW <row>
```

The :DISPlay:STATus:ROW command is used to position the real time eye and InfiniiScan Zone Trigger status labels.

This and the :DISPlay:STATus:COL commands specify the upper left corner of the box relative to the screen.

<row> A value of 0 to 1 may be given for the row where 0 is the far top and 1 the far bottom.

Example For example, a row and column of 0.5 will place the upper left of the status label at the center screen.

```
myScope.WriteString ":DISPlay:STATus:ROW 0.5"
```

Query

```
:DISPlay:STATus:ROW?
```

The :DISPlay:STATus:ROW? query returns the current value of the status label row location.

Returned Format

```
[:DISPlay:STATus:ROW] <row><NL>
```

Example This example places the current value for the status label row location in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:STATus:ROW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History New in version 3.10.

:DISPlay:THEMe

Command

```
:DISPlay:THEMe {"Midnight" | "Platinum"}
```

The :DISPlay:THEMe command lets you select the graphical user interface's "Midnight" or "Platinum" theme.

Query

```
:DISPlay:THEMe?
```

The :DISPlay:THEMe? query returns the selected user interface theme.

Returned Format

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

```
<string> ::= {"Midnight" | "Platinum"}
```

History New in version 6.60.

:DISPlay:WINDow:MAXimize

Command

```
:DISPlay:WINDow:MAXimize <window>
```

```
<window> ::= {AREA<N> | SERial<M> | {JITTer | NOISe} | ISIM
   | PRECprobe | BUS | P4Jitter}
```

The :DISPlay:WINDow:MAXimize command will maximize the size of the specified window.

<N> An integer, 1-8.

<M> An integer, 1-4.

History New in version 5.50.

Version 10.20: The P4Jitter (PAM4 Jitter) window option has been added.

23 :DVM (Digital Voltmeter) Commands

These commands control the digital voltmeter (DVM) feature.

:DVM:ARANge

Command

```
:DVM:ARANge {{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

```
:DVM:ARANge?
```

The :DVM:ARANge? query returns a flag indicating whether the digital voltmeter's Auto Range capability is on or off.

Returned Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 610
- ":DVM:ENABle" on page 608
- ":DVM:MODE" on page 609

History New in version 11.00.

:DVM:CURRent?

Query `:DVM:CURRent?`

The :DVM:CURRent? query returns the displayed 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.

Returned Format `<dvm_value><NL>`

`<dvm_value> ::= floating-point number in NR3 format`

See Also

- ":DVM:SOURce" on page 610
- ":DVM:ENABle" on page 608
- ":DVM:MODE" on page 609

History New in version 11.00.

:DVM:ENABle

Command

```
:DVM:ENABle {{OFF | 0} | {ON | 1}}
```

The :DVM:ENABle command turns the digital voltmeter (DVM) analysis feature on or off.

Query

```
:DVM:ENABle?
```

The :DVM:ENABle? query returns a flag indicating whether the digital voltmeter (DVM) analysis feature is on or off.

Returned Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 610
- ":DVM:MODE" on page 609
- ":DVM:ARANge" on page 606

History

New in version 11.00.

:DVM:MODE

Command

```
:DVM:MODE {DC | DCRMs | ACRMs}
```

The :DVM:MODE command sets the digital voltmeter (DVM) mode:

- DC — displays the DC value of the acquired data.
- DCRMs — displays the root-mean-square value of the acquired data.
- ACRMs — displays the root-mean-square value of the acquired data, with the DC component removed.

Query

```
:DVM:MODE?
```

The :DVM:MODE? query returns the selected DVM mode.

Returned Format

```
<dvm_mode><NL>
```

```
<dvm_mode> ::= {DC | DCRM | ACRM}
```

See Also

- ":DVM:ENABle" on page 608
- ":DVM:SOURce" on page 610
- ":DVM:ARANge" on page 606
- ":DVM:CURRent?" on page 607

History New in version 11.00.

:DVM:SOURce

Command `:DVM:SOURce {CHANnel<n>`

The :DVM:SOURce command selects 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.

<N> 1 to (# analog channels) in NR1 format.

Query `:DVM:SOURce?`

The :DVM:SOURce? query returns the selected DVM input source.

Returned Format `<source><NL>`

`<source> ::= {CHAN<n>}`

See Also

- ":DVM:ENABle" on page 608
- ":DVM:MODE" on page 609
- ":DVM:ARANge" on page 606
- ":DVM:CURRent?" on page 607

History New in version 11.00.

24 :FUNCtion<F> Commands

The FUNCtion subsystem defines functions 1-16. The operands of these functions can be:

- Any of the installed channels in the oscilloscope
- Differential channels or common mode channels

- Waveform memories
- Crosstalk waveforms
- Functions
- Equalization lane function waveforms
- A constant
- Jitter measurement trend or jitter spectrum
- Phase noise frequency domain waveform
- SNDR measurement function waveforms

You can control the vertical scaling and offset functions remotely using the RANGe and OFFSet commands in this subsystem. You can obtain the horizontal scaling and position values of the functions using the :HORizontal:RANGe? and :HORizontal:POSition? queries in this subsystem.

If a channel is not on but is used as an operand, that channel will acquire waveform data.

If the operand waveforms have different memory depths, the function uses the shorter of the two.

If the two operands have the same time scales, the resulting function has the same time scale. If the operands have different time scales, the resulting function has no valid time scale. This is because operations are performed based on the displayed waveform data position, and the time relationship of the data records cannot be considered. When the time scale is not valid, delta time pulse parameter measurements have no meaning, and the unknown result indicator is displayed on the screen.

Constant operands take on the same time scale as the associated waveform operand. Constant operands can be a real number from -1E6 to 1E12.

When specifying function <operand>s, you can choose from waveform source(s) or a constant value:

```
<operand> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | EQUalized<L> | WMEMory<R> | <constant_float_value> | MTRend
   | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}
```

Another function can be a function's source as long as the other function does not use the function being defined. In other words, circular expressions are not allowed.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

:FUNCtion<F>?

Query

```
:FUNCtion<F>?
```

The :FUNCtion<F>? query returns the currently defined source(s) for the function.

Returned Format

```
[:FUNCtion<F>:<operator>] {<operand>[,<operand>]}<NL>
```

<F> An integer, 1-16, representing the selected function.

<operator> Active math operation for the selected function. For example, ADD, AVERage, COMMonmode, DIFF, DIVide, FFTMagnitude, FFTPhase, HIGHpass, INTegrate, INVert, LOWPass, MAGNify, MAXimum, MINimum, MULTiply, SMOoth, SUBTract, or VERSus.

<operand> Any allowable source for the selected FUNCtion, including channels, differential channels, common mode channels. waveform memories 1-4, functions 1-4, a constant, jitter measurement trend, and jitter spectrum. If the function is applied to a constant, the source returns the constant.

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example returns the currently defined source for function 1.

```
myScope.WriteString ":FUNCtion1?"
```

If the headers are off (see :SYSTem:HEADer), the query returns only the operands, not the operator.

```
myScope.WriteString ":SYST:HEAD ON"
myScope.WriteString ":FUNC1:ADD CHAN1,CHAN2"
myScope.WriteString ":FUNC1?"
strSettings = myScope.ReadString   ' Returns ":FUNC1:ADD CHAN1,CHAN2".
myScope.WriteString ":SYST:HEAD OFF"
myScope.WriteString ":FUNC1?"
strSettings = myScope.ReadString   ' Returns "CHAN1,CHAN2".
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:ABSolute

Command

```
:FUNCtion<F>:ABSolute <operand>
```

The :FUNCtion<F>:ABSolute command takes the absolute value an operand.

<operand> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<R> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example turns on the absolute value command using channel 3.

```
myScope.WriteString ":FUNCtion1:ABSolute CHANnel3"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:ADD

Command

```
:FUNCtion<F>:ADD <operand>,<operand>
```

The :FUNCtion<F>:ADD command defines a function that takes the algebraic sum of the two operands.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to add channel 1 to channel 2.

```
myScope.WriteString ":FUNCtion1:ADD CHANnel1,CHANnel2"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:ADEMod

Command

```
:FUNCtion<F>:ADEMod <source>
```

The :FUNCtion<F>:ADEMod command sets the math function to show 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.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to perform the amplitude demodulation function on channel 1.

```
myScope.WriteString ":FUNCtion1:ADEMod CHANnel1"
```

History New in version 4.50.

:FUNCtion<F>:AVERage

Command

```
:FUNCtion<F>:AVERage <operand>[,<averages>]
```

The :FUNCtion<F>:AVERage command defines a function that averages the operand based on the number of specified averages.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<averages> An integer, 2 to 65534 specifying the number of waveforms to be averaged

Example This example sets up function 1 to average channel 1 using 16 averages.

```
myScope.WriteString ":FUNCtion1:AVERage CHANnel1,16"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:COMMonmode

Command

```
:FUNCtion<F>:COMMonmode <operand>,<operand>
```

The :FUNCtion<F>:COMMonmode command defines a function that adds the voltage values of the two operands and divides by 2, point by point.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to view the common mode voltage value of channel 1 and channel 2.

```
myScope.WriteString ":FUNCtion1:COMMonmode CHANnel1,CHANnel2"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:DELay — Delay

Command

```
:FUNCtion<F>:DELay <operand>,<delay_time>
```

The :FUNCtion<F>:DELay command adds the provided time to the X origin of the source waveform, effectively shifting the function waveform in time.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<delay_time> Time, in seconds, set for the delay.

Example This example sets function 2 to be the waveform from channel1, delayed by 100 ps.

```
myScope.WriteString ":FUNCtion2:DELay CHANnel1,100E-12"
```

History New in version 4.30.

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:DIFF — Differentiate

Command

```
:FUNCtion<F>:DIFF <operand>[,<low_pass_phase_align>]
```

The :FUNCtion<F>:DIFF command defines a function that computes the discrete derivative of the operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<low_pass_phase_align> {{ON | 1} | {OFF | 0}

This parameter turns on or off the low pass and phase align filter.

Example This example sets up function 2 to take the discrete derivative of the waveform on channel 2.

```
myScope.WriteString ":FUNCtion2:DIFF CHANnel2"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:DISPlay

Command

```
:FUNCtion<F>:DISPlay {{ON|1} | {OFF|0}}
```

The :FUNCtion<F>:DISPlay command either displays the selected function or removes it from the display.

<F> An integer, 1-16, representing the selected function.

Example This example turns function 1 on.

```
myScope.WriteString ":FUNCtion1:DISPlay ON"
```

Query

```
:FUNCtion<F>:DISPlay?
```

The :FUNCtion<F>:DISPlay? query returns the displayed status of the specified function.

Returned Format

```
[:FUNCtion<F>:DISPlay] {1|0}<NL>
```

Example This example places the current state of function 1 in the variable, strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":FUNCtion1:DISPlay?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:DIVide

Command

```
:FUNCtion<F>:DIVide <operand>,<operand>
```

The :FUNCtion<F>:DIVide command defines a function that divides the first operand by the second operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

Example This example sets up function 2 to divide the waveform on channel 1 by the waveform in waveform memory 4.

```
myScope.WriteString ":FUNCtion2:DIVide CHANnel1,WMEMory4"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFT:DETector:POINts

Command

```
:FUNCtion<F>:FFT:DETector:POINts <number_of_buckets>
```

```
<number_of_buckets> ::= an integer.
```

When a detector is used for the FFT magnitude function (see :FUNCtion<F>:FFT:DETector:TYPE), the :FUNCtion<F>:FFT:DETector:POINts command specifies the maximum number of points (buckets) that detectors should decimate to.

Query

```
:FUNCtion<F>:FFT:DETector:POINts?
```

The :FUNCtion<F>:FFT:DETector:POINts? query returns the specified number of detector points.

Returned Format

```
<number_of_buckets><NL>
```

See Also

- ":FUNCtion<F>:FFT:DETector:TYPE" on page 625
- ":FUNCtion<F>:FFTMagnitude" on page 645
- ":FUNCtion<F>:FFT:VUNits" on page 642

History New in version 5.70.

:FUNCtion<F>:FFT:DETector:TYPE

Command

```
:FUNCtion<F>:FFT:DETector:TYPE <type>

<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage}
```

The :FUNCtion<F>:FFT:DETector:TYPE command specifies whether a detector is used for the FFT magnitude function.

Detectors decimate the number of points on screen to at most the number of detector points (buckets, see :FUNCtion<F>:FFT:DETector:POINts). Detectors give you a way of manipulating the acquired data to emphasize different features of the data. The detector types are:

- OFF — No detector is used.
- SAMPle — Takes the point nearest to the center of every bucket.
- PPOSitive — Takes the most positive point in every bucket.
- PNEGative — Takes the most negative point in every bucket.
- NORMal — Implements a rosenfell algorithm. For details, see the Spectrum Analysis Basics application note.
- AVERage — Takes the average of all points in every bucket.

Query

```
:FUNCtion<F>:FFT:DETector:TYPE?
```

The :FUNCtion<F>:FFT:DETector:TYPE? query returns the selected detector.

Returned Format

```
<type><NL>

<type> ::= {OFF | SAMP | PPOS | PNEG | NORM | AVER}
```

See Also

- ":FUNCtion<F>:FFT:DETector:POINts" on page 624
- ":FUNCtion<F>:FFTMagnitude" on page 645
- ":FUNCtion<F>:FFT:VUNits" on page 642

History

New in version 5.70.

Version 10.10: The RMS detector type is no longer available.

:FUNCtion<F>:FFT:FREQuency

Command

```
:FUNCtion<F>:FFT:FREQuency <center_frequency_value>
```

The :FUNCtion<F>:FFT:FREQuency command sets the center frequency for the FFT when :FUNCtion<F>:FFTMagnitude is defined for the selected function.

<F> An integer, 1-16, representing the selected function.

<center_frequency_value> A real number for the value in Hertz, from -1E12 to 1E12.

Query

```
:FUNCtion<F>:FFT:FREQuency?
```

The :FUNCtion<F>:FFT:FREQuency? query returns the center frequency value.

Returned Format

```
[FUNCtion<F>:FFT:FREQuency] <center_frequency_value><NL>
```

See Also

- ":FUNCtion<F>:FFT:STOP" on page 640
- ":FUNCtion<F>:FFT:SPAN" on page 639
- ":FUNCtion<F>:FFT:RESolution" on page 637

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFT:HSCale

Command

```
:FUNCtion<F>:FFT:HSCale {LINear | LOG}
```

For a FFT math function waveform, the :FUNCtion<F>:FFT:HSCale command specifies whether the horizontal scale is linear or logarithmic.

Query

```
:FUNCtion<F>:FFT:HSCale?
```

The :FUNCtion<F>:FFT:HSCale? query returns the horizontal scale setting.

Returned Format

```
<type><NL>
```

```
<type> ::= {LIN | LOG}
```

See Also

- ":WMEMory<R>:FFT:HSCale" on page 1771

History

New in version 10.10.

:FUNCtion<F>:FFT:IMPedance

Command

```
:FUNCtion<F>:FFT:IMPedance {AUTO | <impedance>}
```

When the FFT vertical units are displayed (and measured) as power (that is, dBm or Watt – see :FUNCtion<F>:FFT:VUNits), the :FUNCtion<F>:FFT:IMPedance command lets you specify the reference impedance of the waveform source so that power is calculated correctly. You can select automatically determined or manually entered values.

AUTO When AUTO is selected, analog input channel reference impedances can be automatically determined by the type of probe detected (differential, common mode, single-ended) or by whether two channels are set up as Differential Channels or Common Mode Channels using the :CHANnel<N>:DIFFerential command.

For:	The automatically determined Reference Impedance is:
Single-ended input channels and almost all high-impedance probes	50 Ω
Differential input channels[1] and differential probes	100 Ω
Common Mode input channels[1] and common mode probes[2]	25 Ω
NOTES: [1]The Differential Channels or Common Mode Channels selections (see :CHANnel<N>:DIFFerential) take precedence over the detected probe type. [2]Common mode is not available on high-impedance probes.	

All other sources (math functions, waveform memories, etc.) are assumed to be 50 **Ω**. If this is not correct, for example, when a waveform memory or math function source comes from a differential or common mode signal, you must manually enter the appropriate reference impedance.

<impedance> Reference impedance from 20 **Ω** to 200 **Ω**. in NR3 format.

Query

```
:FUNCtion<F>:FFT:IMPedance?
```

The :FUNCtion<F>:FFT:IMPedance? query returns AUTO or the reference impedance setting.

Returned Format

```
<setting><NL>

<setting> :: = {AUTO | <impedance>}
```

See Also

- ":FUNCtion<F>:FFTMagnitude" on page 645
- ":FUNCtion<F>:FFT:VUNits" on page 642

- ":CHANnel<N>:DIFFerential" on page 440

History New in version 6.60.

:FUNCtion:FFT:PEAK:SORT

Command

```
:FUNCtion:FFT:PEAK:SORT {DMAGnitude | DFRequency | IMAGnitude
    | IFRequency}
```

The :FUNCtion:FFT:PEAK:SORT command specifies whether peaks are annotated by Decreasing Magnitude (DMAGnitude), Increasing Frequency (IFRequency), Decreasing Frequency (DFRequency), or Increasing Magnitude (IMAGnitude) order. Peak annotations are numbered according to the chosen sort.

<F> An integer, 1-16, representing the selected function.

Query

```
:FUNCtion:FFT:PEAK:SORT?
```

The :FUNCtion:FFT:PEAK:SORT? query returns the selected peak annotation sort order.

Returned Format

```
<order><NL>

<order> ::= {DMAG | DFR | IMAG | IFR}
```

See Also

- ":FUNCtion<F>:FFT:PEAK:COUNt" on page 631
- ":FUNCtion<F>:FFT:PEAK:FREQuency" on page 632
- ":FUNCtion<F>:FFT:PEAK:LEVel" on page 633
- ":FUNCtion<F>:FFT:PEAK:MAGNitude" on page 634
- ":FUNCtion<F>:FFT:PEAK:STATe" on page 635

History New in version 11.15.

:FUNCtion<F>:FFT:PEAK:COUNt

Command

```
:FUNCtion<F>:FFT:PEAK:COUNt <max_peaks_to_find>
```

The :FUNCtion<F>:FFT:PEAK:COUNt command specifies the maximum number of peaks in the FFT to annotate.

<F> An integer, 1-16, representing the selected function.

<max_peaks_to_find> Integer value in NR1 format.

Query

```
:FUNCtion<F>:FFT:PEAK:COUNt?
```

The :FUNCtion<F>:FFT:PEAK:COUNt? query returns the specified maximum number of peaks in the FFT to annotate.

Returned Format

```
<max_peaks_to_find><NL>

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

See Also

- ":FUNCtion:FFT:PEAK:SORT" on page 630
- ":FUNCtion<F>:FFT:PEAK:FREQuency" on page 632
- ":FUNCtion<F>:FFT:PEAK:LEVel" on page 633
- ":FUNCtion<F>:FFT:PEAK:MAGNitude" on page 634
- ":FUNCtion<F>:FFT:PEAK:STATe" on page 635

History New in version 11.15.

:FUNCtion<F>:FFT:PEAK:FREQuency

Query `:FUNCtion<F>:FFT:PEAK:FREQuency?`

The :FUNCtion<F>:FFT:PEAK:FREQuency? query returns a comma-separated string of annotated peak frequency values.

Values are returned according to the order specified by the :FUNCtion:FFT:PEAK:SORT command.

If no peaks are found, 9.99e37 is returned.

<F> An integer, 1-16, representing the selected function.

Returned Format `<string><NL>`

`<string> ::= string of comma-separated frequency values`

See Also

- ":FUNCtion:FFT:PEAK:SORT" on page 630
- ":FUNCtion<F>:FFT:PEAK:COUNt" on page 631
- ":FUNCtion<F>:FFT:PEAK:LEVel" on page 633
- ":FUNCtion<F>:FFT:PEAK:MAGNitude" on page 634
- ":FUNCtion<F>:FFT:PEAK:STATe" on page 635

History New in version 11.15.

:FUNCtion<F>:FFT:PEAK:LEVel

Command

```
:FUNCtion<F>:FFT:PEAK:LEVel <peak_search_level>
```

The :FUNCtion<F>:FFT:PEAK:LEVel command specifies the level above which peaks are identified.

<F> An integer, 1-16, representing the selected function.

<peak_search_level> Floating-point value in NR3 format.

Query

```
:FUNCtion<F>:FFT:PEAK:LEVel?
```

The :FUNCtion<F>:FFT:PEAK:LEVel? query returns the specified level above which peaks are identified.

Returned Format

```
<peak_search_level><NL>

<peak_search_level> ::= float value in NR3 format
```

See Also

- ":FUNCtion:FFT:PEAK:SORT" on page 630
- ":FUNCtion<F>:FFT:PEAK:COUNt" on page 631
- ":FUNCtion<F>:FFT:PEAK:FREQuency" on page 632
- ":FUNCtion<F>:FFT:PEAK:MAGNitude" on page 634
- ":FUNCtion<F>:FFT:PEAK:STATe" on page 635

History New in version 11.15.

:FUNCtion<F>:FFT:PEAK:MAGNitude

Query

```
:FUNCtion<F>:FFT:PEAK:MAGNitude?
```

The :FUNCtion<F>:FFT:PEAK:MAGNitude? query returns a comma-separated string of annotated peak magnitude values.

Values are returned according to the order specified by the :FUNCtion:FFT:PEAK:SORT command.

If no peaks are found, 9.99e37 is returned.

<F> An integer, 1-16, representing the selected function.

Returned Format

```
<string><NL>

<string> ::= string of comma-separated magnitude values
```

See Also

- ":FUNCtion:FFT:PEAK:SORT" on page 630
- ":FUNCtion<F>:FFT:PEAK:COUNt" on page 631
- ":FUNCtion<F>:FFT:PEAK:FREQuency" on page 632
- ":FUNCtion<F>:FFT:PEAK:LEVel" on page 633
- ":FUNCtion<F>:FFT:PEAK:STATe" on page 635

History New in version 11.15.

:FUNCtion<F>:FFT:PEAK:STATe

Command

```
:FUNCtion<F>:FFT:PEAK:STATe {{0 | OFF} | {1 | ON}}
```

The :FUNCtion<F>:FFT:PEAK:STATe command enables or disables FFT peak annotations.

When enabled, the first N peaks in the FFT above the specified Peak Level are annotated. N is specified by the :FUNCtion<F>:FFT:PEAK:COUNt command. The Peak level is specified by the :FUNCtion<F>:FFT:PEAK:LEVel command.

The annotated peak values are displayed in the graphical user interface's FFT Peaks results window at the bottom of the display.

You can get the frequency values of the annotated peaks using the :FUNCtion<F>:FFT:PEAK:FREQuency? query. You can get the magnitude values of the annotated peaks using the :FUNCtion<F>:FFT:PEAK:MAGNitude? query.

<F> An integer, 1-16, representing the selected function.

Query

```
:FUNCtion<F>:FFT:PEAK:STATe?
```

The :FUNCtion<F>:FFT:PEAK:STATe? query returns whether the FFT annotated peaks feature is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":FUNCtion:FFT:PEAK:SORT" on page 630
- ":FUNCtion<F>:FFT:PEAK:COUNt" on page 631
- ":FUNCtion<F>:FFT:PEAK:FREQuency" on page 632
- ":FUNCtion<F>:FFT:PEAK:LEVel" on page 633
- ":FUNCtion<F>:FFT:PEAK:MAGNitude" on page 634

History New in version 11.15.

:FUNCtion<F>:FFT:REFerence

Command

```
:FUNCtion<F>:FFT:REFerence {DISPlay | TRIGger}
```

The :FUNCtion<F>:FFT:REFerence command sets the reference point for calculating the FFT phase function.

<F> An integer, 1-16, representing the selected function.

Example This example sets the reference point to DISPlay.

```
myScope.WriteString ":FUNCtion1:FFT:REFerence DISPlay"
```

Query

```
:FUNCtion<F>:FFT:REFerence?
```

The :FUNCtion<F>:FFT:REFerence? query returns the currently selected reference point for the FFT phase function.

Returned Format

```
[:FUNCtion<F>:FFT:REFerence] {DISPlay | TRIGger}<NL>
```

Example This example places the current state of the function 1 FFT reference point in the string variable, strREF, then prints the contents of the variable to the computer's screen.

```
Dim strREF As String
myScope.WriteString ":FUNCtion1:FFT:REFerence?"
strREF = myScope.ReadString
Debug.Print strREF
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFT:RESolution

Command

```
:FUNCtion<F>:FFT:RESolution <resolution_value>
```

The :FUNCtion<F>:FFT:RESolution command sets the resolution bandwidth of the FFT function.

If either the memory depth or sampling rate is set to AUTO (see :ACQuire:POINts or :ACQuire:SRATe), you can adjust this control. However, if both the memory depth and sampling rate are in manual mode, you cannot set the resolution and can only query it.

The change in resolution bandwidth is achieved by changing the horizontal scale (as with the :TIMebase:SCALe command). Changes to the horizontal scale will also change the resolution bandwidth.

<F> An integer, 1-16, representing the selected function.

<resolution _value> Resolution bandwidth frequency.

The FFT resolution is defined as sampling rate / memory depth when using the Rectangular window (other windows have a Normalized Equivalent Noise Bandwidth factor applied).

$$\text{FFT Resolution} = \frac{\text{Sample Rate}}{\text{Effective Memory Depth}}$$

The effective memory depth is the highest power of 2 less than or equal to the number of sample points across the display. The memory bar in the status area at the top of the display indicates how much of the actual memory depth is across the display.

Query

```
:FUNCtion<F>:FFT:RESolution?
```

The :FUNCtion<F>:FFT:RESolution? query returns the current resolution of the FFT function.

Returned Format

```
[FUNCtion<F>:FFT:RESolution] <resolution_value><NL>
```

See Also

- ":ACQuire:POINts[:ANALog] — Memory depth" on page 310
- ":ACQuire:SRATe[:ANALog] — Analog Sample Rate" on page 323
- ":FUNCtion<F>:FFT:STOP" on page 640
- ":FUNCtion<F>:FFT:FREQuency" on page 626
- ":FUNCtion<F>:FFT:SPAN" on page 639
- ":TIMebase:SCALe" on page 1549

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

Version 5.70: The command form now lets you set the FFT resolution bandwidth.

:FUNCtion<F>:FFT:SPAN

Command

```
:FUNCtion<F>:FFT:SPAN <frequency_span>
```

The :FUNCtion<F>:FFT:SPAN command sets the frequency span for the FFT function.

<F> An integer, 1-16, representing the selected function.

<frequency_span> Frequency value in NR3 format.

Query

```
:FUNCtion<F>:FFT:SPAN?
```

The :FUNCtion<F>:FFT:SPAN? query returns the frequency span setting.

Returned Format

```
[FUNCtion<F>:FFT:SPAN] <frequency_span><NL>
```

See Also

- ":FUNCtion<F>:FFT:STOP" on page 640
- ":FUNCtion<F>:FFT:FREQuency" on page 626
- ":FUNCtion<F>:FFT:RESolution" on page 637

History New in version 5.70.

:FUNCtion<F>:FFT:STOP

Command

```
:FUNCtion<F>:FFT:STOP <stop_frequency>
```

The :FUNCtion<F>:FFT:STOP command sets the stop frequency for the FFT function.

<F> An integer, 1-16, representing the selected function.

<stop_frequency> Frequency value in NR3 format.

Query

```
:FUNCtion<F>:FFT:STOP?
```

The :FUNCtion<F>:FFT:STOP? query returns the stop frequency setting.

Returned Format

```
[FUNCtion<F>:FFT:STOP] <stop_frequency><NL>
```

See Also

- ":FUNCtion<F>:FFT:FREQuency" on page 626
- ":FUNCtion<F>:FFT:SPAN" on page 639
- ":FUNCtion<F>:FFT:RESolution" on page 637

History New in version 5.70.

:FUNCtion<F>:FFT:TDELay

Command

```
:FUNCtion<F>:FFT:TDELay <time_delay>
```

The :FUNCtion<F>:FFT:TDELay command sets the time delay for the FFT phase function.

<time_delay> Time, in seconds, set for the time delay.

Example This example sets the time delay to one millisecond.

```
myScope.WriteString ":FUNCtion1:FFT:TDELay 1E-3"
```

Query

```
:FUNCtion<F>:FFT:TDELay?
```

The :FUNCtion<F>:FFT:TDELay? query returns the time delay for the FFT phase function.

Returned Format

```
[:FUNCtion<F>:FFT:TDELay] <time_delay><NL>
```

Example This example places the FFT phase function's time delay value in the variable, varFftPhaseTimeDelay, then prints the contents of the variable to the computer's screen.

```
Dim varFftPhaseTimeDelay As Variant
myScope.WriteString ":FUNCtion1:FFT:TDELay?""
varFftPhaseTimeDelay = myScope.ReadNumber
Debug.Print FormatNumber(varFftPhaseTimeDelay, 0)
```

See Also

- ":FUNCtion<F>:FFTPhase" on page 646

History New in version 4.20.

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFT:VUNits

Command

```
:FUNCtion<F>:FFT:VUNits <units>
```

```
<units> ::= {DB | DBMV | DBUV | WATT | VRMS | DBMHZ | WATTHZ | VRTHZ}
```

The :FUNCtion<F>:FFT:VUNits command specifies the vertical units for the FFT Magnitude or Power Spectral Density math functions:

- DB — dBm, valid for the FFT Magnitude math function.
- DBMV — dBmV, valid for the FFT Magnitude math function.
- DBUV — dbuV, valid for the FFT Magnitude math function.
- WATT — Watt, valid for the FFT Magnitude math function.
- VRMS — Vrms, valid for the FFT Magnitude math function.
- DBMHZ — dBm/Hz, valid for the Power Spectral Density math function.
- WATTHZ — Watt/Hz, valid for the Power Spectral Density math function.
- VRTHZ — V/√Hz, valid for the Power Spectral Density math function. This is the square root of Power Spectral Density, also known as Amplitude Spectral Density (ASD)

Query

```
:FUNCtion<F>:FFT:VUNits?
```

The :FUNCtion<F>:FFT:VUNits? query returns the FFT magnitude function vertical units setting.

Returned Format

```
<units><NL>
```

See Also

- ":FUNCtion<F>:FFTMagnitude" on page 645
- ":FUNCtion<F>:PSDensity" on page 675
- ":FUNCtion<F>:FFT:DETector:POINts" on page 624
- ":FUNCtion<F>:FFT:DETector:TYPE" on page 625

History

New in version 5.70.

Version 11.30: Added the DBMHZ, WATTHZ, and VRTHZ units for the Power Spectral Density math function.

:FUNCtion<F>:FFT:WINDow

Command

```
:FUNCtion<F>:FFT:WINDow {RECTangular | HANNing | FLATtop
                        | BHARris | HAMMing}
```

The :FUNCtion<F>:FFT:WINDow command sets the window type for the FFT function.

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

- RECTangular — is essentially no window, and all points are multiplied by 1. This window is useful for transient waveforms and waveforms where there are an integral number of cycles in the time record.
- HANNing — is useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements.
- FLATtop — is best for making accurate amplitude measurements of frequency peaks.
- BHARris — (Blackman-Harris) is best used when you want to looks at signals with a strong interference component that is fairly distant from the frequency you want to see. It can be used as a general purpose window as its main lobe is not too wide (decent frequency discrimination) and the side lobes drop off by 90 dB.
- HAMMing —is a "raised cosine" function like the HANNing window but with different coefficients. It has slightly better frequency resolution than the HANNing window.

<F> An integer, 1-16, representing the selected function. This command presently selects all functions, regardless of which integer (1-16) is passed.

Example This example sets the window type for the FFT function to RECTangular.

```
myScope.WriteString ":FUNCtion1:FFT:WINDow RECTangular"
```

Query

```
:FUNCtion<F>:FFT:WINDow?
```

The :FUNCtion<F>:FFT:WINDow? query returns the current selected window for the FFT function.

Returned Format

```
[:FUNCtion<F>:FFT:WINDow] {RECTangular | HANNing | FLATtop
                          | BHARris | HAMMing}<NL>
```

Example This example places the current state of the function 1 FFT window in the string variable, strWND, then prints the contents of the variable to the computer's screen.

```
Dim strWND As String
myScope.WriteString ":FUNCtion1:FFT:WINDow?""
strWND = myScope.ReadString
Debug.Print strWND
```

History Legacy command (existed before version 3.10).

Version 3.11: Added the HAMMing window mode selection.

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFTMagnitude

Command

```
:FUNCtion<F>:FFTMagnitude <operand>
```

The :FUNCtion<F>:FFTMagnitude command computes the Fast Fourier Transform (FFT) of the specified channel, function, or memory. The FFT takes the digitized time record and transforms it to magnitude and phase components as a function of frequency.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to compute the FFT of waveform memory 3.

```
myScope.WriteString ":FUNCtion1:FFTMagnitude WMEMory3"
```

See Also

- ":FUNCtion<F>:FFT:VUNits" on page 642
- ":FUNCtion<F>:FFT:DETector:TYPE" on page 625
- ":FUNCtion<F>:FFT:DETector:POINts" on page 624
- ":FUNCtion<F>:FFT:IMPedance" on page 628
- ":FUNCtion<F>:PSDensity" on page 675

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:FFTPhase

Command

```
:FUNCtion<F>:FFTPhase <source>
```

The :FUNCtion<F>:FFTPhase command computes the Fast Fourier Transform (FFT) of the specified channel, function, or waveform memory. The FFT takes the digitized time record and transforms it into magnitude and phase components as a function of frequency.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to compute the FFT of waveform memory 3.

```
myScope.WriteString ":FUNCtion1:FFTPhase WMEMory3"
```

See Also

• ":FUNCtion<F>:FFT:TDELay" on page 641

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:GATing — Gating

Command

```
:FUNCtion<F>:GATing <operand>[,<gating_start>,<gating_stop>]
```

The :FUNCtion<F>:GATing command defines a horizontal gating function of another waveform (similar to horizontal zoom). Measurements on horizontal gating functions are essentially gated measurements.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<gating_start> Time, in seconds, relative to the source waveform that specifies where the gating window begins.

<gating_stop> Time, in seconds, relative to the source waveform that specifies where the gating window ends.

Example This example sets function 4 to be a horizontal gating of the channel1 waveform beginning at -8 ns and ending at -5 ns.

```
myScope.WriteString ":FUNCtion4:GATing CHANnel1,-8E-9,-5E-9"
```

See Also

- ":FUNCtion<F>:GATing:STARt — Gating window start time" on page 649
- ":FUNCtion<F>:GATing:STOP — Gating window stop time" on page 650

History New in version 4.30.

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:GATing:GLOBal

Command

```
:FUNCtion<F>:GATing:GLOBal <state>[, {GG1 | GG2 | GG3 | GG4}]
```

The :FUNCtion<F>:GATing:GLOBal command enables or disables one of the four global gates for the gating function.

<state> {{OFF | 0} | {ON | 1}}

See Also

- ":FUNCtion<F>:GATing — Gating" on page 647
- ":FUNCtion<F>:GATing:STARt — Gating window start time" on page 649
- ":FUNCtion<F>:GATing:STOP — Gating window stop time" on page 650

History New in version 10.20.

:FUNCtion<F>:GATing:STARt — Gating window start time

Command

```
:FUNCtion<F>:GATing:STARt <gating_start>
```

The :FUNCtion<F>:GATing:STARt command specifies the time, in seconds, where the gating window begins relative to the source waveform (see :FUNCtion<F>:GATing).

<F> An integer, 1-16, representing the selected function.

Example This example sets a -8 ns gating window begin time for function 4.

```
myScope.WriteString ":FUNCtion4:GATing:STARt -8E-9"
```

The gating window is applied to the source operand specified in the :FUNCtion4:GATing command.

Query

```
:FUNCtion<F>:GATing:STARt?
```

The ::FUNCtion<F>:GATing:STARt? query returns the gating window start time.

Returned Format

```
[:FUNCtion<F>:GATing:STARt] <gating_start><NL>
```

See Also

- ":FUNCtion<F>:GATing — Gating" on page 647
- ":FUNCtion<F>:GATing:STOP — Gating window stop time" on page 650

History New in version 5.30.

:FUNCtion<F>:GATing:STOP — Gating window stop time

Command

```
:FUNCtion<F>:GATing:STOP <gating_stop>
```

The :FUNCtion<F>:GATing:STOP command specifies the time, in seconds, where the gating window ends relative to the source waveform (see :FUNCtion<F>:GATing).

<F> An integer, 1-16, representing the selected function.

Example This example sets a -5 ns gating window end time for function 4.

```
myScope.WriteString ":FUNCtion4:GATing:STOP -5E-9""
```

The gating window is applied to the source operand specified in the :FUNCtion4:GATing command.

Query

```
:FUNCtion<F>:GATing:STOP?
```

The ::FUNCtion<F>:GATing:STOP? query returns the gating window stop time.

Returned Format

```
[:FUNCtion<F>:GATing:STOP] <gating_stop><NL>
```

See Also

- ":FUNCtion<F>:GATing — Gating" on page 647
- ":FUNCtion<F>:GATing:STARt — Gating window start time" on page 649

History New in version 5.30.

:FUNCtion<F>:HIGHpass

Command

```
:FUNCtion<F>:HIGHpass <source>,<bandwidth>
```

The :FUNCtion<F>:HIGHpass command applies a single-pole high pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<bandwidth> A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a high pass filter with a bandwidth of 1 MHz.

```
myScope.WriteString ":FUNCtion2:HIGHpass CHANnel4,1E6"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:HORizontal

Command

```
:FUNCtion<F>:HORizontal {AUTO | MANual}
```

The :FUNCtion<F>:HORizontal command sets the horizontal tracking to either AUTO or MANual.

NOTE Using the :FUNCtion<F>:HORizontal:RANGe or :FUNCtion<F>:HORizontal:POSition commands automatically changes the :FUNCtion<F>:HORizontal setting to MANual.

<F> An integer, 1-16, representing the selected function.

Query

```
:FUNCtion<F>:HORizontal?
```

The :FUNCtion<F>:HORizontal? query returns the current horizontal scaling mode of the specified function.

Returned Format

```
[:FUNCtion<F>:HORizontal] {AUTO | MANual}<NL>
```

Example This example places the current state of the function 1 horizontal tracking in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":FUNCtion1:HORizontal?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":FUNCtion<F>:HORizontal:POSition" on page 653
- ":FUNCtion<F>:HORizontal:RANGe" on page 655

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:HORizontal:POSition

Command

```
:FUNCtion<F>:HORizontal:POSition <position_value>
```

The :FUNCtion<F>:HORizontal:POSition command sets the time value at center screen for the selected function.

Using the :FUNCtion<F>:HORizontal:POSition command automatically changes the :FUNCtion<F>:HORizontal setting to MANual.

When you select :FUNCtion<F>:FFTMagnitude, the horizontal position is equivalent to the center frequency. This also automatically selects manual mode.

NOTE

For some math functions (:FUNCtion<F>:ABSolute, :FUNCtion<F>:ADD, :FUNCtion<F>:ADEMod, :FUNCtion<F>:COMMonmode, :FUNCtion<F>:DIFF, :FUNCtion<F>:DIVide, :FUNCtion<F>:HIGHpass, :FUNCtion<F>:INTegrate, :FUNCtion<F>:INVert, :FUNCtion<F>:LOWPass, :FUNCtion<F>:MAXimum, :FUNCtion<F>:MTRend, :FUNCtion<F>:MINimum, :FUNCtion<F>:MULTiply, :FUNCtion<F>:SQUare, :FUNCtion<F>:SQRT, :FUNCtion<F>:SUBTract), the waveform's horizontal scaling is tied to the timebase of the source channel waveform(s).

The :FUNCtion<F>:HORizontal:RANGe and :FUNCtion<F>:HORizontal:POSition commands for these functions give a -221,"Settings conflict" error. Instead, you must use the :TIMebase:RANGe and :TIMebase:POSition commands to make horizontal scale and position changes.

<F> An integer, 1-16, representing the selected function.

<position _value> A real number for the position value in time, in seconds.

Query

```
:FUNCtion<F>:HORizontal:POSition?
```

The :FUNCtion<F>:HORizontal:POSition? query returns the current time value at center screen of the selected function.

Returned Format

```
[:FUNCtion<F>:HORizontal:POSition] <position><NL>
```

Example This example places the current horizontal position setting for function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":FUNCtion2:HORizontal:POSition?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":FUNCtion<F>:HORizontal:RANGe" on page 655
- ":FUNCtion<F>:HORizontal" on page 652
- ":TIMebase:POSition" on page 1542

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

Version 6.00: For functions where the horizontal position cannot be adjusted, this command now gives a -221,"Settings conflict" instead of being accepted without effect.

:FUNCtion<F>:HORizontal:RANGe

Command

```
:FUNCtion<F>:HORizontal:RANGe <range_value>
```

The :FUNCtion<F>:HORizontal:RANGe command sets the current time range for the specified function.

Using the :FUNCtion<F>:HORizontal:RANGe command automatically changes the :FUNCtion<F>:HORizontal setting to MANual.

NOTE

For some math functions (:FUNCtion<F>:ABSolute, :FUNCtion<F>:ADD, :FUNCtion<F>:ADEMod, :FUNCtion<F>:COMMonmode, :FUNCtion<F>:DIFF, :FUNCtion<F>:DIVide, :FUNCtion<F>:HIGHpass, :FUNCtion<F>:INTegrate, :FUNCtion<F>:INVert, :FUNCtion<F>:LOWPass, :FUNCtion<F>:MAXimum, :FUNCtion<F>:MTRend, :FUNCtion<F>:MINimum, :FUNCtion<F>:MULTiply, :FUNCtion<F>:SQUare, :FUNCtion<F>:SQRT, :FUNCtion<F>:SUBTract), the waveform's horizontal scaling is tied to the timebase of the source channel waveform(s).

The :FUNCtion<F>:HORizontal:RANGe and :FUNCtion<F>:HORizontal:POSition commands for these functions give a -221,"Settings conflict" error. Instead, you must use the :TIMebase:RANGe and :TIMebase:POSition commands to make horizontal scale and position changes.

<F> An integer, 1-16, representing the selected function.

<range_value> A real number for the width of screen in current X-axis units (usually seconds).

Query

```
:FUNCtion<F>:HORizontal:RANGe?
```

The :FUNCtion<F>:HORizontal:RANGe? query returns the current time range setting of the specified function.

Returned Format

```
[:FUNCtion<F>:HORizontal:RANGe] <range><NL>
```

Example This example places the current horizontal range setting of function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":FUNCtion2:HORizontal:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":FUNCtion<F>:HORizontal:POSition" on page 653
- ":FUNCtion<F>:HORizontal" on page 652
- ":TIMebase:RANGe" on page 1543

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

Version 6.00: For functions where the horizontal position cannot be adjusted, this command now gives a -221,"Settings conflict" instead of being accepted without effect.

:FUNCtion<F>:INTegrate

Command

```
:FUNCtion<F>:INTegrate <operand>
```

The :FUNCtion<F>:INTegrate command defines a function that computes the integral of the specified operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 1 to compute the integral of waveform memory 3.

```
myScope.WriteString ":FUNCtion1:INTegrate WMEMory3"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:INVert

Command

```
:FUNCtion<F>:INVert <operand>
```

The :FUNCtion<F>:INVert command defines a function that inverts the defined operand's waveform by multiplying by -1.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

Example This example sets up function 2 to invert the waveform on channel 1.

```
myScope.WriteString ":FUNCtion2:INVert CHANnel1"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:LABel

Command

```
:FUNCtion<F>:LABel <quoted_string>
```

The :FUNCtion<F>:LABel command sets the math function waveform label to the quoted string.

Labels can be enabled with the :DISPlay:LABel command.

<F> An integer, 1-16, representing the selected function.

<quoted_string> A series of 16 or fewer characters as a quoted ASCII string.

Query

```
:FUNCtion<F>:LABel?
```

The :FUNCtion<F>:LABel? query returns the label of the specified math function waveform.

Returned Format

```
[:FUNCtion<F>:LABel] <quoted_string><NL>
```

See Also

- ":DISPlay:LABel" on page 584
- ":CHANnel<N>:LABel" on page 467
- ":WMEMory<R>:LABel" on page 1772

History New in version 11.15.

:FUNCtion<F>:LOWPass

Command

```
:FUNCtion<F>:LOWPass <source>,<bandwidth>
```

The :FUNCtion<F>:LOWPass command applies a 4th order Bessel-Thompson low pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

<bandwidth> A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a low pass filter with a bandwidth of 1 MHz.

```
myScope.WriteString ":FUNCtion2:LOWPass CHANnel4,1E6"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MAGNify

Command

```
:FUNCtion<F>:MAGNify <operand>
```

The :FUNCtion<F>:MAGNify command defines a function that is a copy of the operand. The magnify function is a software magnify. No hardware settings are altered as a result of using this function. It is useful for scaling channels, another function, or memories with the RANGe and OFFSet commands in this subsystem.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example creates a function (function 1) that is a magnified version of channel 1.

```
myScope.WriteString ":FUNCtion1:MAGNify CHANnel1"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MATLab

Command

```
:FUNCtion<F>:MATLab <operand>[,<operand>]
```

The :FUNCtion<F>:MATLab command sets the operand(s) for these user-defined functions:

- Butterworth
- FIR
- LFE
- RTEye
- SqrtSumOfSquare

And these InfiniiSim functions:

- InfiniiSim 2 Port
- InfiniiSim 4 Port 1 Src
- InfiniiSim 4 Port CM
- InfiniiSim 4 Port Diff
- InfiniiSim 4 Port Src1
- InfiniiSim 4 Port Src2

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<R> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets the "InfiniiSim 2 Port" math function, operands, and controls.

```
myScope.WriteString ":FUNCtion1:MATLab:OPERator 'InfiniiSim 2 Port'"
myScope.WriteString ":FUNCtion1:MATLab CHANnel1"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol1 'c:\users\public\
documents\infiniium\filters\cable only.tf2'"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol2 5e-9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol3 10e9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol4 2"
```

See Also

- ":FUNCtion<F>:MATLab:OPERator" on page 665
- ":FUNCtion<F>:MATLab:CONTrol<N>" on page 663

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MATLab:CONTrol<N>

Command

```
:FUNCtion<F>:MATLab:CONTrol<N> {<value> | <string>}
```

The :FUNCtion<F>:MATLab:CONTrol<N> command sets control values for these user-defined functions:

- Butterworth
- FIR
- LFE
- RTEye
- SqrtSumOfSquare

And these InfiniiSim functions:

- InfiniiSim 2 Port
- InfiniiSim 4 Port 1 Src
- InfiniiSim 4 Port CM
- InfiniiSim 4 Port Diff
- InfiniiSim 4 Port Src1
- InfiniiSim 4 Port Src2

<F> An integer, 1-16, representing the selected function.

<N> An integer, 1-6, representing the user-defined or InfiniiSim function control.

<value> A double, integer, or enumerated type value. For an enumerated type, the 1 based index is passed to select the enumeration.

<string> A character array.

Example This example sets the "InfiniiSim 2 Port" math function, operands, and controls.

```
myScope.WriteString ":FUNCtion1:MATLab:OPERator 'InfiniiSim 2 Port'"
myScope.WriteString ":FUNCtion1:MATLab CHANnel1"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol1 'c:\users\public\
documents\infiniium\filters\cable only.tf2'"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol2 5e-9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol3 10e9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol4 2"
```

Query

```
:FUNCtion<F>:MATLab:CONTrol<N>?
```

The :FUNCtion<F>:MATLab:CONTrol<N>? query returns the value or string of the user-defined control.

Returned Format

```
[:FUNCtion<F>:MATLab:CONTrol<N>] {<value> | <string>}<NL>
```

Example This example places the current returned value for function 1 control 1 in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":FUNCtion1:MATLab:CONTrol1?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

See Also

- ":FUNCtion<F>:MATLab:OPERator" on page 665
- ":FUNCtion<F>:MATLab" on page 662

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

Version 5.60: Up to 6 user-defined controls supported.

:FUNCtion<F>:MATLab:OPERator

Command

```
:FUNCtion<F>:MATLab:OPERator <string>
```

The :FUNCtion<F>:MATLab:OPERator command sets the Function dialog box operator. Any math function operator name can be specified, not just user-defined or InfiniiSim math functions.

<F> An integer, 1-16, representing the selected function.

<string> A character array that is the name of the math function as it appears in the Function dialog box.

Example This example sets the "InfiniiSim 2 Port" math function, operands, and controls.

```
myScope.WriteString ":FUNCtion1:MATLab:OPERator 'InfiniiSim 2 Port'"
myScope.WriteString ":FUNCtion1:MATLab CHANnel1"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol1 'c:\users\public\
documents\infiniium\filters\cable only.tf2'"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol2 5e-9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol3 10e9"
myScope.WriteString ":FUNCtion1:MATLab:CONTrol4 2"
```

Query

```
:FUNCtion<F>:MATLab:OPERator?
```

The :FUNCtion<F>:MATLab:OPERator? query returns the string of the math function operator.

Returned Format

```
[:FUNCtion<F>:MATLab:OPERator] <string><NL>
```

Example This example places the current operator string for function 1 in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":FUNCtion1:MATLab:OPERator?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

See Also

- ":FUNCtion<F>:MATLab" on page 662
- ":FUNCtion<F>:MATLab:CONTrol<N>" on page 663

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MAXimum

Command

```
:FUNCtion<F>:MAXimum <operand>
```

The :FUNCtion<F>:MAXmum command defines a function that computes the maximum of each time bucket for the defined operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example sets up function 2 to compute the maximum of each time bucket for channel 4.

```
myScope.WriteString ":FUNCtion2:MAXimum CHANnel4"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MHIStogram

Command

```
:FUNCtion<F>:MHIStogram {MEAS1 | MEAS2 | MEAS3 | ...
    | MEAS20}[, <max_bins>[, {MIN | <min>}[, {MAX | <max>}]]]
```

The :FUNCtion<F>:MHIStogram command adds a Meas Histogram function that shows a histogram of measurement values. Measurement values are captured and the histogram is updated as new acquisitions are made.

You can display statistics for the histogram in the Measurements tab using the :MEASure:HISTogram commands and you can get histogram statistics using the :MEASure:HISTogram queries.

<F> An integer, 1-16, representing the selected function.

<max_bins> An integer from 10-1280.

{MIN | <min>}, {MAX | <max>} You can specify the histogram's measurement minimum and measurement maximum bounds with <min> and <max> floating-point values, or if you want the histogram bounds to be automatically determined, use MIN and MAX.

Example This example sets up a histogram function of the first measurement.

```
myScope.WriteString ":FUNCtion1:MHIStogram MEAS6,1280,-20E-12,20E-12"
```

See Also

- ":MEASure:HISTogram:HITS" on page 950
- ":MEASure:HISTogram:M1S" on page 951
- ":MEASure:HISTogram:M2S" on page 952
- ":MEASure:HISTogram:M3S" on page 953
- ":MEASure:HISTogram:MAX" on page 954
- ":MEASure:HISTogram:MEAN" on page 955
- ":MEASure:HISTogram:MEDian" on page 956
- ":MEASure:HISTogram:MIN" on page 957
- ":MEASure:HISTogram:MODE" on page 959
- ":MEASure:HISTogram:PEAK" on page 961
- ":MEASure:HISTogram:PP" on page 962
- ":MEASure:HISTogram:RESolution" on page 963
- ":MEASure:HISTogram:STDDev" on page 964

History New in version 3.50.

Version 4.30: Up to 16 functions supported.

Version 5.00: Now 20 measurements to choose from.

Version 5.20: Lets you specify the maximum number of histogram bins along with the measurement source.

Version 10.10: Added <min> and <max> parameters for specifying the histogram's measurement minimum and measurement maximum.

:FUNCtion<F>:MINimum

Command

```
:FUNCtion<F>:MINimum <operand>
```

The :FUNCtion<F>:MINimum command defines a function that computes the minimum of each time bucket for the defined operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

Example This example sets up function 2 to compute the minimum of each time bucket for channel 4.

```
myScope.WriteString ":FUNCtion2:MINimum CHANnel4"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:MLOG

Command

```
:FUNCtion<F>:MLOG {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20}
```

The :FUNCtion<F>:MLOG command adds a function waveform that is a scrolling record of measurement values over time.

<F> An integer, 1-16, representing the selected function.

See Also

- ":FUNCtion<F>:MTRend" on page 671

History New in version 6.00.

:FUNCtion<F>:MTRend

Command

```
:FUNCtion<F>:MTRend {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20}
```

The :FUNCtion<F>:MTRend command adds a Meas Trend function that shows measurement values for a waveform (based on measurement threshold settings) as the waveform progresses across the screen. For every cycle, a measurement is made, and the value is displayed on the screen for the cycle.

If a measurement cannot be made for part of a waveform, the trend function output is a hole (that is, no value) until a measurement can be made.

<F> An integer, 1-16, representing the selected function.

Example This example sets up a trend function of the first measurement.

```
myScope.WriteString ":FUNCtion2:MTRend MEAS1"
```

History New in version 3.50.

Version 4.30: Up to 16 functions supported.

Version 5.00: Now 20 measurements to choose from.

:FUNCtion<F>:MULTiply

Command

```
:FUNCtion<F>:MULTiply <operand>,<operand>
```

The :FUNCtion<F>:MULTiply command defines a function that algebraically multiplies the first operand by the second operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example defines a function that multiplies channel 1 by waveform memory 1.

```
myScope.WriteString ":FUNCtion1:MULTiply CHANnel1,WMEMory1"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:OFFSet

Command

```
:FUNCtion<F>:OFFSet <offset_value>
```

The :FUNCtion<F>:OFFSet command sets the voltage represented at the center of the screen for the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<offset_value> A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited to being within the vertical range that can be represented by the function data.

Example This example sets the offset voltage for function 1 to 2 mV.

```
myScope.WriteString ":FUNCtion1:OFFSet 2E-3"
```

Query

```
:FUNCtion<F>:OFFSet?
```

The :FUNCtion<F>:OFFSet? query returns the current offset value for the selected function.

Returned Format

```
[:FUNCtion<F>:OFFSet] <offset_value><NL>
```

Example This example places the current setting for offset on function 2 in the numeric variable, varValue, then prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":FUNCtion2:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:PAVerage

Command

```
:FUNCtion<F>:PAVerage <source>[, <num_averages>[, <pts_per_UI>]]
```

The :FUNCtion<F>:PAVerage command sets up the Pattern Average math function.

From a detected bit pattern, the Pattern Average math function removes random jitter and noise and preserves inter-symbol interference and data dependent jitter and noise.

The Pattern Average math function requires clock recovery and at least two error-free copies of an identical repeating bit pattern in acquisition memory.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

<num_averages> This option specifies the number of averages of the detected bit pattern output. You can specify an integer from 2 to 65534.

<pts_per_UI> This option specifies the points per unit interval to use from the input waveform source data. You can specify an integer from 8 to 1024.

See Also

- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":FUNCtion<F>:DISPlay" on page 622

History New in version 6.10.

:FUNCtion<F>:PSDensity

Command

```
:FUNCtion<F>:PSDensity <operand>
```

The :FUNCtion<F>:PSDensity command computes the Power Spectral Density (PSD) of the specified channel, function, or memory.

The Power Spectral Density (PSD) math function shows the power density of a signal as a function of frequency.

The Power Spectral Density math function is similar to the FFT Magnitude math function (see :FUNCtion<F>:FFTMagnitude) except that the spectral power per frequency (dBm/Hz or Watt/Hz) is plotted instead of just the spectral power (dBm or Watt).

By plotting spectral power per frequency (power density), the Power Spectral Density math function is scaled to correctly show the noise power at each frequency. In FFT Magnitude math functions, the noise power depends on the FFT bin width, as noise for the full frequency range of the FFT bin is reflected. While the FFT Magnitude math function is useful for looking at spectral power magnitudes, the Power Spectral Density math function is useful for looking at noise and noise floors.

When the V/√Hz vertical units are selected for the Power Spectral Density math function (see :FUNCtion<F>:FFT:VUNits), the plot shows the square root of the Power Spectral Density, which is known as the Amplitude Spectral Density (ASD).

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, “:FUNCtion<F> Commands,” starting on page 611.

See Also

- ":FUNCtion<F>:FFT:VUNits" on page 642
- ":FUNCtion<F>:FFT:IMPedance" on page 628
- ":FUNCtion<F>:FFTMagnitude" on page 645

History New in version 11.30.

:FUNCtion<F>:RANGe

Command

```
:FUNCtion<F>:RANGe <full_scale_range>
```

The :FUNCtion<F>:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<full_scale_range> A real number for the full-scale vertical range, from -100E15 to 100E15.

Example This example sets the full-scale range for function 1 to 400 mV.

```
myScope.WriteString ":FUNCtion1:RANGe 400E-3"
```

Query

```
:FUNCtion<F>:RANGe?
```

The :FUNCtion<F>:RANGe? query returns the current full-scale range setting for the specified function.

Returned Format

```
[:FUNCtion<F>:RANGe] <full_scale_range><NL>
```

Example This example places the current range setting for function 2 in the numeric variable "varValue", then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":FUNCtion2:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:SMOoth

Command

```
:FUNCtion<F>:SMOoth <operand>[,<points>]
```

The :FUNCtion<F>:SMOoth command defines a function that assigns the smoothing operator to the operand with the number of specified smoothing points.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

<points> An integer, odd numbers from 3 to 4001 specifying the number of smoothing points.

Example This example sets up function 1 using assigning smoothing operator to channel 1 using 5 smoothing points.

```
myScope.WriteString ":FUNCtion1:SMOoth CHANnel1,5"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:SQRT

Command

```
:FUNCtion<F>:SQRT <operand>
```

The :FUNCtion<F>:SQRT command takes the square root of the operand.

<operand> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<R> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example turns on the square root function using channel 3.

```
myScope.WriteString ":FUNCtion1:SQRT CHANnel3"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:SQUare

Command

```
:FUNCtion<F>:SQUare <operand>
```

The :FUNCtion<F>:SQUare command takes the square value of the operand.

<operand> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<R> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example turns on the square value command using channel 3.

```
myScope.WriteString ":FUNCtion1:SQUare CHANnel3"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:SUBTract

Command

```
:FUNCtion<F>:SUBTract <operand>,<operand>
```

The :FUNCtion<F>:SUBTract command defines a function that algebraically subtracts the second operand from the first operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example defines a function that subtracts waveform memory 1 from channel 1.

```
myScope.WriteString ":FUNCtion1:SUBTract CHANnel1,WMEMory1"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:VERSus

Command `:FUNCtion<F>:VERSus <operand>,<operand>`

The :FUNCtion<F>:VERSus command defines a function for an X-versus-Y display. The first operand defines the Y axis and the second defines the X axis. The Y-axis range and offset are initially equal to that of the first operand, and you can adjust them with the RANGe and OFFSet commands in this subsystem.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | EQUalized<L> | WMEMory<n> | <float_value> | MTRend | MSPectrum | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

See the discussion of possible operands in the introduction to Chapter 24, ":FUNCtion<F> Commands," starting on page 611.

Example This example defines function 1 as an X-versus-Y display. Channel 1 is the X axis and waveform memory 2 is the Y axis.

```
myScope.WriteString ":FUNCtion1:VERSus WMEMory2,CHANnel1"
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:VERTical

Command

```
:FUNCtion<F>:VERTical {AUTO | MANual}
```

The :FUNCtion<F>:VERTical command sets the vertical scaling mode of the specified function to either AUTO or MANual.

This command also contains the following commands and queries:

- OFFset
- RANge

<F> An integer, 1-16, representing the selected function.

Query

```
:FUNCtion<F>:VERTical?
```

The :FUNCtion<F>:VERTical? query returns the current vertical scaling mode of the specified function.

Returned Format

```
[:FUNCtion<F>:VERTical] {AUTO | MANual}<NL>
```

Example This example places the current state of the vertical tracking of function 1 in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":FUNCtion1:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:VERTical:OFFSet

Command

```
:FUNCtion<F>:VERTical:OFFSet <offset_value>
```

The :FUNCtion<F>:VERTical:OFFSet command sets the voltage represented at center screen for the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<offset_value> A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited only to being within the vertical range that can be represented by the function data.

Query

```
:FUNCtion<F>:VERTical:OFFset?
```

The :FUNCtion<F>:VERTical:OFFSet? query returns the current offset value of the selected function.

Returned Format

```
[:FUNCtion<F>:VERTical:OFFset] <offset_value><NL>
```

Example This example places the current offset setting for function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":FUNCtion2:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

:FUNCtion<F>:VERTical:RANGe

Command

```
:FUNCtion<F>:VERTical:RANGe <full_scale_range>
```

The :FUNCtion<F>:VERTical:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual, if the oscilloscope is not already in manual mode.

<F> An integer, 1-16, representing the selected function.

<full_scale_range> A real number for the full-scale vertical range, from -100E15 to 100E15.

Query

```
:FUNCtion<F>:VERTical:RANGe?
```

The :FUNCtion<F>:VERTical:RANGe? query returns the current range setting of the specified function.

Returned Format

```
[:FUNCtion<F>:VERTical:RANGe] <range><NL>
```

Example This example places the current vertical range setting of function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":FUNCtion2:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 4.30: Up to 16 functions supported.

25 :HARDcopy Commands

The HARDcopy subsystem commands set various parameters for printing the screen. The print sequence is activated when the root level command :PRINt is sent.

:HARDcopy:AREA

Command

```
:HARDcopy:AREA {GRATicule | SCReen}
```

The :HARDcopy:AREA command selects which data from the screen is to be printed. When you select GRATicule, only the graticule area of the screen is printed (this is the same as choosing Waveforms Only in the Configure Printer dialog box). When you select SCReen, the entire screen is printed.

Example This example selects the graticule for printing.

```
myScope.WriteString ":HARDcopy:AREA GRATicule"
```

Query

```
:HARDcopy:AREA?
```

The :HARDcopy:AREA? query returns the current setting for the area of the screen to be printed.

Returned Format

```
[:HARDcopy:AREA] {GRATicule | SCReen}<NL>
```

Example This example places the current selection for the area to be printed in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":HARDcopy:AREA?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

:HARDcopy:DPRinter

Command

```
:HARDcopy:DPRinter {<printer_number> | <printer_string>}
```

The :HARDcopy:DPRinter command selects the default printer to be used.

<printer _number> An integer representing the attached printer. This number corresponds to the number returned with each printer name by the :HARDcopy:PRINters? query.

<printer _string> A string of alphanumeric characters representing the attached printer.

The :HARDcopy:DPRinter command specifies a number or string for the printer attached to the oscilloscope. The printer string must exactly match the character strings in the File->Print Setup dialog boxes, or the strings returned by the :HARDcopy:PRINters? query.

Examples This example sets the default printer to the second installed printer returned by the :HARDcopy:PRINters? query.

```
myScope.WriteString ":HARDcopy:DPRinter 2"
```

This example sets the default printer to the installed printer with the name "HP Laser".

```
myScope.WriteString ":HARDcopy:DPRinter ""HP Laser"""
```

Query

```
:HARDcopy:DPRinter?
```

The :HARDcopy:DPRinter? query returns the current printer number and string.

Returned Format

```
[:HARDcopy:DPRinter?] {<printer_number>,<printer_string>,DEFAULT}<NL>
```

Or, if there is no default printer (no printers are installed), only a <NL> is returned.

Example This example places the current setting for the hard copy printer in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDcopy:DPRinter?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

NOTE **It takes several seconds to change the default printer. Any programs that try to set the default printer must wait (10 seconds is a safe amount of time) for the change to complete before sending other commands. Otherwise, the oscilloscope will become unresponsive.**

History Legacy command (existed before version 3.10).

:HARDcopy:FACTors

Command

```
:HARDcopy:FACTors {{ON | 1} | {OFF | 0}}
```

The :HARDcopy:FACTors command determines whether the oscilloscope setup factors will be appended to screen or graticule images. FACTors ON is the same as choosing Include Setup Information in the Configure Printer dialog box.

Example This example turns on the setup factors.

```
myScope.WriteString ":HARDcopy:FACTors ON"
```

Query

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns the current setup factors setting.

Returned Format

```
[:HARDcopy:FACTors] {1 | 0}<NL>
```

Example This example places the current setting for the setup factors in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDcopy:FACTors?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:HARDcopy:IMAGe

Command
```
:HARDcopy:IMAGe {NORMal | INVert}
```

The :HARDcopy:IMAGe command prints the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Colors in the Configure Printer dialog box.

Example This example sets the hard copy image output to normal.

```
myScope.WriteString ":HARDcopy:IMAGe NORMal"
```

Query
```
:HARDcopy:IMAGe?
```

The :HARDcopy:IMAGe? query returns the current image setting.

Returned Format
```
[:HARDcopy:IMAGe] {NORMal | INVert}<NL>
```

Example This example places the current setting for the hard copy image in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDcopy:IMAGe?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:HARDcopy:PRINters?

Query

```
:HARDcopy:PRINters?
```

The :HARDcopy:PRINters? query returns the currently available printers.

Returned Format

```
[:HARDcopy:PRINters?]
<printer_count><NL><printer_data><NL>[,<printer_data><NL>]
```

<printer_count> The number of printers currently installed.

<printer _data> The printer number and the name of an installed printer. The word DEFAULT appears next to the printer that is the currently selected default printer.

The <printer_data> return string has the following format:
<printer_number>,<printer_string>{,DEFAULT}

Example This example places the number of installed printers into the variable varCount, loops through it that number of times, and prints the installed printer names to the computer's screen.

```
Dim varResults As Variant
Dim lngI As Long

myScope.WriteString ":HARDcopy:PRINters?"
varResults = myScope.ReadList(ASCIIType_BSTR, vbLf)
Debug.Print FormatNumber(varResults(0), 0)

For lngI = 1 To varResults(0)
  Debug.Print CStr(varResults(lngI))
Next lngI
```

History Legacy command (existed before version 3.10).

26 :HISTogram Commands

The HISTogram commands and queries control the histogram features. A histogram is a probability distribution that shows the distribution of acquired data within a user-definable histogram window.

You can display the histogram either vertically, for voltage measurements, or horizontally, for timing measurements.

The most common use for histograms is measuring and characterizing noise or jitter on displayed waveforms. Noise is measured by sizing the histogram window to a narrow portion of time and observing a vertical histogram that measures the noise on a waveform. Jitter is measured by sizing the histogram window to a narrow portion of voltage and observing a horizontal histogram that measures the jitter on an edge.

Histograms and the database

The histograms, mask testing, and color grade persistence use a specific database that uses a different memory area from the waveform record for each channel. When any of these features are turned on, the oscilloscope starts building the database. The database is the size of the graticule area. Behind each pixel is a 21-bit counter that is incremented each time data from a channel or function hits a pixel. The maximum count (saturation) for each counter is 2,097,151. You can use the DISPlay:CGRade:LEVels command to see if any of the counters are close to saturation.

The database continues to build until the oscilloscope stops acquiring data or all both features (color grade persistence and histograms) are turned off. You can clear the database by turning off all three features that use the database.

The database does not differentiate waveforms from different channels or functions. If three channels are on and the waveform from each channel happens to light the same pixel at the same time, the counter is incremented by three. However, it is not possible to tell how many hits came from each waveform. To separate waveforms, you can position the waveforms vertically with the channel offset. By separating the waveforms, you can avoid overlapping data in the database caused by multiple waveforms. Even if the display is set to show only the most recent acquisition, the database keeps track of all pixel hits while the database is building.

Remember that color grade persistence, mask testing, and histograms all use the same database. Suppose that the database is building because color grade persistence is ON; when mask testing or histograms are turned on, they can use the information already established in the database as though they had been turned on the entire time.

To avoid erroneous data, clear the display after you change oscilloscope setup conditions or DUT conditions and acquire new data before extracting measurement results.

:HISTogram:AXIS

Command

```
:HISTogram:AXIS {VERTical | HORizontal}
```

The :HISTogram:AXIS command selects the type of histogram. A horizontal histogram can be used to measure time related information like jitter. A vertical histogram can be used to measure voltage related information like noise.

Example

This example defines a vertical histogram.

```
myScope.WriteString ":HISTogram:AXIS VERTical"
```

Query

```
:HISTogram:AXIS?
```

The :HISTogram:AXIS? query returns the currently selected histogram type.

Returned Format

```
[:HISTogram:AXIS] {VERTical | HORizontal}<NL>
```

Example

This example returns the histogram type and prints it to the computer's screen.

```
Dim strAxis As String
myScope.WriteString ":HISTogram:AXIS?"
strAxis = myScope.ReadString
Debug.Print strAxis
```

History

Legacy command (existed before version 3.10).

:HISTogram:HORizontal:BINS

Command

```
:HISTogram:HORizontal:BINS <max_bins>
```

```
<max_bins> ::= integer from 10-1280
```

The :HISTogram:HORizontal:BINS command sets the maximum number of bins used for a horizontal waveform histogram.

Query

```
:HISTogram:HORizontal:BINS?
```

The :HISTogram:HORizontal:BINS? query returns the maximum number of bins setting.

Returned Format

```
<max_bins><NL>
```

```
<max_bins> ::= integer from 10-1280
```

See Also

- ":HISTogram:MODE" on page 698
- ":HISTogram:AXIS" on page 693
- ":HISTogram:MEASurement:BINS" on page 695
- ":HISTogram:VERTical:BINS" on page 700

History New in version 5.20.

:HISTogram:MEASurement:BINS

Command

```
:HISTogram:MEASurement:BINS <max_bins>
```

```
<max_bins> ::= integer from 10-8000
```

The :HISTogram:MEASurement:BINS command sets the maximum number of bins used for a measurement histogram.

Query

```
:HISTogram:MEASurement:BINS?
```

The :HISTogram:MEASurement:BINS? query returns the maximum number of bins setting.

Returned Format

```
<max_bins><NL>
```

```
<max_bins> ::= integer from 10-8000
```

See Also

- ":HISTogram:MODE" on page 698
- ":HISTogram:AXIS" on page 693
- ":HISTogram:HORizontal:BINS" on page 694
- ":HISTogram:VERTical:BINS" on page 700

History New in version 5.20.

:HISTogram:MEASurement:MAX

Command

```
:HISTogram:MEASurement:MAX {MAX | <max>}
```

```
<max> ::= measurement maximum floating-point value
```

The :HISTogram:MEASurement:MAX command specifies the histogram's measurement maximum. This is the upper bound of the histogram.

You can specify a <max> floating-point value, or if you want the measurement maximum to be automatically determined, use MAX.

Query

```
:HISTogram:MEASurement:MAX?
```

The :HISTogram:MEASurement:MAX? query returns the specified measurement maximum value.

If MAX was specified, the value returned is 1.79769313486232E+308 (the highest 64-bit floating-point value).

Returned Format

```
<max><NL>
```

See Also

- ":HISTogram:MEASurement:MIN" on page 697

History

New in version 10.10.

:HISTogram:MEASurement:MIN

Command

```
:HISTogram:MEASurement:MIN {MIN | <min>}
```

```
<min> ::= measurement minimum floating-point value
```

The :HISTogram:MEASurement:MIN command specifies the histogram's measurement minimum. This is the lower bound of the histogram.

You can specify a <min> floating-point value, or if you want the measurement minimum to be automatically determined, use MIN.

Query

```
:HISTogram:MEASurement:MIN?
```

The :HISTogram:MEASurement:MIN? query returns the specified measurement minimum value.

If MIN was specified, the value returned is -1.79769313486232E+308 (the lowest 64-bit floating-point value).

Returned Format

```
<min><NL>
```

See Also

- ":HISTogram:MEASurement:MAX" on page 696

History

New in version 10.10.

:HISTogram:MODE

Command

```
:HISTogram:MODE {OFF | MEASurement | WAVeforms}
```

NOTE The MEASurement parameter is available only when the Jitter Analysis Software license is installed.

The :HISTogram:MODE command selects the histogram mode. The histogram may be off, set to track the waveforms, or set to track the measurement when the Jitter Analysis Software license is installed. When the Jitter Analysis Software license is installed, sending the :MEASure:JITTer:HISTogram ON command will automatically set :HISTOgram:MODE to MEASurement.

Example This example sets the histogram mode to track the waveform.

```
myScope.WriteString ":HISTogram:MODE WAVeform"
```

Query

```
:HISTogram:MODE?
```

The :HISTogram:MODE? query returns the currently selected histogram mode.

Returned Format

```
[:HISTogram:MODE] {OFF | MEASurement | WAVeform}<NL>
```

Example This example returns the result of the mode query and prints it to the computer's screen.

```
Dim strMode As String
myScope.WriteString ":HISTogram:MODE?"
strMode = myScope.ReadString
Debug.Print strMode
```

History Legacy command (existed before version 3.10).

:HISTogram:SCALe:SIZE

Command

```
:HISTogram:SCALe:SIZE <size>
```

The :HISTogram:SCALe:SIZE command sets histogram size for vertical and horizontal mode.

<size> The size is from 0.5 to 8.0 for the horizontal mode and from 0.5 to 10.0 for the vertical mode.

Example This example sets the histogram size to 3.5.

```
myScope.WriteString ":HISTogram:SCALe:SIZE 3.5"
```

Query

```
:HISTogram:SCALe:SIZE?
```

The :HISTogram:SCALe:SIZE? query returns the correct size of the histogram.

Returned Format

```
[:HISTogram:SCALe:SIZE] <size><NL>
```

Example This example returns the result of the size query and prints it to the computer's screen.

```
Dim strSize As String
myScope.WriteString ":HISTogram:SCALe:SIZE?"
strSize = myScope.ReadString
Debug.Print strSize
```

History Legacy command (existed before version 3.10).

:HISTogram:VERTical:BINS

Command

```
:HISTogram:VERTical:BINS <max_bins>
```

```
<max_bins> ::= integer from 10-1024
```

The :HISTogram:VERTical:BINS command sets the maximum number of bins used for a vertical waveform histogram.

Query

```
:HISTogram:VERTical:BINS?
```

The :HISTogram:VERTical:BINS? query returns the maximum number of bins.

Returned Format

```
<max_bins><NL>
```

```
<max_bins> ::= integer from 10-1024
```

See Also

- ":HISTogram:MODE" on page 698
- ":HISTogram:AXIS" on page 693
- ":HISTogram:HORizontal:BINS" on page 694
- ":HISTogram:MEASurement:BINS" on page 695

History New in version 5.20.

:HISTogram:WINDow:DEFault

Command

```
:HISTogram:WINDow:DEFault
```

The :HISTogram:WINDow:DEFault command positions the histogram markers to a default location on the display. Each marker will be positioned one division off the left, right, top, and bottom of the display.

Example This example sets the histogram window to the default position.

```
myScope.WriteString ":HISTogram:WINDow:DEFault"
```

History Legacy command (existed before version 3.10).

:HISTogram:WINDow:SOURce

Command

```
:HISTogram:WINDow:SOURce
```

The :HISTogram:WINDow:SOURce command selects the source of the histogram window. The histogram window will track the source's vertical and horizontal scale.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the histogram window's source to Channel 1.

```
myScope.WriteString ":HISTogram:WINDow:SOURce CHANnel1"
```

Query

```
:HISTogram:WINDow:SOURce?
```

The :HISTogram:WINDow:SOURce? query returns the currently selected histogram window source.

Returned Format

```
[:HISTogram:WINDow:SOURce] {CHAN<N> | DIFF<D> | COMM<C> | FUNC<F>
    | WMEM<N> | CLOC | EQU<L> | MTR | MSP | XT<X> | INP | CORR | ERR
    | LFPR}<NL>
```

Example This example returns the result of the window source query and prints it to the computer's screen.

```
Dim strWinsour As String
myScope.WriteString ":HISTogram:WINDow:SOURce?"
strWinsour = myScope.ReadString
Debug.Print strWinsour
```

History Legacy command (existed before version 3.10).

:HISTogram:WINDow:LLIMit

Command

```
:HISTogram:WINDow:LLIMit <left_limit>
```

The :HISTogram:WINDow:LLIMit command moves the Ax marker (left limit) of the histogram window. The histogram window determines the portion of the display used to build the database for the histogram. The histogram window markers will track the scale of the histogram window source.

<left_limit> A real number that represents the left boundary of the histogram window.

Example This example sets the left limit position to -200 microseconds.

```
myScope.WriteString ":HISTogram:WINDow:LLIMit -200E-6"
```

Query

```
:HISTogram:WINDow:LLIMit?
```

The :HISTogram:WINDow:LLIMit? query returns the value of the left limit histogram window marker.

Returned Format

```
[:HISTogram:WINDow:LLIMit] <left_limit><NL>
```

Example This example returns the result of the left limit position query and prints it to the computer's screen.

```
Dim strLL As String
myScope.WriteString ":HISTogram:WINDow:LLIMit?"
strLL = myScope.ReadString
Debug.Print strLL
```

History Legacy command (existed before version 3.10).

:HISTogram:WINDow:RLIMit

Command

```
:HISTogram:WINDow:RLIMit <right_limit>
```

The :HISTogram:WINDow:RLIMit command moves the Bx marker (right limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<right_limit> A real number that represents the right boundary of the histogram window.

Example This example sets the Bx marker to 200 microseconds.

```
myScope.WriteString ":HISTogram:WINDow:RLIMit 200E-6"
```

Query

```
:HISTogram:WINDow:RLIMit?
```

The :HISTogram:WINDow:RLIMit? query returns the value of the right histogram window marker.

Returned Format

```
[:HISTogram:WINDow:RLIMit] <right_limit><NL>
```

Example This example returns the result of the Bx position query and prints it to the computer's screen.

```
Dim strRL As String
myScope.WriteString ":HISTogram:WINDow:RLIMit?"
strRL = myScope.ReadString
Debug.Print strRL
```

History Legacy command (existed before version 3.10).

:HISTogram:WINDow:BLIMit

Command

```
:HISTogram:WINDow:BLIMit <bottom_limit>
```

The :HISTogram:WINDow:BLIMit command moves the Ay marker (bottom limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<bottom_limit> A real number that represents the bottom boundary of the histogram window.

Example This example sets the position of the Ay marker to -250 mV.

```
myScope.WriteString ":HISTogram:WINDow:BLIMit -250E-3"
```

Query

```
:HISTogram:WINDow:BLIMit?
```

The :HISTogram:WINDow:BLIMit? query returns the value of the Ay histogram window marker.

Returned Format

```
[:HISTogram:WINDow:BLIMit] <bottom_limit><NL>
```

Example This example returns the result of the Ay position query and prints it to the computer's screen.

```
Dim strBL As String
myScope.WriteString ":HISTogram:WINDow:BLIMit?"
strBL = myScope.ReadString
Debug.Print strBL
```

History Legacy command (existed before version 3.10).

:HISTogram:WINDow:TLIMit

Command `:HISTogram:WINDow:TLIMit <top_limit>`

The :HISTogram:WINDow:TLIMit command moves the By marker (top limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<top_limit> A real number that represents the top boundary of the histogram window.

Example This example sets the position of the By marker to 250 mV.

```
myScope.WriteString ":HISTogram:WINDow:TLIMit 250E-3"
```

Query `:HISTogram:WINDow:TLIMit?`

The :HISTogram:WINDow:TLIMit? query returns the value of the By histogram window marker.

Returned Format `[:HISTogram:WINDow:TLIMit] <top_limit><NL>`

Example This example returns the result of the By position query and prints it to the computer's screen.

```
Dim strTL As String
myScope.WriteString ":HISTogram:WINDow:TLIMit?"
strTL = myScope.ReadString
Debug.Print strTL
```

History Legacy command (existed before version 3.10).

27 :HOSTed Commands

The commands in the HOSTed subsystem are used to configure and manage the list of oscilloscopes in a MultiScope system.

MultiScope systems can combine up to 10 Infiniium oscilloscopes to create an oscilloscope system with up to 40 time-synchronized channels.

In a MultiScope system, oscilloscopes are connected in daisy-chain configuration where the reference clock output of the Leader oscilloscope is connected to the reference clock input of the Follower 1 oscilloscope and the trigger output of the Leader oscilloscope is connected to the auxiliary trigger input of the Follower 1 oscilloscope, and so on. A calibration signal from one of the Follower 1 oscilloscope is split and fed into a channel input on all the oscilloscopes to set up time-correlation.

For more information on MultiScope systems, see:

- The *Keysight MultiScope Hardware Configuration Guide*.
- The online help in the Infiniium Offline software.

:HOSTed:CALibrate:CALibrate

Command `:HOSTed:CALibrate:CALibrate`

The :HOSTed:CALibrate:CALibrate command performs the MultiScope system time-correlation calibration at the level selected by :HOSTed:CALibrate:LEVel.

This command does nothing when the MANual level is selected.

To get the status of the calibration, use the :HOSTed:CALibrate:STATus:LEVel? query.

See Also

- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:CHANnel

Command

```
:HOSTed:CALibrate:CHANnel <source>

<source> ::= CHANnel<N>
```

The :HOSTed:CALibrate:CHANnel command selects the calibration channel or synchronization input where the MultiScope system time-correlation procedures expect to find the calibration signal.

This signal comes from the Follower 1 oscilloscope's calibrator output and is split and fed into the calibration channel on each oscilloscope in the MultiScope system.

<N> An integer, 1 to the number of analog input channels.

Query

```
:HOSTed:CALibrate:CHANnel?
```

The :HOSTed:CALibrate:CHANnel? query returns the selected the calibration channel or synchronization input.

Returned Format

```
[:HOSTed:CALibrate:CHANnel] <source><NL>

<source> ::= CHAN<N>
```

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:DESKew:CHANnels

Command `:HOSTed:CALibrate:DESKew:CHANnels`

The :HOSTed:CALibrate:DESKew:CHANnels command deskews all channels in the MultiScope system. The calibration process prompts you to connect the Leader oscilloscope's Cal Out signal to each of the system's input channels in turn.

To get the status of the calibration, use the :HOSTed:CALibrate:STATus:CHANnels? query.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:DESKew:FRAMes

Command `:HOSTed:CALibrate:DESKew:FRAMes`

The :HOSTed:CALibrate:DESKew:FRAMes command deskews frames in the MultiScope system by measuring only one input channel from each oscilloscope frame. It assumes that the skew of all channels within a single oscilloscope frame is the same since they were deskewed in production.

To get the status of the calibration, use the :HOSTed:CALibrate:STATus:FRAMes? query.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:DESKew:SIGNals

Command `:HOSTed:CALibrate:DESKew:SIGNals`

Given a common edge on configured channels in the MultiScope system, send the :HOSTed:CALibrate:DESKew:SIGNals command to align the horizontal positions of the closest rising edges of all input signals.

This is typically used for demonstration or quick verification purposes, but can also be a quick alternative to the system deskew process if your SUT (signals under test) are already connected and have the necessary rising edges.

To get the status of the calibration, use the :HOSTed:CALibrate:STATus:SIGNals? query.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:DESKew:ZERO

Command `:HOSTed:CALibrate:DESKew:ZERO`

The :HOSTed:CALibrate:DESKew:ZERO command resets MultiScope system signal skew values to zero.

This is provided as a convenience because the skew values are distributed across all channels of all oscilloscope frames and are not changed by setup recall or default setup. Factory default setup does set all skew values to zero, but it must be performed on all oscilloscope frames.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:LEVel

Command

```
:HOSTed:CALibrate:LEVel <level>

<level> ::= {MANual | BASic | PRECision}
```

The :HOSTed:CALibrate:LEVel command selects the MultiScope system calibration level:

- MANual — No time-correlation calibration is performed. However, you can still:
  - Capture and view signals in the MultiScope system.
  - Phase-lock the timebase reference clocks of the oscilloscopes in the MultiScope system.
  - Input a calibration signal to the oscilloscopes and manually measure the time skew between oscilloscope frames.
  - Manually enter skew values for waveforms to align them in time.
- BASic — The time-correlation calibration procedure automates all the time calibration steps you could perform manually.

  The calibration output from the Follower 1 oscilloscope is split and fed into the calibration channel on each oscilloscope in the MultiScope system.

  After the basic calibration is performed, you can disconnect the calibration channel on each oscilloscope and use it as a normal input channel.
- PRECision — Select this calibration level if you want to perform jitter and drift time correction between the oscilloscopes in the MultiScope system. This calibration level gives you the highest time accuracy because jitter and drift calibrations continue to be made as the oscilloscopes acquire data.

  For this calibration level, the calibration channel must remain connected during normal operation.

To perform the MultiScope system time-correlation calibration at the BASic or PRECision levels, send the :HOSTed:CALibrate:CALibrate command.

Query

```
:HOSTed:CALibrate:LEVel?
```

The :HOSTed:CALibrate:LEVel? query returns the selected calibration level.

Returned Format

```
[:HOSTed:CALibrate:LEVel] <level><NL>

<level> ::= {MAN | BAS | PREC}
```

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714

- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:PROMpt

Command `:HOSTed:CALibrate:PROMpt {{0 | OFF} | {1 | ON}}`

The :HOSTed:CALibrate:PROMpt command specifies whether the user interface software prompts you to make the proper connections while the MultiScope system calibration runs.

You can select OFF when you know required connections have already been made and you do not want to be prompted to make them.

Query `:HOSTed:CALibrate:PROMpt?`

The :HOSTed:CALibrate:PROMpt? query returns the setting.

Returned Format `[:HOSTed:CALibrate:PROMpt] <setting><NL>`

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

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:STATus:CHANnels?

Query

```
:HOSTed:CALibrate:STATus:CHANnels?
```

The :HOSTed:CALibrate:STATus:CHANnels? query returns the MultiScope system calibration status of "Deskew Channels" (see :HOSTed:CALibrate:DESKew:CHANnels).

Returned Format

```
<status><NL>
```

<status>	Status Description
UNAVAILABLE	Configuration is incompatible. This could be because: ▪ There are insufficient connections ▪ Manual Calibration is selected
NOTAPPLIED	The deskew is available, but not currently done.
PASSED	Calibration completed and passed.
INPROGRESS	Calibration is in progress.

Nothing is applied for a status result other than PASSED.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:STATus:FRAMes?

Query `:HOSTed:CALibrate:STATus:FRAMes?`

The :HOSTed:CALibrate:STATus:FRAMes? query returns the MultiScope system calibration status of "Deskew Frames" (see :HOSTed:CALibrate:DESKew:FRAMes).

Returned Format `<status><NL>`

<status>	Status Description
UNAVAILABLE	Configuration is incompatible. This could be because: ▪ There are insufficient connections ▪ Manual Calibration is selected
NOTAPPLIED	The deskew is available, but not currently done.
PASSED	Calibration completed and passed.
INPROGRESS	Calibration is in progress.

Nothing is applied for a status result other than PASSED.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:STATus:LEVel?

Query

```
:HOSTed:CALibrate:STATus:LEVel?
```

The :HOSTed:CALibrate:STATus:LEVel? query returns the MultiScope system calibration status of the currently selected calibration level (see :HOSTed:CALibrate:LEVel and :HOSTed:CALibrate:CALibrate).

Returned Format

```
<status><NL>
```

<status>	Status Description
FAILED	Calibration completed and failed.
PASSED	Calibration completed and passed.
INPROGRESS	Calibration is in progress.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History New in version 5.50.

:HOSTed:CALibrate:STATus:SIGNals?

Query

```
:HOSTed:CALibrate:STATus:SIGNals?
```

The :HOSTed:CALibrate:STATus:SIGNals? query returns the MultiScope system calibration status of "Deskew Signals" (see :HOSTed:CALibrate:DESKew:SIGNals).

Returned Format

```
<status><NL>
```

<status>	Status Description
UNAVAILABLE	Configuration is incompatible. This could be because: ▪ There are insufficient connections ▪ Manual Calibration is selected
NOTAPPLIED	The deskew is available, but not currently done.
PASSED	Calibration completed and passed.
INPROGRESS	Calibration is in progress.

Nothing is applied for a status result other than PASSED.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History

New in version 5.50.

:HOSTed:CALibrate:TREF:DETect

Command `:HOSTed:CALibrate:TREF:DETect`

When the MultiScope system's timebase reference clock status is unlocked, or after you have changed reference clock connections between oscilloscopes, send the :HOSTed:CALibrate:TREF:DETect command to automatically detect the reference clock connection between oscilloscopes.

Running the BASic or PRECision calibrations (see :HOSTed:CALibrate:LEVel) will automatically detect the timebase reference clock. In the MANual calibration level, you can send the :HOSTed:CALibrate:TREF:DETect command.

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721

History New in version 5.50.

:HOSTed:EOACq

Command `:HOSTed:EOACq {{0 | OFF} | {1 | ON}}`

With Infiniium Offline only, the :HOSTed:EOACq command enables or disables the **Enable Offline Acquisition** user preference.

You must enable the **Enable Offline Acquisition** user preference when sending waveforms from multiple acquisitions to Infiniium Offline using the :WAVeform:DATA command.

Query `:HOSTed:EOACq?`

The :HOSTed:EOACq? query returns whether the **Enable Offline Acquisition** user preference is enabled or disabled.

Returned Format `<setting><NL>`

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

See Also

- Chapter 11, "Analyzing Multiple Acquisitions in Infiniium Offline," starting on page 211
- ":WAVeform:DATA" on page 1675

History New in version 11.20.

:HOSTed:FOLLower<N>:ACHannels?

Query

```
:HOSTed:FOLLower<N>:ACHannels?
```

The :HOSTed:FOLLower<N>:ACHannels? query returns the channel numbers assigned to a Follower oscilloscope in a MultiScope system. There can be up to 9 Follower oscilloscopes in a MultiScope system.

<N> An integer, 1-9.

Returned Format

```
<channel_range><NL>

<channel_range> ::= range of channel numbers, for example, 5-8
```

See Also

- ":HOSTed:NCONnected?" on page 732
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727
- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731

History New in version 5.50.

:HOSTed:FOLLower<N>:CONFigure

Command

```
:HOSTed:FOLLower<N>:CONFigure "<instrument_VISA_string>"
```

The :HOSTed:FOLLower<N>:CONFigure command identifies a Follower oscilloscope in a MultiScope system by its VISA address. There can be up to 9 Follower oscilloscopes in a MultiScope system.

<N> An integer, 1-9.

Example This example identifies the oscilloscope whose VISA address is "TCPIP0::141.121.237.226::inst0::INSTR" as the Follower 1 oscilloscope.

```
myScope.WriteString ":HOSTed:FOLLower1:CONFigure 'TCPIP0::141.121.237.22
6::inst0::INSTR'"
```

See Also

- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727
- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731

History New in version 5.50.

:HOSTed:FOLLower<N>:CONNect

Command `:HOSTed:FOLLower<N>:CONNect`

The :HOSTed:FOLLower<N>:CONNect command opens the connection to a Follower oscilloscope in a MultiScope system. There can be up to 9 Follower oscilloscopes in a MultiScope system.

<N> An integer, 1-9.

See Also

- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:DISConnect" on page 727
- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731

History New in version 5.50.

:HOSTed:FOLLower<N>:DISConnect

Command

```
:HOSTed:FOLLower<N>:DISConnect
```

The :HOSTed:FOLLower<N>:DISConnect command closes the connection to a Follower oscilloscope in a MultiScope system. There can be up to 9 Follower oscilloscopes in a MultiScope system.

<N> An integer, 1-9.

See Also

- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731

History New in version 5.50.

:HOSTed:LEADer:ACHannels?

Query

```
:HOSTed:LEADer:ACHannels?
```

The :HOSTed:LEADer:ACHannels? query returns the channel numbers assigned to the Leader oscilloscope in a MultiScope system.

Returned Format

```
<channel_range><NL>

<channel_range> ::= range of channel numbers, for example, 1-4
```

See Also

- ":HOSTed:NCONnected?" on page 732
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731
- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727

History New in version 5.50.

:HOSTed:LEADer:CONFigure

Command

```
:HOSTed:LEADer:CONFigure "<instrument_VISA_string>"
```

The :HOSTed:LEADer:CONFigure command identifies the Leader oscilloscope in a MultiScope system by its VISA address.

Example

This example identifies the oscilloscope whose VISA address is "TCPIP0::141.121.238.47::inst0::INSTR" as the Leader oscilloscope.

```
myScope.WriteString ":HOSTed:LEADer:CONFigure 'TCPIP0::141.121.238.47::i
nst0::INSTR'"
```

See Also

- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:LEADer:DISConnect" on page 731
- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727

History

New in version 5.50.

:HOSTed:LEADer:CONNect

Command `:HOSTed:LEADer:CONNect`

The :HOSTed:LEADer:CONNect command opens the connection to the Leader oscilloscope in a MultiScope system.

See Also

- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:DISConnect" on page 731
- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727

History New in version 5.50.

:HOSTed:LEADer:DISConnect

Command `:HOSTed:LEADer:DISConnect`

The :HOSTed:LEADer:DISConnect command closes the connection to the Leader oscilloscope in a MultiScope system.

See Also

- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:LEADer:CONFigure" on page 729
- ":HOSTed:LEADer:CONNect" on page 730
- ":HOSTed:FOLLower<N>:ACHannels?" on page 724
- ":HOSTed:FOLLower<N>:CONFigure" on page 725
- ":HOSTed:FOLLower<N>:CONNect" on page 726
- ":HOSTed:FOLLower<N>:DISConnect" on page 727

History New in version 5.50.

:HOSTed:NCONnected?

Query `:HOSTed:NCONnected?`

The :HOSTed:NCONnected? query returns a number that indicates whether in hosted mode, and if in hosted mode, the number of hosted oscilloscope frames connected. This query can return:

- 0 — Not in hosted mode. There are no hosted oscilloscope frames connected.
- 1 — One hosted oscilloscope frame is connected, either itself or a remote frame when using Infiniium Offline.
- 2 through 10 — The number of hosted oscilloscope frames connected.

With four analog input channels in each frame, the number returned tells you the potential number of channels in the MultiScope system. If the query returns a 0 or 1, you know there can be up to four channels, if the query returns 2 or more, multiply the returned number by four to get the number of possible channels.

Returned Format

```
<#_of_frames><NL>

<#_of_frames> ::= number of hosted oscilloscope frames connected,
   from 0 to 10 in NR1 format
```

See Also

- ":HOSTed:LEADer:ACHannels?" on page 728
- ":HOSTed:FOLLower<N>:ACHannels?" on page 724

History New in version 6.10.

:HOSTed:PERiodic

Command

```
:HOSTed:PERiodic <drift_corr>

<drift_corr> ::= {OFF | TIME}
```

The :HOSTed:PERiodic command turns periodic drift correction on (TIME) or off.

Query

```
:HOSTed:PERiodic?
```

The :HOSTed:PERiodic? query returns the periodic drift correction setting.

Returned Format

```
<drift_corr><NL>

<drift_corr> ::= {OFF | TIME}
```

See Also

- ":HOSTed:PERiodic" on page 733

History

New in version 5.70.

28 :ISCan (InfiniiScan) Commands

The ISCan commands and queries control the InfiniiScan feature of the oscilloscope. InfiniiScan provides several ways of searching through the waveform data to find unique events.

:ISCan:DELay

Command

```
:ISCan:DELay {OFF | <delay_time>}
```

The :ISCan:DELay command sets the delay time from when the hardware trigger occurs and when InfiniiScan tries to find the waveform event that has been defined.

OFF Turns off the delay from the hardware trigger.

<delay_time> Sets the amount of time that the InfiniiScan trigger is delayed from the hardware trigger.

Example The following example causes the oscilloscope to delay by 1 ms.

```
myScope.WriteString ":ISCan:DELay 1E-06"
```

Query

```
:ISCan:DELay?
```

The query returns the current set delay value.

Returned Format

```
[:ISCan:DELay] {OFF | <delay_time>}<NL>
```

Example The following example returns the current delay value and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:DELay?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:MEASurement:FAIL

Command `:ISCan:MEASurement:FAIL {INSide | OUTSide}`

The :ISCan:MEASurement:FAIL command sets the fail condition for an individual measurement. The conditions for a test failure are set on the measurement selected by the :ISCan:MEASurement command.

When a measurement failure is detected by the limit test the oscilloscope triggers and the trigger action is executed.

INSide INside causes the oscilloscope to fail a test when the measurement results are within the parameters set by the :ISCan:MEASurement:LIMit and :ISCan:MEASurement:ULIMit commands.

OUTSide OUTside causes the oscilloscope to fail a test when the measurement results exceed the parameters set by the :ISCan:MEASurement:LLIMit and the :ISCan:MEASurement:ULIMit commands.

Example The following example causes the oscilloscope to trigger when the measurements are outside the lower or upper limits.

```
myScope.WriteString ":ISCan:MEASurement:FAIL OUTSide"
```

Query `:ISCan:MEASurement:FAIL?`

The query returns the current set fail condition.

Returned Format `[:ISCan:MEASurement:FAIL] {INSide | OUTSide}<NL>`

Example The following example returns the current fail condition and prints the result to the controller's screen.

```
Dim strFAIL As String
myScope.WriteString ":ISCan:MEASurement:FAIL?"
strFAIL = myScope.ReadString
Debug.Print strFAIL
```

History Legacy command (existed before version 3.10).

:ISCan:MEASurement:LLIMit

Command

```
:ISCan:MEASurement:LLIMit <lower_value>
```

The :ISCan:MEASurement:LLIMit (lower limit) command sets the lower test limit for the currently selected measurement. The :ISCan:MEASurement command selects the measurement used.

<lower_value> A real number.

Example The following example sets the lower test limit to 1.0.

```
myScope.WriteString ":ISCan:MEASurement:LLIMit 1.0"
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the measurement limit to trigger when the signal is outside the specified limit.

Query

```
:ISCan:MEASurement:LLIMit?
```

The query returns the current value set by the command.

Returned Format

```
[:ISCan:MEASurement:LLIMit]<lower_value><NL>
```

Example The following example returns the current lower test limit and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:MEASurement:LLIMit?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:MEASurement

Command

```
:ISCan:MEASurement {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20}
```

The :ISCan:MEASurement command selects the current source for Measurement Limit Test Trigger. It selects one of the active measurements as referred to by their position in the Measurement tab area at the bottom of the screen. Measurements are numbered from left to right in the Measurements tab area of the screen.

Example

The following example selects the first measurement as the source for the limit testing commands.

```
myScope.WriteString ":ISCan:MEASurement MEAS1"
```

Query

```
:ISCan:MEASurement?
```

The query returns the currently selected measurement source.

Returned Format

```
[:ISCan:MEASurement] {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20}<NL>
```

Example

The following example returns the currently selected measurement source for the limit testing commands.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:MEASurement?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also

Measurements are started by the commands in the Measurement Subsystem.

History

Legacy command (existed before version 3.10).

Version 5.00: Now 20 measurements to choose from.

:ISCan:MEASurement:ULIMit

Command

```
:ISCan:MEASurement:ULIMit <upper_value>
```

The :ISCan:MEASurement:ULIMit (upper limit) command sets the upper test limit for the active measurement currently selected by the :ISCan:MEASurement command.

<upper_value> A real number.

Example The following example sets the upper limit of the currently selected measurement to 500 mV.

```
myScope.WriteString ":ISCan:MEASurement:ULIMit 500E-3"
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the :ISCan:MEASurement:FAIL OUTside command to specify that the oscilloscope will trigger when the voltage exceeds 500 mV.

Query

```
:ISCan:MEASurement:ULIMit?
```

The query returns the current upper limit of the limit test.

Returned Format

```
[:ISCan:MEASurement:ULIMit] <upper_value><NL>
```

Example The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:MEASurement:ULIMit?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:MODE

Command `:ISCan:MODE {OFF | MEASurement | NONMonotonic | RUNT | SERial | ZONE}`

The :ISCan:MODE command selects the type of InfiniiScan trigger mode:

- OFF — Turns off the InfiniiScan trigger mode.
- MEASurement — Sets the Measurement limit trigger mode.
- NONMonotonic — Sets the Non-monotonic Edge trigger mode.
- RUNT — Sets the Runt trigger mode.
- SERial — Sets the General Serial trigger mode.
- ZONE — Sets the Zone Qualify trigger mode.

Example The following example selects the runt trigger.

```
myScope.WriteString ":ISCan:MODE RUNT"
```

Query `:ISCan:MODE?`

The query returns the currently selected InfiniiScan trigger mode.

Returned Format `[:ISCan:MEASurement] {OFF | MEAS | NONM | RUNT | SER | ZONE}<NL>`

Example The following example returns the currently selected InfiniiScan trigger mode.

```
Dim strMODE As String
myScope.WriteString ":ISCan:MODE?"
strMODE = myScope.ReadString
Debug.Print strMODE
```

History Legacy command (existed before version 3.10).

:ISCan:NONMonotonic:EDGE

Command

```
:ISCan:NONMonotonic:EDGE {EITHer | FALLing | RISing}
```

The :ISCan:NONMonotonic:EDGE command selects the rising edge, the falling edge, or either edge for the Non-monotonic edge trigger mode.

EITHer Sets the edge used by the Non-monotonic edge trigger to both rising and falling edges.

FALLing Sets the edge used by the Non-monotonic edge trigger to falling edges.

RISing Sets the edge used by the Non-monotonic edge trigger to rising edges.

Example The following example selects the falling edge non-monotonic trigger.

```
myScope.WriteString ":ISCan:NONMonotonic:EDGE FALLing"
```

Query

```
:ISCan:NONMonotonic:EDGE?
```

The query returns the currently selected edge type for the Non-Monotonic Edge trigger.

Returned Format

```
[:ISCan:NONMonotonic:EDGE]{EITHer | FALLing | RISing}<NL>
```

Example The following example returns the currently selected edge type used for the Non-monotonic Edge trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:NONMonotonic:EDGE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

History Legacy command (existed before version 3.10).

:ISCan:NONMonotonic:HYSTeresis

Command

```
:ISCan:NONMonotonic:HYSTeresis <value>
```

The :ISCan:NONMonotonic:HYSTeresis command sets the hysteresis value used for the Non-monotonic Edge trigger.

<value> is a real number for the hysteresis.

Example The following example sets the hysteresis value used by the Non-monotonic trigger mode to 10 mV.

```
myScope.WriteString ":ISCan:NONMonotonic:HYSTeresis 1E-2"
```

Query

```
:ISCan:NONMonotonic:HYSTeresis?
```

The query returns the hysteresis value used by the Non-monotonic Edge trigger mode.

Returned Format

```
[:ISCan:NONMonotonic:HYSTeresis]<value><NL>
```

Example The following example returns and prints the value of the hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":ISCan:NONMonotonic:HYSTeresis?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:NONMonotonic:SOURce

Command

```
:ISCan:NONMonotonic:SOURce CHANnel<N>
```

The :ISCan:NONMonotonic:SOURce command sets the source used for the Non-monotonic Edge trigger.

<N> An integer, 1 to the number of analog input channels.

Example The following example sets the source used by the Non-monotonic trigger mode to channel 1.

```
myScope.WriteString ":ISCan:NONMonotonic:SOURce CHANnel1"
```

Query

```
:ISCan:NONMonotonic:SOURce?
```

The query returns the source used by the Non-monotonic Edge trigger mode.

Returned Format

```
[:ISCan:NONMonotonic:SOURce] CHANnel<N><NL>
```

Example The following example returns the currently selected source for the Non-monotonic Edge trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:NONMonotonic:SOURce?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

History Legacy command (existed before version 3.10).

:ISCan:RUNT:HYSTeresis

Command

```
:ISCan:RUNT:HYSTeresis <value>
```

The :ISCan:RUNT:HYSTeresis command sets the hysteresis value used for the Runt trigger.

<value> is a real number for the hysteresis.

Example The following example sets the hysteresis value used by the Runt trigger mode to 10 mV.

```
myScope.WriteString ":ISCan:RUNT:HYSTeresis 1E-2"
```

Query

```
:ISCan:RUNT:HYSTeresis?
```

The query returns the hysteresis value used by the Runt trigger mode.

Returned Format

```
[:ISCan:RUNT:HYSTeresis]<value><NL>
```

Example The following example returns and prints the value of the hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:RUNT:HYSTeresis?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:RUNT:LLEVel

Command

```
ISCan:RUNT:LLEVel <lower_level>
```

The :ISCan:RUNT:LLEVel (lower level) command sets the lower level limit for the Runt trigger mode.

<lower_level> A real number.

Example The following example sets the lower level limit to 1.0 V.

```
myScope.WriteString ":ISCan:RUNT:LLEVel 1.0"
```

Query

```
:ISCan:RUNT:LLEVel?
```

The query returns the lower level limit set by the command.

Returned Format

```
[:ISCan:RUNT:LLEVel] <lower_level><NL>
```

Example The following example returns the current lower level used by the Runt trigger and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:RUNT:LLEVel?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:RUNT:SOURce

Command

```
:ISCan:RUNT:SOURce CHANnel<N>
```

The :ISCan:RUNT:SOURce command sets the source used for the Runt trigger.

<N> An integer, 1 to the number of analog input channels.

Example The following example sets the source used by the Runt trigger mode to channel 1.

```
myScope.WriteString ":ISCan:RUNT:SOURce CHANnel1"
```

Query

```
:ISCan:RUNT:SOURce?
```

The query returns the source used by the Runt trigger mode.

Returned Format

```
[:ISCan:RUNT:SOURce] CHANnel<N><NL>
```

Example The following example returns the currently selected source for the Runt trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:RUNT:SOURce?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

History Legacy command (existed before version 3.10).

:ISCan:RUNT:ULEVel

Command

```
:ISCan:RUNT:ULEVel <upper_level>
```

The :ISCan:RUNT:ULEVel (upper level) command sets the upper level limit for the Runt trigger mode.

<upper_level> A real number.

Example The following example sets the upper level value used by the Runt trigger mode to 500 mV.

```
myScope.WriteString ":ISCan:RUNT:ULEVel 500E-3"
```

Query

```
:ISCan:RUNT:ULEVel?
```

The query returns the current upper level value used by the Runt trigger.

Returned Format

```
[:ISCan:RUNT:ULEVel] <upper_level><NL>
```

Example The following example returns the current upper level used by the Runt trigger and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":ISCan:RUNT:ULEVel?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:ISCan:SERial:PATTern

Command

```
:ISCan:SERial:PATTern "<pattern>"
```

The :ISCan:SERial:PATTern command sets the pattern used for the Serial trigger.

<pattern> is a 1, 0, or X binary character string of up to 80 characters. The pattern can only be expressed in the binary format.

Example The following example sets the pattern used by the Serial trigger to 101100.

```
myScope.WriteString ":ISCan:SERial:PATTern ""101100"""
```

Query

```
:ISCan:SERial:PATTern?
```

The query returns the pattern used by the Serial trigger mode.

Returned Format

```
[:ISCan:SERial:PATTern] <pattern><NL>
```

Example The following example returns the currently selected pattern for the Serial trigger mode.

```
Dim strPATTERN As String
myScope.WriteString ":ISCan:SERial:PATTern?"
strPATTERN = myScope.ReadString
Debug.Print strPATTERN
```

History Legacy command (existed before version 3.10).

:ISCan:SERial:SOURce

Command `:ISCan:SERial:SOURce CHANnel<N>`

The :ISCan:SERial:SOURce command sets the source used for the Serial trigger.

<N> An integer, 1 to the number of analog input channels.

Example The following example sets the source used by the Serial trigger mode to channel 1.

```
myScope.WriteString ":ISCan:SERial:SOURce CHANnel1"
```

Query `:ISCan:SERial:SOURce?`

The query returns the source used by the Serial trigger mode.

Returned Format `[:ISCan:SERial:SOURce]CHANnel<N><NL>`

Example The following example returns the currently selected source for the Serial trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:SERial:SOURce?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

History Legacy command (existed before version 3.10).

:ISCan:ZONE:HIDE

Command

```
:ISCan:ZONE:HIDE {{ON | 1} | {OFF | 0}}
```

The :ISCan:ZONE:HIDE command lets you hide or show all InfiniiScan zones on the display.

Example The following example hides all InfiniiScan zones on the display.

```
myScope.WriteString ":ISCan:ZONE:HIDE ON"
```

Query

```
:ISCan:ZONE:HIDE?
```

The query returns the current zone hide setting.

Returned Format

```
[:ISCan:ZONE:HIDE] {1 | 0}<NL>
```

Example The following example returns the current zone hide setting.

```
Dim strHide As String
myScope.WriteString ":ISCan:ZONE:HIDE?"
strHide = myScope.ReadString
Debug.Print strHide
```

See Also

- ":ISCan:ZONE:SOURce" on page 752
- ":ISCan:ZONE<Z>:MODE" on page 753
- ":ISCan:ZONE<Z>:PLACement" on page 754
- ":ISCan:ZONE<Z>:SOURce" on page 755
- ":ISCan:ZONE<Z>:STATe" on page 756

History New in version 4.00.

:ISCan:ZONE:SOURce

Command

```
:ISCan:ZONE:SOURce CHANnel<N>
```

The :ISCan:ZONE:SOURce command sets the source for all zones used in the zone qualify trigger.

<N> An integer, 1 to the number of analog input channels.

Example The following example sets the source used by all zones in the zone qualify trigger to channel 1.

```
myScope.WriteString ":ISCan:ZONE:SOURce CHANnel1"
```

Query

```
:ISCan:ZONE:SOURce?
```

The query returns the source used for all zones in the zone qualify trigger.

Returned Format

```
[:ISCan:ZONE:SOURce] CHANnel<N><NL>
```

Example The following example returns the currently selected source for all zones in the zone qualify trigger.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:ZONE:SOURce?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also

- ":ISCan:ZONE:HIDE" on page 751
- ":ISCan:ZONE<Z>:MODE" on page 753
- ":ISCan:ZONE<Z>:PLACement" on page 754
- ":ISCan:ZONE<Z>:SOURce" on page 755
- ":ISCan:ZONE<Z>:STATe" on page 756

History Legacy command (existed before version 3.10).

:ISCan:ZONE<Z>:MODE

Command

```
:ISCan:ZONE<Z>:MODE {INTersect | NOTintersect | OINTersect | ONOT}
```

The :ISCan:ZONE<Z>:MODE command sets the Zone Qualify trigger mode. For the INTersect mode, the waveform must enter the zone region to qualify as a valid waveform. For NOTintersect mode, the waveform cannot enter a zone region to qualify as a valid waveform.

<Z> An integer from 1-8.

Example The following example sets the mode to intersect for zone 1.

```
myScope.WriteString ":ISCan:ZONE1:MODE INTersect"
```

Query

```
:ISCan:ZONE<Z>:MODE?
```

The query returns the mode used by zone 1.

Returned Format

```
[:ISCan:ZONE<Z>:MODE] {INT | NOT | OINT | ONOT}<NL>
```

Example The following example returns the currently selected mode for zone 1.

```
Dim strMODE As String
myScope.WriteString ":ISCan:ZONE1:MODE?"
strMODE = myScope.ReadString
Debug.Print strMODE
```

See Also

- ":ISCan:ZONE:HIDE" on page 751
- ":ISCan:ZONE:SOURce" on page 752
- ":ISCan:ZONE<Z>:PLACement" on page 754
- ":ISCan:ZONE<Z>:SOURce" on page 755
- ":ISCan:ZONE<Z>:STATe" on page 756

History Legacy command (existed before version 3.10).

:ISCan:ZONE<Z>:PLACement

Command

```
:ISCan:ZONE<Z>:PLACement <width>,<height>,<x_center>,<y_center>
```

The :ISCan:ZONE<Z>:PLACement command sets the location and size of a zone for the zone qualify trigger mode.

<Z> An integer from 1-8.

<width> A real number defining the width of a zone in seconds.

<height> A real number defining the height of a zone in volts.

<x_center> A real number defining the x coordinate of the center of the zone in seconds.

<y_center> A real number defining the y coordinate of the center of the zone in volts.

Example The following example sets the size of zone 1 to be 500 ps wide and 0.5 volts high and centered about the xy coordinate of 1.5 ns and 1 volt.

```
myScope.WriteString ":ISCan:ZONE1:PLACement 500e-12,0.5,1.5e-9,1"
```

Query

```
:ISCan:ZONE<Z>:PLACement?
```

The query returns the placement values used by zone 1.

Returned Format

```
[:ISCan:ZONE<Z>:PLACement] <width>,<height>,<x_center>,<y_center><NL>
```

Example The following example returns the current placement values for zone 1.

```
Dim strPLACEMENT As String
myScope.WriteString ":ISCan:ZONE1:PLACement?"
strPLACEMENT = myScope.ReadString
Debug.Print strPLACEMENT
```

See Also

- ":ISCan:ZONE:HIDE" on page 751
- ":ISCan:ZONE:SOURce" on page 752
- ":ISCan:ZONE<Z>:MODE" on page 753
- ":ISCan:ZONE<Z>:SOURce" on page 755
- ":ISCan:ZONE<Z>:STATe" on page 756

History Legacy command (existed before version 3.10).

:ISCan:ZONE<Z>:SOURce

Command

```
:ISCan:ZONE<Z>:SOURce CHANnel<N>
```

The :ISCan:ZONE<Z>:SOURce command sets the source used for a particular zone in the zone qualify trigger.

<Z> An integer from 1-8.

<N> An integer, 1 to the number of analog input channels.

Example The following example sets the source used by zone 1 to channel 1.

```
myScope.WriteString ":ISCan:ZONE1:SOURce CHANnel1"
```

Query

```
:ISCan:ZONE<Z>:SOURce?
```

The query returns the source used by the particular zone.

Returned Format

```
[:ISCan:ZONE<Z>:SOURce] CHANnel<N><NL>
```

Example The following example returns the currently selected source for zone 1.

```
Dim strSOURCE As String
myScope.WriteString ":ISCan:ZONE1:SOURce?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also

- ":ISCan:ZONE:HIDE" on page 751
- ":ISCan:ZONE:SOURce" on page 752
- ":ISCan:ZONE<Z>:MODE" on page 753
- ":ISCan:ZONE<Z>:PLACement" on page 754
- ":ISCan:ZONE<Z>:STATe" on page 756

History Legacy command (existed before version 3.10).

:ISCan:ZONE<Z>:STATe

Command

```
:ISCan:ZONE<Z>:STATe {{ON | 1} | {OFF | 0}}
```

The :ISCan:ZONE<Z>:STATe command turns a zone off or on for the zone qualify trigger.

<Z> An integer from 1-8.

Example The following example turns on zone 2.

```
myScope.WriteString ":ISCan:ZONE2:STATe ON"
```

Query

```
:ISCan:ZONE<Z>:STATe?
```

The query returns the state value for a zone.

Returned Format

```
[:ISCan:ZONE<Z>:STATe] {1 | 0}<NL>
```

Example The following example returns the current state value for zone 2.

```
Dim strSTATE As String
myScope.WriteString ":ISCan:ZONE2:STATe?"
strSTATE = myScope.ReadString
Debug.Print strSTATE
```

See Also

- ":ISCan:ZONE:HIDE" on page 751
- ":ISCan:ZONE:SOURce" on page 752
- ":ISCan:ZONE<Z>:MODE" on page 753
- ":ISCan:ZONE<Z>:PLACement" on page 754
- ":ISCan:ZONE<Z>:SOURce" on page 755

History Legacy command (existed before version 3.10).

29 :LANE<N> (Equalization) Commands

The Equalization application is used to re-open partially or completely closed real-time eye diagrams. For additional information on equalization, consult the *Infiniium Serial Data Equalization User's Guide*.

Before the 6.40 version of Infiniium oscilloscope software, you could perform equalization on a single input source. Now, you can perform equalization on up to four sources at once. Each of the four "lanes" of equalization has its own settings, distinct from the other lanes, allowing for independent equalization on different signals, concurrent equalization on the same signal, or any combination thereof.

:LANE<N>:COPYto

Command

```
:LANE<N>:COPYto LANE<L>
```

The :LANE<N>:COPYto command copies all valid settings from LANE<N> to LANE<L> (both <N> and <L> are integers between 1 and 4, inclusive).

This command includes all CTLE, FFE, and DFE settings, with one special case exception: If LANE<X> is using EQUalized<X-1> as its source, and LANE<X>'s settings are copied to LANE<Y>, LANE<Y> will attempt to use EQUalized<Y-1> as its source. The only case in which this does not work is if Y is 1, in which case LANE<Y>'s source is unchanged from LANE<X>.

See Also

- ":LANE<N>:SOURce" on page 816

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:ACGain

Command

```
:LANE<N>:EQUalizer:CTLE:ACGain <ac_gain>
```

The :LANE<N>:EQUalizer:CTLE:ACGain command sets the AC Gain parameter for the Continuous Time Linear Equalization when USB31 is selected for the "# of Poles" option.

<ac_gain> A real number

Example This example sets the CTLE AC Gain parameter to 1.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:ACGain 1"
```

Query

```
:LANE<N>:EQUalizer:CTLE:ACGain?
```

The :LANE<N>:EQUalizer:CTLE:ACGain? query returns the CTLE's AC Gain parameter setting.

See Also

- ":LANE<N>:EQUalizer:CTLE:NUMPoles" on page 768

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:DBACgain

Command

```
:LANE<N>:EQUalizer:CTLE:DBACgain <ac_gain>
```

The :LANE<N>:EQUalizer:CTLE:DBACgain command sets the AC Gain parameter in dB for the Continuous Time Linear Equalization when USB31 is selected for the "# of Poles" option.

<ac_gain> A real number

Example This example sets the CTLE AC Gain parameter to 1.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:DBACgain 1"
```

Query

```
:LANE<N>:EQUalizer:CTLE:DBACgain?
```

The :LANE<N>:EQUalizer:CTLE:DBACgain? query returns the CTLE's AC Gain parameter in dB setting.

See Also

- ":LANE<N>:EQUalizer:CTLE:ACGain" on page 761
- ":LANE<N>:EQUalizer:CTLE:NUMPoles" on page 768

History New in version 11.10.

:LANE<N>:EQUalizer:CTLE:DBDCG2

Command

```
:LANE<N>:EQUalizer:CTLE:DBDCG2 <dc_gain_2>
```

The :LANE<N>:EQUalizer:CTLE:DBDCG2 command sets the DC Gain 2 parameter in dB when "2 Pole, 2 Gain" is selected for the "# of Poles" option.

<dc_gain_2> A real number.

Query

```
:LANE<N>:EQUalizer:CTLE:DBDCG2?
```

The :LANE<N>:EQUalizer:CTLE:DBDCG2? query returns the CTLE's DC Gain 2 parameter in dB setting.

Returned Format

```
<dc_gain_2><NL>
```

See Also

- ":LANE<N>:EQUalizer:CTLE:DCGain2" on page 766
- ":LANE<N>:EQUalizer:CTLE:LF" on page 767
- ":LANE<N>:EQUalizer:CTLE:DCGain" on page 765
- ":LANE<N>:EQUalizer:CTLE:Z1" on page 778
- ":LANE<N>:EQUalizer:CTLE:P1" on page 770
- ":LANE<N>:EQUalizer:CTLE:P2" on page 771

History New in version 11.10.

:LANE<N>:EQUalizer:CTLE:DBDCgain

Command

```
:LANE<N>:EQUalizer:CTLE:DBDCgain <dc_gain>
```

The :LANE<N>:EQUalizer:CTLE:DBDCgain command sets the DC Gain parameter in dB for the Continuous Time Linear Equalization.

<dc_gain> A real number

Example This example sets the CTLE DC Gain parameter to 1.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:DBDCgain 1"
```

Query

```
:LANE<N>:EQUalizer:CTLE:DBDCgain?
```

The :LANE<N>:EQUalizer:CTLE:DBDCgain? query returns the CTLE's DC Gain parameter in dB.

See Also

- ":LANE<N>:EQUalizer:CTLE:DCGain" on page 765

History New in version 11.10.

:LANE<N>:EQUalizer:CTLE:DCGain

Command

```
:LANE<N>:EQUalizer:CTLE:DCGain <dc_gain>
```

The :LANE<N>:EQUalizer:CTLE:DCGain command sets the DC Gain parameter for the Continuous Time Linear Equalization.

<dc_gain> A real number

Example This example sets the CTLE DC Gain parameter to 1.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:DCGain 1"
```

Query

```
:LANE<N>:EQUalizer:CTLE:DCGain?
```

The :LANE<N>:EQUalizer:CTLE:DCGain? query returns the CTLE's DC Gain parameter.

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:DCGain2

Command

```
:LANE<N>:EQUalizer:CTLE:DCGain2 <dc_gain_2>
```

The :LANE<N>:EQUalizer:CTLE:DCGain2 command sets the DC Gain 2 parameter when "2 Pole, 2 Gain" is selected for the "# of Poles" option.

<dc_gain_2> A real number.

Query

```
:LANE<N>:EQUalizer:CTLE:DCGain2?
```

The :LANE<N>:EQUalizer:CTLE:DCGain2? query returns the CTLE's DC Gain 2 parameter setting.

Returned Format

```
<dc_gain_2><NL>
```

See Also

- ":LANE<N>:EQUalizer:CTLE:LF" on page 767
- ":LANE<N>:EQUalizer:CTLE:DCGain" on page 765
- ":LANE<N>:EQUalizer:CTLE:Z1" on page 778
- ":LANE<N>:EQUalizer:CTLE:P1" on page 770
- ":LANE<N>:EQUalizer:CTLE:P2" on page 771

History New in version 10.25.

:LANE<N>:EQUalizer:CTLE:LF

Command

```
:LANE<N>:EQUalizer:CTLE:LF <LF_frequency>
```

The :LANE<N>:EQUalizer:CTLE:LF command sets the LF Frequency parameter when "2 Pole, 2 Gain" is selected for the "# of Poles" option.

<LF_frequency> A real number.

Query

```
:LANE<N>:EQUalizer:CTLE:LF?
```

The :LANE<N>:EQUalizer:CTLE:LF? query returns the CTLE's LF Frequency parameter setting.

Returned Format

```
<LF_frequency><NL>
```

See Also

- ":LANE<N>:EQUalizer:CTLE:DCGain" on page 765
- ":LANE<N>:EQUalizer:CTLE:DCGain2" on page 766
- ":LANE<N>:EQUalizer:CTLE:Z1" on page 778
- ":LANE<N>:EQUalizer:CTLE:P1" on page 770
- ":LANE<N>:EQUalizer:CTLE:P2" on page 771

History New in version 10.25.

:LANE<N>:EQUalizer:CTLE:NUMPoles

Command

```
:LANE<N>:EQUalizer:CTLE:NUMPoles {{P2Z1 | POLE2} | {P3Z1 | POLE3}
    | P4Z1 | P3Z2 | P6Z2 | {P2ACG | USB31} | LFG2}
```

The :LANE<N>:EQUalizer:CTLE:NUMPoles command selects from these Continuous Time Linear Equalizer (CTLE) options:

- {P2Z1 | POLE2} — 2 Pole, 1 Zero.
- {P3Z1 | POLE3} — 3 Pole, 1 Zero.
- P4Z1 — 4 Pole, 1 Zero.
- P3Z2 — 3 Pole, 2 Zeros.
- P6Z2 — 6 Pole, 2 Zeros.
- {P2ACG | USB31} — 2 Pole, AC Gain.
- LFG2 — 2 Pole, 2 Gain.

Example

This example selects a 2 Pole, 1 Zero CTLE.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:NUMPoles P2Z1"
```

Query

```
:LANE<N>:EQUalizer:CTLE:NUMPoles?
```

The :LANE<N>:EQUalizer:CTLE:NUMPoles? query returns the current "number of poles" selection.

Returned Format

```
[:LANE<N>:EQUalizer:CTLE:NUMPoles] {P2Z1 | P3Z1 | P4Z1 | P3Z2 | P6Z2
    | P2ACG | LFG2}
```

See Also

- ":LANE<N>:EQUalizer:CTLE:P1" on page 770
- ":LANE<N>:EQUalizer:CTLE:P2" on page 771
- ":LANE<N>:EQUalizer:CTLE:P3" on page 772
- ":LANE<N>:EQUalizer:CTLE:P4" on page 773
- ":LANE<N>:EQUalizer:CTLE:P5" on page 774
- ":LANE<N>:EQUalizer:CTLE:P6" on page 775
- ":LANE<N>:EQUalizer:CTLE:Z1" on page 778
- ":LANE<N>:EQUalizer:CTLE:Z2" on page 779
- ":LANE<N>:EQUalizer:CTLE:ACGain" on page 761
- ":LANE<N>:EQUalizer:CTLE:DCGain" on page 765
- ":LANE<N>:EQUalizer:CTLE:DCGain2" on page 766
- ":LANE<N>:EQUalizer:CTLE:LF" on page 767

History

New in version 10.20.

Version 10.25: Added the LFG2 (2 Pole, 2 Gain) option for the "number of poles" selection.

Version 11.20: Added the P6Z2 (6 Pole, 2 Zeros) option for the "number of poles" selection.

:LANE<N>:EQUalizer:CTLE:P1

Command

```
:LANE<N>:EQUalizer:CTLE:P1 <pole1_freq>
```

The :LANE<N>:EQUalizer:CTLE:P1 command sets the Pole 1 frequency for the Continuous Time Linear Equalization.

<pole1_freq> A real number

Example This example sets the CTLE Pole 1 frequency to 1GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P1 1e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P1?
```

The :LANE<N>:EQUalizer:CTLE:P1? query returns the CTLE's Pole 1 frequency.

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:P2

Command

```
:LANE<N>:EQUalizer:CTLE:P2 <pole2_freq>
```

The :LANE<N>:EQUalizer:CTLE:P2 command sets the Pole 2 frequency for the Continuous Time Linear Equalization.

<pole2_freq> A real number

Example This example sets the CTLE Pole 2 frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P2 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P2?
```

```
The :LANE<N>:EQUalizer:CTLE:P2? query returns the CTLE's Pole 2 frequenc
y.
```

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:P3

Command

```
:LANE<N>:EQUalizer:CTLE:P3 <pole3_freq>
```

The :LANE<N>:EQUalizer:CTLE:P3 command sets the Pole 3 frequency for the Continuous Time Linear Equalization.

<pole3_freq> A real number

Example This example sets the CTLE Pole 3 frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P3 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P3?
```

The :LANE<N>:EQUalizer:CTLE:P3? query returns the CTLE's Pole 3 frequency.

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:P4

Command

```
:LANE<N>:EQUalizer:CTLE:P4 <pole4_freq>
```

The :LANE<N>:EQUalizer:CTLE:P4 command sets the Pole 4 frequency for the Continuous Time Linear Equalization.

<pole4_freq> A real number

Example This example sets the CTLE Pole 4 frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P4 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P4?
```

The :LANE<N>:EQUalizer:CTLE:P4? query returns the CTLE's Pole 4 frequency.

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:P5

Command

```
:LANE<N>:EQUalizer:CTLE:P5 <pole5_freq>
```

The :LANE<N>:EQUalizer:CTLE:P5 command sets the Pole 5 frequency for the Continuous Time Linear Equalization.

<pole5_freq> A real number

Example This example sets the CTLE Pole 5 frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P5 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P5?
```

The :LANE<N>:EQUalizer:CTLE:P5? query returns the CTLE's Pole 5 frequency.

History New in version 11.15.

:LANE<N>:EQUalizer:CTLE:P6

Command

```
:LANE<N>:EQUalizer:CTLE:P6 <pole6_freq>
```

The :LANE<N>:EQUalizer:CTLE:P6 command sets the Pole 6 frequency for the Continuous Time Linear Equalization.

<pole6_freq> A real number

Example This example sets the CTLE Pole 6 frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:P6 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:P6?
```

The :LANE<N>:EQUalizer:CTLE:P6? query returns the CTLE's Pole 6 frequency.

History New in version 11.15.

:LANE<N>:EQUalizer:CTLE:RATE

Command

```
:LANE<N>:EQUalizer:CTLE:RATE <data_rate>
```

The :LANE<N>:EQUalizer:CTLE:RATE command sets the data rate for the CTLE equalizer.

<data_rate> A real number.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

Example This example sets the CTLE data rate to 3e9.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:RATE 3e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:RATE?
```

The :LANE<N>:EQUalizer:CTLE:RATE? query returns the CTLE's data rate.

See Also

- ":ANALyze:SIGNal:TYPE" on page 404

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:STATe

Command

```
:LANE<N>:EQUalizer:CTLE:STATe {(OFF | 0) | (ON | 1)}
```

The :LANE<N>:EQUalizer:CTLE:STATe command turns the Continuous Time Linear Equalizer (CTLE) on or off.

Example This example turns on CTLE.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:STATe ON"
```

Query

```
:LANE<N>:EQUalizer:CTLE:STATe?
```

The :LANE<N>:EQUalizer:CTLE:STATe? query returns whether or not CTLE is turned on.

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:Z1

Command

```
:LANE<N>:EQUalizer:CTLE:Z1 <zero_freq_1>
```

The :LANE<N>:EQUalizer:CTLE:Z1 command sets the first zero frequency for the 3-pole Continuous Time Linear Equalization.

<zero_freq_1> A real number in NR3 format.

Example This example sets the 3-pole CTLE's first zero frequency to 900 MHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:Z1 650e6"
```

Query

```
:LANE<N>:EQUalizer:CTLE:Z1?
```

The :LANE<N>:EQUalizer:CTLE:Z1? query returns the 3-pole CTLE's first zero frequency.

Returned Format

```
<zero_freq_1><NL>
```

See Also

- ":LANE<N>:EQUalizer:CTLE:Z2" on page 779
- ":LANE<N>:EQUalizer:CTLE:NUMPoles" on page 768

History New in version 10.20.

:LANE<N>:EQUalizer:CTLE:Z2

Command

```
:LANE<N>:EQUalizer:CTLE:Z2 <zero_freq_2>
```

The :LANE<N>:EQUalizer:CTLE:Z2 command sets the second zero frequency for the 3-pole Continuous Time Linear Equalization.

<zero_freq_2> A real number in NR3 format.

Example This example sets the 3-pole CTLE's second zero frequency to 4 GHz.

```
myScope.WriteString ":LANE1:EQUalizer:CTLE:Z2 4e9"
```

Query

```
:LANE<N>:EQUalizer:CTLE:Z2?
```

The :LANE<N>:EQUalizer:CTLE:Z2? query returns the 3-pole CTLE's second zero frequency.

Returned Format

```
<zero_freq_2><NL>
```

See Also

- ":LANE<N>:EQUalizer:CTLE:Z1" on page 778
- ":LANE<N>:EQUalizer:CTLE:NUMPoles" on page 768

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:NTAPs

Command

```
:LANE<N>:EQUalizer:DFE:NTAPs <number>
```

The :LANE<N>:EQUalizer:DFE:NTAPs command sets the number of taps to be used in the DFE algorithm.

DFE tap indices always begin with 1 and extend to the number of taps.

<number> An integer between 1 and 40

Example This example sets the number of DFE taps to 3.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:NTAPs 3"
```

Query

```
:LANE<N>:EQUalizer:DFE:NTAPs?
```

The :LANE<N>:EQUalizer:DFE:NTAPs? query returns the number of DFE taps.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:STATe

Command

```
:LANE<N>:EQUalizer:DFE:STATe {(OFF | 0) | (ON | 1)}
```

The :LANE<N>:EQUalizer:DFE:STATe command turns the Decision Feedback Equalization on or off.

Example This example turns on DFE.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:STATe ON"
```

Query

```
:LANE<N>:EQUalizer:DFE:STATe?
```

The :LANE<N>:EQUalizer:DFE:STATe? query returns whether or not DFE is turned on.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP

Command

```
:LANE<N>:EQUalizer:DFE:TAP <tap>, <value>
```

The :LANE<N>:EQUalizer:DFE:TAP command sets the tap value for each DFE tap. For example, when <tap> is equal to 1 then the 1st tap is set to <value>.

DFE tap indices always start at 1 and extend to the number of taps.

<tap> The tap number.

<value> The tap value

Example This example sets the DFE Tap 1 to 0.432.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP 1,0.432"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP? <tap>
```

The :LANE<N>:EQUalizer:DFE:TAP? query returns the DFE tap values.

See Also

- ":LANE<N>:EQUalizer:DFE:NTAPs" on page 780

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:ALGorithm

Command

```
:LANE<N>:EQUalizer:DFE:TAP:ALGorithm {LSEF | PMAX}
```

The :LANE<N>:EQUalizer:DFE:TAP:ALGorithm command sets the DFE tap optimization algorithm:

- LSEF — Least Squared Error Fit.

  This is the traditional DFE tap optimization algorithm. This tap optimization is able to run on open eyes only.

- PMAX — PMAX-Normalized (Pulse Response).

  When performing equalization on a PAM-4 signal (see :ANALyze:SIGNal:TYPE), you can choose this DFE tap optimization algorithm. Because a specific signal pattern like PRBS13Q is required for this algorithm, the tap optimization is able to run with closed eyes.

  The PMAX-Normalized DFE tap optimization algorithm uses a linear fit pulse response filter, and you can specify these filter parameters:

  - Linear Fit Pulse Length — See :LANE<N>:EQUalizer:DFE:TAP:LFPLength.
  - Linear Fit Pulse Delay — See :LANE<N>:EQUalizer:DFE:TAP:LFPDelay.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:ALGorithm?
```

The :LANE<N>:EQUalizer:DFE:TAP:ALGorithm? query returns the selected DFE tap optimization algorithm setting.

Returned Format

```
<algorithm><NL>

<algorithm> ::= {LSEF | PMAX}
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:SEED" on page 796

History New in version 10.25.

:LANE<N>:EQUalizer:DFE:TAP:AUTomatic

Command

```
:LANE<N>:EQUalizer:DFE:TAP:AUTomatic
```

The :LANE<N>:EQUalizer:DFE:TAP:AUTomatic command starts the DFE tap optimization. Be sure to first specify the number of taps, the max/min tap values, and the Normalize DC Gain setting.

Example This example starts the DFE tap optimization.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:AUTomatic"
```

See Also

- ":LANE<N>:EQUalizer:DFE:NTAPs" on page 780
- ":LANE<N>:EQUalizer:DFE:TAP:MIN" on page 793
- ":LANE<N>:EQUalizer:DFE:TAP:MAX" on page 791
- ":LANE<N>:EQUalizer:DFE:TAP:NORMalize" on page 795

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:DELay

Command

```
:LANE<N>:EQUalizer:DFE:TAP:DELay <delay>
```

The :LANE<N>:EQUalizer:DFE:TAP:DELay command specifies a delay of the DFE equalized waveform relative to an explicit recovered clock in order to center the DFE eye, post equalization.

You can automatically set the tap delay to center the DFE eye by using the :LANE<N>:EQUalizer:DFE:TAP:DELay:AUTomatic command.

You can also affect the DFE eye location when creating the DFE equalized waveform by using the :LANE<N>:EQUalizer:DFE:THReshold:DELay command to delay the DFE decision threshold.

<delay> The delay value in NR3 (real number) format.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:DELay?
```

The :LANE<N>:EQUalizer:DFE:TAP:DELay? query returns the value of the specified DFE tap delay.

Returned Format

```
<delay><NL>

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

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:DELay:AUTomatic" on page 786
- ":LANE<N>:EQUalizer:DFE:THReshold:DELay" on page 802

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:DELay:AUTomatic

Command `:LANE<N>:EQUalizer:DFE:TAP:DELay:AUTomatic`

The :LANE<N>:EQUalizer:DFE:TAP:DELay:AUTomatic command computes a DFE delay value to center a DFE eye on the screen horizontally. The current real-time eye data is used to center the DFE eye.

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:DELay" on page 785

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:GAIN

Command

```
:LANE<N>:EQUalizer:DFE:TAP:GAIN <gain>
```

The eye diagram drawn after DFE is applied is attenuated. To amplify the eye back to its original size (so you can directly compare the eye at the receiver to the eye at the transmitter), a gain factor needs to be applied. The :LANE<N>:EQUalizer:DFE:TAP:GAIN command allows you to set this gain. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<gain> A real number

Example This example sets the gain to 3.23.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:GAIN 3.23"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:GAIN?
```

The :LANE<N>:EQUalizer:DFE:TAP:GAIN? query returns the current gain value.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:LFPDelay

Command `:LANE<N>:EQUalizer:DFE:TAP:LFPDelay <integer>`

The :LANE<N>:EQUalizer:DFE:TAP:LFPDelay command specifies the Linear Fit Pulse Delay parameter when using the PMAX-Normalized DFE tap optimization algorithm (see :LANE<N>:EQUalizer:DFE:TAP:ALGorithm).

The PMAX-Normalized DFE tap optimization algorithm uses a linear fit pulse response filter and requires a specific signal pattern.

Test documents that require a PRBS13Q or other specific signal pattern may suggest values for this filter parameter, which is typically denoted using D_p.

<integer> An integer for the Linear Fit Pulse Delay from 0-20.

Query `:LANE<N>:EQUalizer:DFE:TAP:LFPDelay?`

The :LANE<N>:EQUalizer:DFE:TAP:LFPDelay? query returns the Linear Fit Pulse Delay.

Returned Format `<integer><NL>`

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:ALGorithm" on page 783
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789

History New in version 11.30.

:LANE<N>:EQUalizer:DFE:TAP:LFPLength

Command

```
:LANE<N>:EQUalizer:DFE:TAP:LFPLength <integer>
```

The :LANE<N>:EQUalizer:DFE:TAP:LFPLength command specifies the Linear Fit Pulse Length parameter when using the PMAX-Normalized DFE tap optimization algorithm (see :LANE<N>:EQUalizer:DFE:TAP:ALGorithm).

The PMAX-Normalized DFE tap optimization algorithm uses a linear fit pulse response filter and requires a specific signal pattern.

Test documents that require a PRBS13Q or other specific signal pattern may suggest values for this filter parameter, which is typically denoted using N_p.

<integer> An integer for the Linear Fit Pulse Length from 2-500.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:LFPLength?
```

The :LANE<N>:EQUalizer:DFE:TAP:LFPLength? query returns the Linear Fit Pulse Length.

Returned Format

```
<integer><NL>
```

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:ALGorithm" on page 783
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788

History New in version 11.30.

:LANE<N>:EQUalizer:DFE:TAP:LTARget

Command

```
:LANE<N>:EQUalizer:DFE:TAP:LTARget <lower_target>
```

The Lower Target field dictates the logical low value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical low, its value will be equal to Lower Target. The :LANE<N>:EQUalizer:DFE:TAP:LTARget command allows you to set this value.

<lower_target> A real number

Example This example sets the Lower Target to 1.0.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:LTARget 1.0"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:LTARget?
```

The :LANE<N>:EQUalizer:DFE:TAP:LTARget? query returns the current value for the Lower Target field.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:MAX

Command

```
:LANE<N>:EQUalizer:DFE:TAP:MAX <max_tap_value>[, <tap>]
```

Some standards have upper and lower limits on the tap values. The :LANE<N>:EQUalizer:DFE:TAP:MAX command sets the upper limit on taps determined through optimization.

<max_tap_value> A real number.

<tap> A one-based tap index. If <tap> is not specified, then all taps will have the limit.

Example This example sets the Upper Limit field to 3.23.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:MAX 3.23"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:MAX? [<tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MAX? query returns the Upper Limit used in the DFE tap optimization.

If <tap> is not specified, the Tap 1 limit is returned.

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:MAXV" on page 792
- ":LANE<N>:EQUalizer:DFE:TAP:MIN" on page 793
- ":LANE<N>:EQUalizer:DFE:TAP:MINV" on page 794
- ":LANE<N>:EQUalizer:DFE:TAP:STATus" on page 797
- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 10.20.

Version 10.25: The max value can now be specified for individual taps, and the query can return individual tap values.

:LANE<N>:EQUalizer:DFE:TAP:MAXV

Command

```
:LANE<N>:EQUalizer:DFE:TAP:MAXV <max_tap_value_in_volts>[, <tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MAXV command sets the maximum tap value for DFE auto tap setup in volts as opposed to the :LANE<N>:EQUalizer:DFE:TAP:MAX command that sets the max in unitless values.

If the unitless values are changed by the :LANE<N>:EQUalizer:DFE:TAP:MAX command, they supersede the voltage values.

<max_tap_value_in_volts> A real number.

<tap> A one-based tap index. If <tap> is not specified, then all taps will have the limit.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:MAXV? [<tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MAXV? query returns the maximum tap value in volts used in the DFE auto tap setup.

If <tap> is not specified, the Tap 1 limit is returned.

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:MAX" on page 791
- ":LANE<N>:EQUalizer:DFE:TAP:MIN" on page 793
- ":LANE<N>:EQUalizer:DFE:TAP:MINV" on page 794
- ":LANE<N>:EQUalizer:DFE:TAP:STATus" on page 797
- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 10.20.

Version 10.25: The max (in volts) value can now be specified for individual taps, and the query can return individual tap values.

:LANE<N>:EQUalizer:DFE:TAP:MIN

Command

```
:LANE<N>:EQUalizer:DFE:TAP:MIN <min_tap_value>[, <tap>]
```

Some standards have upper and lower limits on the tap values. The :LANE<N>:EQUalizer:DFE:TAP:MIN command sets the lower limit on taps determined through optimization.

<min_tap_value> A real number.

<tap> A one-based tap index. If <tap> is not specified, then all taps will have the limit.

Example This example sets the Lower Limit field to 3.23.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:MIN 3.23"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:MIN? [<tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MIN? query returns the Lower Limit used in the DFE tap optimization.

If <tap> is not specified, the Tap 1 limit is returned.

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:MAX" on page 791
- ":LANE<N>:EQUalizer:DFE:TAP:MAXV" on page 792
- ":LANE<N>:EQUalizer:DFE:TAP:MINV" on page 794
- ":LANE<N>:EQUalizer:DFE:TAP:STATus" on page 797
- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 10.20.

Version 10.25: The max value can now be specified for individual taps, and the query can return individual tap values.

:LANE<N>:EQUalizer:DFE:TAP:MINV

Command

```
:LANE<N>:EQUalizer:DFE:TAP:MINV <min_tap_value_in_volts>[, <tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MINV command sets the minimum tap value for DFE auto tap setup in volts as opposed to the :LANE<N>:EQUalizer:DFE:TAP:MIN command that sets the min in unitless values.

If the unitless values are changed by the :LANE<N>:EQUalizer:DFE:TAP:MIN command, they supersede the voltage values.

<min_tap_value_in_volts> A real number.

<tap> A one-based tap index. If <tap> is not specified, then all taps will have the limit.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:MINV? [<tap>]
```

The :LANE<N>:EQUalizer:DFE:TAP:MINV? query returns the minimum tap value in volts used in the DFE auto tap setup.

If <tap> is not specified, the Tap 1 limit is returned.

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:MAX" on page 791
- ":LANE<N>:EQUalizer:DFE:TAP:MAXV" on page 792
- ":LANE<N>:EQUalizer:DFE:TAP:MIN" on page 793
- ":LANE<N>:EQUalizer:DFE:TAP:STATus" on page 797
- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 10.20.

Version 10.25: The min (in volts) tap value can now be specified for individual taps, and the query can return individual tap values.

:LANE<N>:EQUalizer:DFE:TAP:NORMalize

Command

```
:LANE<N>:EQUalizer:DFE:TAP:NORMalize {{0 | OFF} | {1 | ON}}
```

The :LANE<N>:EQUalizer:DFE:TAP:NORMalize command specifies whether the Normalize DC Gain setting is ON or OFF. When ON, the eye diagram is automatically scaled so that it is the same size as the transmitted eye.

the Normalize DC Gain setting should be set (if desired) prior to calling the :LANE<N>:EQUalizer:DFE:TAP:AUTOmatic command.

This command maps to the **Normalize DC Gain** setting in the Equalization Auto Tap Setup dialog box in the front panel graphical user interface.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:NORMalize?
```

The :LANE<N>:EQUalizer:DFE:TAP:NORMalize? query returns the Normalize DC Gain setting.

Returned Format

```
<setting><NL>

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

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:SEED

Command

```
:LANE<N>:EQUalizer:DFE:TAP:SEED {OFF | ON}[, <value>]
```

When you select the PMAX-Normalized DFE tap optimization algorithm (see :LANE<N>:EQUalizer:DFE:TAP:ALGorithm), the :LANE<N>:EQUalizer:DFE:TAP:SEED command lets you specify whether taps should be seeded and optionally what seed value to use.

Normally DFE will exclude the first several symbols from the real-time eye because they are used to seed the DFE. For example, if you are using a 4-tap DFE, the first four UIs are not included in the real-time eye.

However, if you want to include all of the unit intervals in the real-time eye, you can enable the **Seed DFE Taps** setting. When **Seed DFE Taps** is enabled, initial symbols are all seeded to the <value> you enter so that the DFE can start working on the first UI.

DFE tap seeding is useful for DDR buses that turn on and off frequently.

<value> A 0 or 1 for an NRZ signal; a 0, 1, 2, or 3 for a PAM-4 signal.

Query

```
:LANE<N>:EQUalizer:DFE:TAP:SEED?
```

The :LANE<N>:EQUalizer:DFE:TAP:SEED? query returns the **Seed DFE Taps** settings.

Returned Format

```
{0 | 1},<value><NL>
```

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:ALGorithm" on page 783

History New in version 11.30.

:LANE<N>:EQUalizer:DFE:TAP:STATus

Query

```
:LANE<N>:EQUalizer:DFE:TAP:STATus? <tap_number>
```

You can specify limits on the tap coefficients with the :LANE<N>:EQUalizer:DFE:TAP:MAX, :LANE<N>:EQUalizer:DFE:TAP:MAXV, :LANE<N>:EQUalizer:DFE:TAP:MIN, and :LANE<N>:EQUalizer:DFE:TAP:MINV commands.

If the optimization process (:LANE<N>:EQUalizer:DFE:TAP:AUTomatic) would choose a tap value that would be clamped by the limits, the :LANE<N>:EQUalizer:DFE:TAP:STATus? query returns HIGH to indicate the tap value would be clamped high or LOW to indicate the tap value would be clamped low. If the optimization process would choose a tap value that is within the limits, the :LANE<N>:EQUalizer:DFE:TAP:STATus? query returns VAL to indicate a valid tap value.

<tap_number> A one-based tap index integer number.

Returned Format

```
<status><NL>

<status> ::= {VAL | HIGH | LOW}
```

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:MAX" on page 791
- ":LANE<N>:EQUalizer:DFE:TAP:MAXV" on page 792
- ":LANE<N>:EQUalizer:DFE:TAP:MIN" on page 793
- ":LANE<N>:EQUalizer:DFE:TAP:MINV" on page 794
- ":LANE<N>:EQUalizer:DFE:TAP:AUTomatic" on page 784

History New in version 11.25.

:LANE<N>:EQUalizer:DFE:TAP:UTARget

Command

```
:LANE<N>:EQUalizer:DFE:TAP:UTARget <upper_target>
```

The Upper Target field dictates the logical high value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical high, its value will be equal to Upper Target. The :LANE<N>:EQUalizer:DFE:TAP:UTARget command allows you to set this value.

<upper_target> A real number

Example This example sets the Upper Target to 1.0.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:UTARget 1.0"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:UTARget?
```

The :LANE<N>:EQUalizer:DFE:TAP:UTARget? query returns the current value for the Upper Target field.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:TAP:WIDTh

Command

```
:LANE<N>:EQUalizer:DFE:TAP:WIDTh <width>
```

The :LANE<N>:EQUalizer:DFE:TAP:WIDTh command sets the Eye Width field for the DFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for DFE is 0.0. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":LANE1:EQUalizer:DFE:TAP:WIDTh 0.0"
```

Query

```
:LANE<N>:EQUalizer:DFE:TAP:WIDTh?
```

The :LANE<N>:EQUalizer:DFE:TAP? query returns the eye width used in the DFE tap optimization.

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:THReshold:BANDwidth

Command

```
:LANE<N>:EQUalizer:DFE:THReshold:BANDwidth <bw_value>
```

When the DFE threshold bandwidth mode is set to CUSTom (by the :LANE<N>:EQUalizer:DFE:THReshold:BWMode command), the :LANE<N>:EQUalizer:DFE:THReshold:BANDwidth command specifies the threshold bandwidth value.

<bw_value> A real number.

Query

```
:LANE<N>:EQUalizer:DFE:THReshold:BANDwidth?
```

The :LANE<N>:EQUalizer:DFE:THReshold:BANDwidth? query returns the custom threshold bandwidth value.

See Also

- ":LANE<N>:EQUalizer:DFE:THReshold:BWMode" on page 801

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:THReshold:BWMode

Command `:LANE<N>:EQUalizer:DFE:THReshold:BWMode {OFF | CUSTom | TSBandwidth}`

When lane equalization is being displayed as a function (:LANE<N>:EQUalizer:LOCation FUNCtion), the :LANE<N>:EQUalizer:DFE:THReshold:BWMode command sets the threshold bandwidth mode for the DFE:

- TSBandwidth — Tracks the bandwidth limit of the source waveform.
- CUSTom — Use the :LANE<N>:EQUalizer:DFE:THReshold:BANDwidth command to specify the custom bandwidth value.
- OFF

Query `:LANE<N>:EQUalizer:DFE:THReshold:BWMode?`

The :LANE<N>:EQUalizer:DFE:THReshold:BWMode? query returns the current setting of the threshold bandwidth mode.

Returned Format `<thr_bw_mode><NL>`

`<thr_bw_mode> ::= {OFF | CUST | TSB}`

See Also

- ":LANE<N>:EQUalizer:LOCation" on page 815
- ":LANE<N>:EQUalizer:DFE:THReshold:BANDwidth" on page 800

History New in version 10.20.

:LANE<N>:EQUalizer:DFE:THReshold:DELay

Command

```
:LANE<N>:EQUalizer:DFE:THReshold:DELay <threshold_delay>
```

The :LANE<N>:EQUalizer:DFE:THReshold:DELay command sets a delay to move the decision threshold relative to the original waveform when creating the DFE equalized waveform.

This command is in contrast to the :LANE<N>:EQU:DFE:TAP:DELay command that moves the DFE equalized waveform relative to an explicit recovered clock in order to center the DFE eye post equalization.

<threshold_delay> The delay value in NR3 (real number) format.

Query

```
:LANE<N>:EQUalizer:DFE:THReshold:DELay?
```

The :LANE<N>:EQUalizer:DFE:THReshold:DELay? query returns the decision threshold delay value.

Returned Format

```
<threshold_delay><NL>

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

See Also

- ":LANE<N>:EQUalizer:DFE:TAP:DELay" on page 785

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:BANDwidth

Command

```
:LANE<N>:EQUalizer:FFE:BANDwidth <bandwidth>
```

The :LANE<N>:EQUalizer:FFE:BANDwidth command is only needed if the :LANE<N>:EQUalizer:FFE:BWMode command is set to CUSTom and in this case it sets the bandwidth at which the response generated by equalization rolls off. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

<bandwidth> The bandwidth at which the response generated by equalization rolls off.

Query

```
:LANE<N>:EQUalizer:FFE:BANDwidth?
```

The :LANE<N>:EQUalizer:FFE:BANDwidth? query returns the current value for the BANDwidth parameter.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:BWMode

Command

```
:LANE<N>:EQUalizer:FFE:BWMode {TSBandwidth | TTDelay | CUSTom}
```

The :LANE<N>:EQUalizer:FFE:BWMode command sets the bandwidth at which the response generated by equalization is rolled off. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

Example This example sets the FFE Bandwidth Mode to TTDelay.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:BWMode TTDelay"
```

Query

```
:LANE<N>:EQUalizer:FFE:BWMode?
```

The :LANE<N>:EQUalizer:FFE:BWMode? query returns the FFE Bandwidth Mode.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:NPRecursor

Command

```
:LANE<N>:EQUalizer:FFE:NPRecursor <number>
```

The :LANE<N>:EQUalizer:FFE:NPRecursor command sets the number of precursor taps to be used in the FFE algorithm.

<number> An integer between 1 and (NTAPs - 1)

Example This example sets the number of FFE precursor taps to 3.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:NPRecursor 3"
```

Query

```
:LANE<N>:EQUalizer:FFE:NPRecursor?
```

The :LANE<N>:EQUalizer:FFE:NPRecursor? query returns the number of FFE precursor taps.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:NTAPs

Command

```
:LANE<N>:EQUalizer:FFE:NTAPs <number>
```

The :LANE<N>:EQUalizer:FFE:NTAPs command sets the number of taps to be used in the FFE algorithm.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<number> an integer between 2 and 40

Example This example sets the number of FFE taps to 3.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:NTAPs 3"
```

Query

```
:LANE<N>:EQUalizer:FFE:NTAPs?
```

The :LANE<N>:EQUalizer:FFE:NTAPs? query returns the number of FFE taps.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:RATE

Command

```
:LANE<N>:EQUalizer:FFE:RATE <data_rate>
```

The :LANE<N>:EQUalizer:FFE:RATE command sets the data rate for the FFE equalizer.

<data_rate> A real number.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

Example This example sets the FFE data rate to 3e9.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:RATE 3e9"
```

Query

```
:LANE<N>:EQUalizer:FFE:RATE?
```

```
The :LANE<N>:EQUalizer:FFE:RATE? query returns the FFE's data rate.
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:STATe

Command

```
:LANE<N>:EQUalizer:FFE:STATe {(OFF | 0) | (ON | 1)}
```

The :LANE<N>:EQUalizer:FFE:STATe command turns the Feed-Forward Equalized (FFE) on or off.

Example This example turns on FFE.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:STATe ON"
```

Query

```
:LANE<N>:EQUalizer:FFE:STATe?
```

The :LANE<N>:EQUalizer:FFE:STATe? query returns whether or not FFE is turned on.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TAP

Command

```
:LANE<N>:EQUalizer:FFE:TAP <tap>, <value>
```

The :LANE<N>:EQUalizer:FFE:TAP command sets the tap value for each FFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<tap> The tap number; when <tap> == 0, Tap 0 is set

<value> The tap value

Example This example sets the second FFE tap to -1.432.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:TAP 2,-1.432"
```

Query

```
:LANE<N>:EQUalizer:FFE:TAP? <tap>
```

The :LANE<N>:EQUalizer:FFE:TAP? query returns the FFE tap values.

See Also

- ":LANE<N>:EQUalizer:FFE:NTAPs" on page 806

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TAP:AUTomatic

Command

```
:LANE<N>:EQUalizer:FFE:TAP:AUTomatic
```

The :LANE<N>:EQUalizer:FFE:TAP:AUTomatic command starts the FFE tap optimization. Be sure to first specify the number of taps and specify the Pattern and Eye Width parameters.

Example This example starts the FFE tap optimization.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:TAP:AUTomatic"
```

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TAP:DELay

Command

```
:LANE<N>:EQUalizer:FFE:TAP:DELay <delay>
```

The :LANE<N>:EQUalizer:FFE:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<delay> A real number

Query

```
:LANE<N>:EQUalizer:FFE:TAP:DELay?
```

The :LANE<N>:EQUalizer:FFE:TAP:DELay? query returns the value for the FFE Delay field.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TAP:WIDTh

Command

```
:LANE<N>:EQUalizer:FFE:TAP:WIDTh <width>
```

The :LANE<N>:EQUalizer:FFE:TAP:WIDTh command sets the Eye Width field for the FFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for FFE is 0.33. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:TAP:WIDTh 0.0"
```

Query

```
:LANE<N>:EQUalizer:FFE:TAP:WIDTh?
```

The :LANE<N>:EQUalizer:FFE:TAP:WIDTh? query returns the eye width used in the FFE tap optimization.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TDELay

Command

```
:LANE<N>:EQUalizer:FFE:TDELay <delay_value>
```

The :LANE<N>:EQUalizer:FFE:TDELay command is only needed if the :LANE<N>:EQUalizer:FFE:TDMode is set to CUSTom. To determine what this value should be, use the equation: tap delay = 1/[(data rate)x(# of taps per bit)]. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

<delay_value> A real number

Query

```
:LANE<N>:EQUalizer:FFE:TDELay?
```

The :LANE<N>:EQUalizer:FFE:TDELay? query returns the current value for the tap delay.

History New in version 10.20.

:LANE<N>:EQUalizer:FFE:TDMode

Command

```
:LANE<N>:EQUalizer:FFE:TDMode {TBITrate | CUSTom}
```

The :LANE<N>:EQUalizer:FFE:TDMode command sets Tap Delay field to either Track Data Rate or Custom. If you are using one tap per bit, use the TBITrate selection. If you are using multiple taps per bit, use CUSTom and then use the :LANE<N>:EQUalizer:FFE:TDELay command to set the value. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

Example

This example sets the FFE Tap Delay mode to TBITrate.

```
myScope.WriteString ":LANE1:EQUalizer:FFE:TDMode TBITrate"
```

Query

```
:LANE<N>:EQUalizer:FFE:TDMode?
```

The :LANE<N>:EQUalizer:FFE:TDMode? query returns the current Tap Delay mode.

History

New in version 10.20.

:LANE<N>:EQUalizer:LOCation

Command
```
:LANE<N>:EQUalizer:LOCation {INPLace | FUNCtion}
```

The :LANE<N>:EQUalizer:LOCation command tells the equalization lane whether to equalize in-place (modifying the source waveform itself) or display as a function (creating a separate equalized waveform, which is what was done in the Infiniium oscilloscope software versions before 10.20).

For linear equalization (CTLE and FFE), "in-place" means the equalization runs completely in hardware, greatly improving speed. For DFE, "in-place" means the pre-10.20 version of DFE is performed (that is, the DFE that modifies the display of the real-time eye only).

Rules for In-Place Equalization

The following rules determine whether in-place equalization is legal, and can be used to explain interactions involving the "in-place" vs. "as-a-function" selection:

1. Of all running lanes using source <S>, only the lowest-numbered lane can equalize <S> in place.
2. In-place CTLE and FFE can be applied to analog channels only.
3. In-place DFE can be applied to all sources with a real-time eye displayed.

Query
```
:LANE<N>:EQUalizer:LOCation?
```

The :LANE<N>:EQUalizer:LOCation? query returns the location setting for the equalization lane.

Returned Format
```
[:LANE<N>:EQUalizer:LOCation] {INPL | FUNC}<NL>
```

See Also

- ":LANE<N>:SOURce" on page 816

History New in version 10.20.

:LANE<N>:SOURce

Command

```
:LANE<N>:SOURce <source>
```

The :LANE<N>:SOURce command sets the source for the equalization lane.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR | EQUalized<L>}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Chaining Equalization Lanes An integer, 1-3. LANE<N> can use as its source the equalization of the previous lane, EQUalized<N-1>, provided that the result of that equalization is displayed as a function. This lets you chain equalization lanes from one to the next. In other words, LANE<N> can be the source for LANE<N+1>.

If the LANE<N> equalization is "in-place", its source (for example, CHANnel1) can also be the source for LANE<N+1> but no other lanes, and LANE<N+1> cannot be done "in-place".

Example This example sets the first equalization lane source to Channel 1.

```
myScope.WriteString ":LANE1:SOURce CHANnel1"
```

Query

```
:LANE<N>:SOURce?
```

The :LANE<N>:SOURce? query returns the equalization lane's source.

See Also

- ":LANE<N>:EQUalizer:LOCation" on page 815

History New in version 10.20.

:LANE<N>:STATe

Command

```
:LANE<N>:STATe {(OFF | 0) | (ON | 1)}
```

The :LANE<N>:STATe command turns the equalization lane on or off. This command has no effect on the states of the three types of equalization available within the lane (CTLE, FFE, or DFE).

Example This example turns on equalization lane number one.

```
myScope.WriteString ":LANE1:STATe ON"
```

Query

```
:LANE<N>:STATe?
```

The :LANE<N>:STATe? query returns whether or not the equalization lane is turned on.

History New in version 10.20.

:LANE<N>:VERTical

Command

```
:LANE<N>:VERTical {AUTO | MANual}
```

The :LANE<N>:VERTical command sets the equalization lane's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the first equalization lane's vertical scale mode to automatic.

```
myScope.WriteString ":LANE1:VERTical AUTO"
```

Query

```
:LANE<N>:VERTical?
```

The :LANE<N>:VERTical? query returns the current equalization lane's vertical scale mode setting.

Returned Format

```
[:LANE<N>:VERTical] {AUTO | MAN}
```

History

New in version 10.20.

:LANE<N>:VERTical:OFFSet

Command

```
:LANE<N>:VERTical:OFFSet <offset>
```

The :LANE<N>:VERTical:OFFSet command sets the equalization lane's vertical offset.

<offset> A real number for the equalization lane's vertical offset.

Example This example sets the first equalization lane's vertical offset to 1 volt.

```
myScope.WriteString ":LANE1:VERTical:OFFSet 1"
```

Query

```
:LANE<N>:VERTical:OFFSet?
```

The:LANE<N>:VERTical:OFFSet? query returns the equalization lane's vertical offset setting.

Returned Format

```
[:LANE<N>:VERTical:OFFSet] <value><NL>
```

<value> The equalization lane's vertical offset setting.

History New in version 10.20.

:LANE<N>:VERTical:RANGe

Command
```
:LANE<N>:VERTical:RANGe <range>
```

The :LANE<N>:VERTical:RANGe command sets the equalization lane's vertical range.

<range> A real number for the full-scale equalization lane's vertical range.

Example This example sets the first equalization lane's vertical range to 8 volts (1 volts times 8 divisions.)

```
myScope.WriteString ":LANE1:VERTical:RANGe 8"
```

Query
```
:LANE<N>:VERTical:RANGe?
```

The :LANE<N>:VERTical:RANGe? query returns the equalization lane's vertical range setting.

Returned Format
```
[:LANE<N>:VERTical:RANGe] <value><NL>
```

<value> The equalization lane's vertical range setting.

History New in version 10.20.

30 :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?

Query

```
:LISTer:DATA? [{{SBUS1 | SBUS2 | SBUS3 | SBUS4} [,<type>] | BUS}]
```

The :LISTer:DATA? query returns the lister data (SBUS1, SBUS2, SBUS3, or SBUS4) or the Digital Listing data (BUS).

To turn on the Digital Listing, you need to enable one or more of the digital buses (see :BUS<B>:DISPlay).

<type> {PACKets | SYMBols | PAYLoad}

Specifies which display window to save.

Returned Format

```
<binary block><NL>

<binary_block> ::= comma-separated data with newlines at the
                   end of each row
```

See Also

- ":LISTer:DISPlay" on page 823
- ":BUS<B>:DISPlay" on page 417
- "Definite-Length Block Response Data" on page 123

History New in version 3.50.

Version 5.00: Added the <type> parameter for specifying which display window to save.

Version 11.20: Added the BUS parameter for returning Digital Listing data.

:LISTer:DISPlay

Command

```
:LISTer:DISPlay <value>

<value> ::= {OFF | ON | SBUS1 | SBUS2 | SBUS3 | SBUS4}
```

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

```
:LISTer:DISPlay?
```

The :LISTer:DISPlay? query returns the Lister display setting.

Returned Format

```
<value><NL>

<value> ::= {OFF | ON | SBUS1 | SBUS2 | SBUS3 | SBUS4}
```

See Also

- ":SBUS<N>[:DISPlay]" on page 1427
- ":LISTer:DATA?" on page 822

History

New in version 3.50.

31 :LTESt (Limit Test) Commands

The Limit Test commands and queries control the limit test features of the oscilloscope. Limit testing automatically compares measurement results with pass or fail limits. The limit test tracks up to 20 measurements. The action taken when the test fails is also controlled with commands in this subsystem.

:LTESt:ADDStats

Command `:LTESt:ADDStats {{0 | OFF} | {1 | ON}}`

The :LTESt:ADDStats command enables or disables the limit test "statistics on passing measurements only" option.

When ON, statistics are compiled on passing measurements only.

Query `:LTESt:ADDStats?`

The :LTESt:ADDStats? query returns the "statistics on passing measurements only" setting.

Returned Format `<setting><NL>`

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

See Also

- ":LTESt:RUMode:SOFailure" on page 832

History New in version 5.70.

:LTESt:FAIL

Command

```
:LTESt:FAIL {{INSide | OUTSide} | {COUNt | RANGe}}
```

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, OUTSide

These are the fail condition options for most measurements.

- 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.

COUNt, RANGe

When performing limit test on the BER (Per Acq) measurement (:MEASure:BERPeracq), the fail condition options are COUNt and RANGe instead of INSide and OUTSide.

- COUNt — causes the oscilloscope to fail a test when the BER (Bit Error Ratio) fail count occurs. In this case, the count is specified by the :LTESt:LLIMit command (and the :LTESt:ULIMit command does not apply).
- RANGe — causes the oscilloscope to fail a test when the BER (Bit Error Ratio) fail count occurs within a certain number of bits. In this case, the count is specified by the :LTESt:LLIMit command and the range of bits is specified by the :LTESt:ULIMit command. This option is useful for finding burst errors.

Example

The following example causes the oscilloscope to fail a test when the measurements are outside the lower and upper limits.

```
myScope.WriteString ":LTESt:FAIL OUTSide"
```

Query

```
:LTESt:FAIL?
```

The query returns the currently set fail condition.

Returned Format

```
[:LTESt:FAIL] {INSide | OUTSide}<NL>
```

Example

The following example returns the current fail condition and prints the result to the controller's screen.

```
Dim strFAIL As String
myScope.WriteString ":LTESt:FAIL?"
strFAIL = myScope.ReadString
Debug.Print strFAIL
```

See Also

- ":LTESt:LLIMit — Lower Limit" on page 829
- ":LTESt:MEASurement" on page 830
- ":LTESt:RESults?" on page 831
- ":LTESt:TEST" on page 833

- ":LTESt:ULIMit — Upper Limit" on page 834
- ":MEASure:BERPeracq" on page 882

History Legacy command (existed before version 3.10).

:LTESt:LLIMit — Lower Limit

Command

```
:LTESt:LLIMit <lower_value>
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:LLIMit (Lower LIMit) command sets the lower test limit.

<lower_value> A real number.

Example The following example sets the lower test limit to 1.0.

```
myScope.WriteString ":LTESt:LLIMit 1.0"
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the limit test to fail when the signal is outside the specified limit.

Query

```
:LTESt:LLIMit?
```

The query returns the current value set by the command.

Returned Format

```
[:LTESt:LLIMit]<lower_value><NL>
```

Example The following example returns the current lower test limit and prints the result to the controller's screen.

```
Dim strLLIM As String
myScope.WriteString ":LTESt:LLIMit?"
strLLIM = myScope.ReadString
Debug.Print strLLIM
```

See Also

- ":LTESt:FAIL" on page 827
- ":LTESt:MEASurement" on page 830
- ":LTESt:RESults?" on page 831
- ":LTESt:TEST" on page 833
- ":LTESt:ULIMit — Upper Limit" on page 834

History Legacy command (existed before version 3.10).

:LTESt:MEASurement

Command

```
:LTESt:MEASurement {MEAS<N>}
```

The :LTESt:MEASurement command selects the measurement source for the FAIL, LLIMit, ULIMit, and TEST commands. It selects one of the active measurements by its number, where MEAS1 is the most recently added measurement.

<N> An integer, 1-40.

Example The following example selects the first measurement as the source for the limit testing commands.

```
myScope.WriteString ":LTESt:MEASurement MEAS1"
```

Query

```
:LTESt:MEASurement?
```

The query returns the currently selected measurement source.

Returned Format

```
[:LTESt:MEASurement] {MEAS<N>} <NL>
```

Example The following example returns the currently selected measurement source for the limit testing commands.

```
Dim strSOURCE As String
myScope.WriteString ":LTESt:MEASurement?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also Measurements are started in the :MEASure subsystem.

See Also

- ":LTESt:FAIL" on page 827
- ":LTESt:LLIMit — Lower Limit" on page 829
- ":LTESt:RESults?" on page 831
- ":LTESt:TEST" on page 833
- ":LTESt:ULIMit — Upper Limit" on page 834

History Legacy command (existed before version 3.10).

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:LTESt:RESults?

Query `:LTESt:RESults? {MEAS<N>}`

The 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.

<N> An integer, 1-40.

Returned Format `[:LTESt:RESults] <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.

Example The following example returns the values for the limit test of measurement 1.

```
Dim strRESULTS As String
myScope.WriteString ":LTESt:RESults? MEAS1"
strRESULTS = myScope.ReadString
Debug.Print strRESULTS
```

See Also Measurements are started in the Measurement Subsystem.

See Also

- ":LTESt:FAIL" on page 827
- ":LTESt:LLIMit — Lower Limit" on page 829
- ":LTESt:MEASurement" on page 830
- ":LTESt:TEST" on page 833
- ":LTESt:ULIMit — Upper Limit" on page 834

History Legacy command (existed before version 3.10).

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:LTESt:RUMode:SOFailure

Command

```
: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

```
:LTESt:RUMode:SOFailure?
```

The :LTESt:RUMode:SOFailure? query returns the "stop on failure" setting.

Returned Format

```
<setting><NL>

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

See Also

- ":LTESt:ADDStats" on page 826

History

New in version 5.70.

:LTESt:TEST

Command

```
:LTESt:TEST {{ON | 1} | {OFF | 0}}
```

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.

Example

The following example turns off the limit test function for the active measurement currently selected by the :LTESt:MEASurement command.

```
myScope.WriteString ":LTESt:TEST OFF"
```

Query

```
:LTESt:TEST?
```

The query returns the state of the TEST control for the active measurement currently selected by the :LTESt:MEASurement command.

Returned Format

```
[:LTESt:TEST] {1 | 0} <NL>
```

Example

The following example returns the current state of the limit test and prints the result to the controller's screen.

```
Dim strTEST As String
myScope.WriteString ":LTESt:TEST?"
strTEST = myScope.ReadString
Debug.Print strTEST
```

See Also

- ":LTESt:FAIL" on page 827
- ":LTESt:LLIMit — Lower Limit" on page 829
- ":LTESt:MEASurement" on page 830
- ":LTESt:RESults?" on page 831
- ":LTESt:ULIMit — Upper Limit" on page 834

History

Legacy command (existed before version 3.10).

:LTESt:ULIMit — Upper Limit

Command

```
:LTESt:ULIMit <upper_value>
```

For the active measurement currently selected by the :LTESt:MEASurement command, the :LTESt:ULIMit (Upper LIMit) command sets the upper test limit.

<upper_value> A real number.

Example The following example sets the upper limit of the currently selected measurement to 500 mV.

```
myScope.WriteString ":LTESt:ULIMit 500E-3"
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the LTESt:FAIL OUTside command to specify that the limit subsystem will fail a measurement when the voltage exceeds 500 mV.

Query

```
:LTESt:ULIMit?
```

The query returns the current upper limit of the limit test.

Returned Format

```
[:LTESt:ULIMit] <upper_value><NL>
```

Example The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
Dim strULIM As String
myScope.WriteString ":LTESt:ULIMit?"
strULIM = myScope.ReadString
Debug.Print strULIM
```

See Also

- ":LTESt:FAIL" on page 827
- ":LTESt:LLIMit — Lower Limit" on page 829
- ":LTESt:MEASurement" on page 830
- ":LTESt:RESults?" on page 831
- ":LTESt:TEST" on page 833

History Legacy command (existed before version 3.10).

32 :LXI Commands

The :LXI commands and queries are common for instruments that support the LAN eXtensions for Instrumentation (LXI) standard.

:LXI:IDENtify[:STATe]

Command

```
:LXI:IDENtify[:STATe] {{0 | OFF} | {1 | ON}}
```

The :LXI:IDENtify[:STATe] command lets you identify an oscilloscope.

- ON — Opens the oscilloscope graphical user interface LXI LAN dialog box and flashes the green LXI logo.
- OFF — Turns off the flashing logo and closes the LXI LAN dialog box.

Query

```
:LXI:IDENtify[:STATe]?
```

The :LXI:IDENtify[:STATe]? query returns whether the oscilloscope is being identified in the graphical user interface LXI LAN dialog box.

Returned Format

```
<setting><NL>
```

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

See Also

- ":SYSTem:DSP" on page 1526

History

New in version 11.30.

33 :MARKer Commands

The commands in the MARKer subsystem specify and query the settings of the time markers (X axis) and current measurement unit markers (volts, amps, and watts for the Y axis). You typically set the Y-axis measurement units using the :CHANnel:UNITs command.

NOTE

Guidelines for Using Queries in Marker Modes

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries. If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected.

:MARKer:CURSor?

Query `:MARKer:CURSor? {DELTa | STARt | STOP}`

The :MARKer:CURSor? query returns the time and current measurement unit values of the specified marker (if markers are in Track Waveforms mode) as an ordered pair of time and measurement unit values.

- If DELTA is specified, the value of delta Y and delta X are returned.
- If START is specified, marker A's x-to-y positions are returned.
- If STOP is specified, marker B's x-to-y positions are returned.

CAUTION

The :MARKer:CURSor? query may change marker mode and results.

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other marker queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries.

If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected. In addition, measurement results may not be what you expected.

Returned Format

```
[:MARKer:CURSor] {DELTa | STARt | STOP}
{<Ax, Ay> | <Bx, By> | <deltaX, deltaY>}<NL>
```

Example This example returns the current position of the X cursor and measurement unit marker 1 to the string variable, strPosition. The program then prints the contents of the variable to the computer's screen.

```
Dim strPosition As String   ' Dimension variable.
myScope.WriteString ":MARKer:CURSor? STARt"
strPosition = myScope.ReadString
Debug.Print strPosition
```

History Legacy command (existed before version 3.10).

:MARKer:DELTa

Command

```
:MARKer:DELTa {{0 | OFF} | {1 | ON}}
```

The :MARKer:DELTa command turns on or off the graphical user interface's **Delta Markers** check box setting to display deltas on the screen (as opposed to the deltas in the results area and remote queries).

Query

```
:MARKer:DELTa?
```

The :MARKer:DELTa? query returns the graphical user interface's **Delta Markers** check box setting (in the Markers dialog box).

Returned Format

```
<setting><NL>

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

See Also

- ":MARKer:XDELta?" on page 852
- ":MARKer:YDELta?" on page 855

History New in version 6.10.

:MARKer:MEASurement:MEASurement

Command

```
:MARKer:MEASurement:MEASurement {MEASurement<N>}[, {{1 | ON}
    | {0 | OFF}}]
```

The :MARKer:MEASurement:MEASurement command adds or deletes measurement tracking markers. This setting is only used when the :MARKer:MODE is set to MEASurement.

When not specifically turning a particular measurement tracking marker on or off (for example, ":MARK:MEAS:MEAS MEAS1"), all markers are deleted, and measurement tracking markers for the specified measurement are added.

When specifically turning a particular measurement tracking marker on or off (for example, ":MARK:MEAS:MEAS MEAS2,ON" or ":MARK:MEAS:MEAS MEAS3,OFF"), only the measurement tracking markers for the specified measurement are affected. No other markers are deleted before turning the specified measurement tracking marker on or off.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

Example This example deletes all markers and adds measurement tracking markers for the fourth measurement.

```
myScope.WriteString ":MARKer:MEASurement:MEASurement MEASurement4"
```

Query

```
:MARKer:MEASurement:MEASurement? [MEASurement<N>]
```

The :MARKer:MEASurement:MEASurement? query returns the status of the measurement tracking marker.

Without any parameters (for example, ":MARK:MEAS:MEAS?"), the query looks at the tracking markers most recently added without using the "ON | OFF" syntax. If those tracking markers are still on, the measurement number is returned (for example, "MEAS1"). If those tracking markers have since been deleted, an empty string is returned.

When using the MEASurement<N> query parameter, a 1 or 0 is returned if measurement tracking markers for the specified measurement are ON or OFF.

NOTE

For :MARKer:MEASurement:MEASurement, command and query syntax using the optional [] square bracket parameters is called new-style syntax. Syntax not using the optional [] square bracket parameters is called old-style syntax. Keysight recommends when using new-style commands to also use new-style queries. Likewise, when using old-style commands, use old-style queries.

Adding and deleting measurement tracking markers from the graphical user interface (GUI) is the same as using new-style command syntax.

Returned Format

```
[:MARKer:MEASurement:MEASurement] {MEAS<N> | {1 | 0}}<NL>
```

Example

This example places the status of the measurement tracking marker in the string variable, strTrackMeasStatus, then prints the contents of the variable to the computer's screen.

```
Dim strTrackMeas As String   ' Dimension variable.
myScope.WriteString ":MARKer:MEASurement:MEASurement?"
strTrackMeasStatus = myScope.ReadString
Debug.Print strTrackMeasStatus
```

See Also

- ":MARKer:MODE" on page 843

History

New in version 3.20.

Version 5.00: Up to 20 measurements are supported.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

Version 11.30: Parameters have been added to support multiple measurement tracking markers.

:MARKer:MODE

Command

```
:MARKer:MODE {OFF | MANual | WAVeform | MEASurement | XONLy | YONLy}
```

The :MARKer:MODE command sets the marker mode:

- OFF — Removes the marker information from the display.
- MANual — Enables manual placement of both X (horizontal) and Y (vertical) markers.
- XONLy — Enables manual placement of X (horizontal) markers.
- YONLy — Enables manual placement of Y (vertical) markers.
- WAVeform — Tracks the current waveform.
- MEASurement — Tracks the most recent measurement.

Example This example sets the marker mode to waveform.

```
myScope.WriteString ":MARKer:MODE WAVeform"
```

Query

```
:MARKer:MODE?
```

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

Returned Format

```
[:MARKer:MODE] {OFF | MAN | WAV | MEAS | XONL | YONL}<NL>
```

Example This example places the current marker mode in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:MODE?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

Version 5.00: The FFTPeak mode was removed.

Version 5.70: Added XONLy and YONLy options for for the "Manual (X only)" and "Manual (Y only)" marker modes.

:MARKer:TSTArt

Command `:MARKer:TSTArt <Ax_position>`

The :MARKer:TSTArt command sets the Ax marker position. The :MARKer:X1Position command described in this chapter also sets the Ax marker position.

NOTE

Use :MARKer:X1Position Instead of :MARKer:TSTArt

The :MARKer:TSTArt command and query perform the same function as the :MARKer:X1Position command and query. The :MARKer:TSTArt command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:X1Position for new programs.

<Ax_position> A real number for the time at the Ax marker, in seconds.

Example This example sets the Ax marker at 90 ns. Notice that this example uses the X1Position command instead of TSTArt.

```
myScope.WriteString ":MARKer:X1Position 90E-9"
```

Query `:MARKer:TSTArt?`

The :MARKer:TSTArt? query returns the time at the Ax marker.

Returned Format `[:MARKer:TSTArt] <Ax_position><NL>`

Example This example places the current setting of the Ax marker in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:X1Position? query instead of the :MARKer:TSTArt? query.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:X1Position?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

NOTE

Do Not Use TST as the Short Form of TSTArt and TSTOp

The short form of the TSTArt command and query does not follow the defined convention for short form commands. Because the short form, TST, is the same for TSTArt and TSTOp, sending TST produces an error. Use TSTA for TSTArt.

History Legacy command (existed before version 3.10).

:MARKer:TSTOp

Command

```
:MARKer:TSTOp <Bx_position>
```

The :MARKer:TSTOp command sets the Bx marker position. The :MARKer:X2Position command described in this chapter also sets the Bx marker position.

NOTE

Use :MARKer:X2Position Instead of :MARKer:TSTOp

The :MARKer:TSTOp command and query perform the same function as the :MARKer:X2Position command and query. The :MARKer:TSTOp command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:X2Position for new programs.

<Bx_position> A real number for the time at the Bx marker, in seconds.

Example This example sets the Bx marker at 190 ns. Notice that this example uses the X2Position command instead of TSTOp.

```
myScope.WriteString ":MARKer:X2Position 190E-9"
```

Query

```
:MARKer:TSTOp?
```

The :MARKer:TSTOp? query returns the time at the Bx marker position.

Returned Format

```
[:MARKer:TSTOp] <Bx_position><NL>
```

Example This example places the current setting of the Bx marker in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:X2Position? query instead of the :MARKer:TSTOp? query.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:X2Position?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

NOTE

Do Not Use TST as the Short Form of TSTArt and TSTOp

The short form of the TSTOp command and query does not follow the defined convention for short form commands. Because the short form, TST, is the same for TSTArt and TSTOp, sending TST produces an error. Use TSTO for TSTOp.

History Legacy command (existed before version 3.10).

:MARKer:VSTArt

Command

```
:MARKer:VSTArt <Ay_position>
```

The :MARKer:VSTArt command sets the Ay marker position and moves the Ay marker to the specified measurement unit value on the specified source. The :MARKer:Y1Position command described in this chapter does also.

NOTE

Use :MARKer:Y1Position Instead of :MARKer:VSTArt

The :MARKer:VSTArt command and query perform the same function as the :MARKer:Y1Position command and query. The :MARKer:VSTArt command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:Y1Position for new programs.

<Ay_position> A real number for the current measurement unit value at Ay (volts, amps, or watts).

Example This example sets Ay to -10 mV. Notice that this example uses the Y1Position command instead of VSTArt.

```
myScope.WriteString ":MARKer:Y1Position -10E-3"
```

Query

```
:MARKer:VSTArt?
```

The :MARKer:VSTArt? query returns the current measurement unit level of Ay.

Returned Format

```
[:MARKer:VSTArt] <Ay_position><NL>
```

History Legacy command (existed before version 3.10).

:MARKer:VSTOp

Command

```
:MARKer:VSTOp <By_position>
```

The :MARKer:VSTOp command sets the By marker position. The :MARKer:Y2Position command described in this chapter also sets the By marker position.

NOTE

Use :MARKer:Y2Position Instead of :MARKer:VSTOp

The :MARKer:VSTOp command and query perform the same function as the :MARKer:Y2Position command and query. The :MARKer:VSTOp command is provided for compatibility with programs written for previous oscilloscopes. You should use :MARKer:Y2Position for new programs.

<By_position> A real number for the time at the By marker, in seconds.

Example This example sets the By marker at 10 mV. Notice that this example uses the Y2Position command instead of VSTOp.

```
myScope.WriteString ":MARKer:Y2Position 10E-3"
```

Query

```
:MARKer:VSTOp?
```

The :MARKer:VSTOp? query returns the time at the By marker position.

Returned Format

```
[:MARKer:VSTOp] <By_position><NL>
```

Example This example places the current setting of the By marker in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen. Notice that this example uses the :MARKer:Y2? query instead of the :MARKer:VSTOp? query.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:Y2Position?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:MARKer:X1Position

Command

```
:MARKer:X1Position <Ax_position>
```

The :MARKer:X1Position command sets the Ax marker position, and moves the Ax marker to the specified time with respect to the trigger time.

<Ax_position> A real number for the time at the Ax marker in seconds.

Example This example sets the Ax marker to 90 ns.

```
myScope.WriteString ":MARKer:X1Position 90E-9"
```

Query

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the time at the Ax marker position.

Returned Format

```
[:MARKer:X1Position] <Ax_position><NL>
```

Example This example returns the current setting of the Ax marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:X1Position?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also :MARKer:TSTArt

History Legacy command (existed before version 3.10).

:MARKer:X2Position

Command `:MARKer:X2Position <Bx_position>`

The :MARKer:X2Position command sets the Bx marker position and moves the Bx marker to the specified time with respect to the trigger time.

<Bx_position> A real number for the time at the Bx marker in seconds.

Example This example sets the Bx marker to 90 ns.

```
myScope.WriteString ":MARKer:X2Position 90E-9"
```

Query `:MARKer:X2Position?`

The :MARKer:X2Position? query returns the time at Bx marker in seconds.

Returned Format `[:MARKer:X2Position] <Bx_position><NL>`

Example This example returns the current position of the Bx marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:X2Position?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MARKer:X1Y1source

Command

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

The :MARKer:X1Y1source command sets the source for the Ax and Ay markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | HISTogram | PNOise | INPut | CORRected | ERRor | LFPR | DIGital<M> | BUS<B>}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example selects channel 1 as the source for markers Ax and Ay.

```
myScope.WriteString ":MARKer:X1Y1source CHANnel1"
```

Query

```
:MARKer:X1Y1source?
```

The :MARKer:X1Y1source? query returns the current source for markers Ax and Ay.

Returned Format

```
[:MARKer:X1Y1source] {CHAN<N> | DIFF<D> | COMM<C> | FUNC<F> | WMEM<R>
   | CLOC | MTR | MSP | EQU<L> | XT<X> | HIST | PNO | INP | CORR
   | ERR | LFPR | DIG<M> | BUS<B>}<NL>
```

Example This example returns the current source selection for the Ax and Ay markers to the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:X1Y1source?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

:MARKer:X2Y2source

Command `:MARKer:X2Y2source <source>`

The :MARKer:X2Y2source command sets the source for the Bx and By markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | HISTogram | PNOise | INPut | CORRected | ERRor | LFPR | DIGital<M> | BUS<B>}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example selects channel 1 as the source for markers Bx and By.

```
myScope.WriteString ":MARKer:X2Y2source CHANnel1"
```

Query `:MARKer:X2Y2source?`

The :MARKer:X2Y2source? query returns the current source for markers Bx and By.

Returned Format

```
[:MARKer:X2Y2source] {CHAN<N> | DIFF<D> | COMM<C> | FUNC<F> | WMEM<R>
   | CLOC | MTR | MSP | EQU<L> | XT<X> | HIST | PNO | INP | CORR
   | ERR | LFPR | DIG<M> | BUS<B>}<NL>
```

Example This example returns the current source selection for the Bx and By markers to the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:X2Y2source?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

:MARKer:XDELta?

Query `:MARKer:XDELta?`

The :MARKer:XDELta? query returns the time difference between Ax and Bx time markers.

Xdelta = time at Bx - time at Ax

Returned Format `[:MARKer:XDELta] <time><NL>`

<time> Time difference between Ax and Bx time markers in seconds.

Example This example returns the current time between the Ax and Bx time markers to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:XDELta?"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

History Legacy command (existed before version 3.10).

:MARKer:Y1Position

Command

```
:MARKer:Y1Position <Ay_position>
```

The :MARKer:Y1Position command sets the Ay marker position on the specified source.

<Ay_position> A real number for the current measurement unit value at Ay (volts, amps, or watts).

Example This example sets the Ay marker to 10 mV.

```
myScope.WriteString ":MARKer:Y1Position 10E-3"
```

Query

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns the current measurement unit level at the Ay marker position.

Returned Format

```
[:MARKer:Y1Position] <Ay_position><NL>
```

Example This example returns the current setting of the Ay marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:Y1Position?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MARKer:Y2Position

Command

```
:MARKer:Y2Position <By_position>
```

The :MARKer:Y2Position command sets the By marker position on the specified source.

<By_position> A real number for the current measurement unit value at By (volts, amps, or watts).

Example This example sets the By marker to-100 mV.

```
myScope.WriteString ":MARKer:Y2Position -100E-3"
```

Query

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns the current measurement unit level at the By marker position.

Returned Format

```
[:MARKer:Y2Position] <By_position><NL>
```

Example This example returns the current setting of the By marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:Y2Position?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MARKer:YDELta?

Query `:MARKer:YDELta?`

The :MARKer:YDELta? query returns the current measurement unit difference between Ay and By.

Ydelta = value at By - value at Ay

Returned Format `[:MARKer:YDELta] <value><NL>`

<value> Measurement unit difference between Ay and By.

Example This example returns the voltage difference between Ay and By to the numeric variable, varVolts, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MARKer:YDELta?"
varVolts = myScope.ReadNumber
Debug.Print FormatNumber(varVolts, 0)
```

History Legacy command (existed before version 3.10).

:MARKer<K>:CMODe

Command

```
:MARKer<K>:CMODe {CUSTom | SOURce}
```

The :MARKer<K>:CMODe command specifies a particular marker's color mode:

- CUSTom — A custom color can be specified for the marker using the :MARKer<K>:COLor command.
- SOURce — The marker is set to the color of the associated channel, waveform memory, or math function source.

<K> An integer, 1-60.

Query

```
:MARKer<K>:CMODe?
```

The :MARKer<K>:CMODe? query returns a particular marker's color mode.

Returned Format

```
<color_mode><NL>
<color_mode> ::= {CUST | SOUR}
```

See Also

- ":MARKer<K>:COLor" on page 857

History New in version 10.25.

:MARKer<K>:COLor

Command `:MARKer<K>:COLor <color>`

The :MARKer<K>:COLor command gives the marker a custom color when the color mode is set to CUSTom (see :MARKer<K>:CMODe).

<K> An integer, 1-60.

<color> A quoted string where the color is specified as "#HHHHHHHH" where Eight-digit hex notation consists of a hash symbol (#), followed by eight characters. The first two represent the alpha channel of the color. The remaining six characters represent the RGB (red, green, blue) value of the color. Additionally, a named color from the following list can be specified (strings are not case-sensitive):

Table 15 Named Marker Colors

AliceBlue (#FFF0F8FF)	DarkSlateGray (#FF2F4F4F)	LightSalmon (#FFFFA07A)	PaleVioletRed (#FFDB7093)
AntiqueWhite (#FFFAEBD7)	DarkTurquoise (#FF00CED1)	LightSeaGreen (#FF20B2AA)	PapayaWhip (#FFFFEFD5)
Aqua (#FF00FFFF)	DarkViolet (#FF9400D3)	LightSkyBlue (#FF87CEFA)	PeachPuff (#FFFFDAB9)
Aquamarine (#FF7FFFD4)	DeepPink (#FFFF1493)	LightSlateGray (#FF778899)	Peru (#FFCD853F)
Azure (#FFF0FFFF)	DeepSkyBlue (#FF00BFFF)	LightSteelBlue (#FFB0C4DE)	Pink (#FFFFC0CB)
Beige (#FFF5F5DC)	DimGray (#FF696969)	LightYellow (#FFFFFFE0)	Plum (#FFDDA0DD)
Bisque (#FFFFE4C4)	DodgerBlue (#FF1E90FF)	Lime (#FF00FF00)	PowderBlue (#FFB0E0E6)
Black (#FF000000)	FireBrick (#FFB22222)	LimeGreen (#FF32CD32)	Purple (#FF800080)
BlanchedAlmond (#FFFFEBCD)	FloralWhite (#FFFFFAF0)	Linen (#FFFAF0E6)	Red (#FFFF0000)
Blue (#FF0000FF)	ForestGreen (#FF228B22)	Magenta (#FFFF00FF)	RosyBrown (#FFBC8F8F)
BlueViolet (#FF8A2BE2)	Fuchsia (#FFFF00FF)	Maroon (#FF800000)	RoyalBlue (#FF4169E1)
Brown (#FFA52A2A)	Gainsboro (#FFDCDCDC)	MediumAquamarine (#FF66CDAA)	SaddleBrown (#FF8B4513)
BurlyWood (#FFDEB887)	GhostWhite (#FFF8F8FF)	MediumBlue (#FF0000CD)	Salmon (#FFFA8072)

Table 15 Named Marker Colors (continued)

	CadetBlue (#FF5F9EA0)		Gold (#FFFFD700)		MediumOrchid (#FFBA55D3)		SandyBrown (#FFF4A460)
	Chartreuse (#FF7FFF00)		Goldenrod (#FFDAA520)		MediumPurple (#FF9370DB)		SeaGreen (#FF2E8B57)
	Chocolate (#FFD2691E)		Gray (#FF808080)		MediumSeaGreen (#FF3CB371)		Seashell (#FFFFF5EE)
	Coral (#FFFF7F50)		Green (#FF00FF00)		MediumSlateBlue (#FF7B68EE)		Sienna (#FFA0522D)
	CornflowerBlue (#FF6495ED)		GreenYellow (#FFADFF2F)		MediumSpringGreen (#FF00FA9A)		Silver (#FFC0C0C0)
	CornSilk (#FFFFF8DC)		Honeydew (#FFF0FFF0)		MediumTurquoise (#FF48D1CC)		SkyBlue (#FF87CEEB)
	Crimson (#FFDC143C)		HotPink (#FFFF69B4)		MediumVioletRed (#FFC71585)		SlateBlue (#FF6A5ACD)
	Cyan (#FF00FFFF)		IndianRed (#FFCD5C5C)		MidnightBlue (#FF191970)		SlateGray (#FF708090)
	DarkBlue (#FF00008B)		Indigo (#FF4B0082)		MintCream (#FFF5FFFA)		Snow (#FFFFFAFA)
	DarkCyan (#FF008B8B)		Ivory (#FFFFFFF0)		MistyRose (#FFFFE4E1)		SpringGreen (#FF00FF7F)
	DarkGoldenrod (#FFB8860B)		Khaki (#FFF0E68C)		Moccasin (#FFFFE4B5)		SteelBlue (#FF4682B4)
	DarkGray (#FFA9A9A9)		Lavender (#FFE6E6FA)		NavajoWhite (#FFFFDEAD)		Tan (#FFD2B48C)
	DarkGreen (#FF006400)		LavenderBlush (#FFFFF0F5)		Navy (#FF000080)		Teal (#FF008080)
	DarkKhaki (#FFBDB76B)		LawnGreen (#FF7CFC00)		OldLace (#FFFDF5E6)		Thistle (#FFD8BFD8)
	DarkMagenta (#FF8B008B)		LemonChiffon (#FFFFFACD)		Olive (#FF808000)		Tomato (#FFFF6347)
	DarkOliveGreen (#FF556B2F)		LightBlue (#FFADD8E6)		OliveDrab (#FF6B8E23)		Turquoise (#FF40E0D0)
	DarkOrange (#FFFF8C00)		LightCoral (#FFF08080)		Orange (#FFFFA500)		Violet (#FFEE82EE)
	DarkOrchid (#FF9932CC)		LightCyan (#FFE0FFFF)		OrangeRed (#FFFF4500)		Wheat (#FFF5DEB3)
	DarkRed (#FF8B0000)		LightGoldenrodYellow (#FFFAFAD2)		Orchid (#FFDA70D6)		White (#FFFFFFFF)

Table 15 Named Marker Colors (continued)

	DarkSalmon (#FFE9967A)		LightGray (#FFD3D3D3)		PaleGoldenrod (#FFEEE8AA)		WhiteSmoke (#FFF5F5F5)
	DarkSeaGreen (#FF8FBC8F)		LightGreen (#FF90EE90)		PaleGreen (#FF98FB98)		Yellow (#FFFFFF00)
	DarkSlateBlue (#FF483D8B)		LightPink (#FFFFB6C1)		PaleTurquoise (#FFAFEEEE)		YellowGreen (#FF9ACD32)

Query `:MARKer<K>:COLor?`

The :MARKer<K>:COLor? query returns the marker custom color.

Returned Format `<color><NL>`

`<color> ::= quoted string`

See Also

- ":MARKer<K>:CMODe" on page 856

History New in version 10.25.

:MARKer<K>:DELTa

Command

```
:MARKer<K>:DELTa MARKer<L>,{{0 | OFF} | {1 | ON}}
```

The :MARKer<K>:DELTa command sets a particular marker's "delta to" relationship with another marker of the same type.

<K>, <L> An integer, 1-60.

Query

```
:MARKer<K>:DELTa? MARKer<L>
```

The :MARKer<K>:DELTa? query returns a particular marker's "delta to" state and delta values if the state is 1 (ON).

Returned Format

```
<marker_delta_results><NL>

<marker_delta_results> ::= <delta-to_state>,<delta_X>,<delta_X_inv>,
   <delta_Y>,<delta_Y_over_delta_X>

<delta-to_state> ::= {0 | 1}

<delta_X> ::= ΔX value in NR3 format

<delta_X_inv> ::= 1/ΔX value in NR3 format

<delta_Y> ::= ΔY value in NR3 format

<delta_Y_over_delta_X> ::= ΔY/ΔX value in NR3 format
```

If the delta measurement does not apply or cannot be made or if the "delta to" relationship is 0 (OFF), the infinity representation value (9.99999E+37) is returned.

See Also

- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:X:POSition" on page 866
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:ENABle

Command

```
:MARKer<K>:ENABle {{0 | OFF} | {1 | ON}}
```

The :MARKer<K>:ENABle command turns a particular marker on or off.

<K> An integer, 1-60.

Query

```
:MARKer<K>:ENABle?
```

The :MARKer<K>:ENABle? query returns whether a particular marker is on or off.

Returned Format

```
[:MARKer<K>:ENABle] <setting><NL>

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

See Also

- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:X:POSition" on page 866
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:NAME

Command

```
:MARKer<K>:NAME <name>
```

The :MARKer<K>:NAME command gives the marker a name.

<K> An integer, 1-60.

<name> A quoted string.

Query

```
:MARKer<K>:NAME?
```

The :MARKer<K>:NAME? query returns the marker name.

Returned Format

```
<name><NL>

<name> ::= quoted string
```

See Also

- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:X:POSition" on page 866
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:SOURce

Command

```
:MARKer<K>:SOURce <source>
```

The :MARKer<K>:SOURce command specifies the waveform source of a particular marker.

This command is similar to :MARKer:X1Y1source or :MARKer:X2Y2source commands for marker 1 and marker 2, respectively.

The waveform you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

<K> An integer, 1-60.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | HISTogram | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR | DIGital<M> | BUS<B>}

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example selects channel 1 as the source for markers 3.

```
myScope.WriteString ":MARKer3:SOURce CHANnel1"
```

Query

```
:MARKer<K>:SOURce?
```

The :MARKer<K>:SOURce? query returns the specified source for a particular marker.

Returned Format

```
[:MARKer<K>:SOURce] <source><NL>

<source> ::= {CHAN<N> | DIFF<D> | COMM<C> | FUNC<F> | WMEM<R> | CLOC
    | MTR | MSP | EQU<L> | HIST | DIG<M> | BUS<B> | XT<X> | PNO | INP
    | CORR | ERR | LFPR}
```

See Also

- ":MARKer:X1Y1source" on page 850
- ":MARKer:X2Y2source" on page 851
- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:X:POSition" on page 866
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:TYPE

Command

```
:MARKer<K>:TYPE {XMANual | YMANual | TRACk | RF}
```

The :MARKer<K>:TYPE command specifies a particular marker's type:

- XMANual — manual X only horizontal marker that can be moved freely.
- YMANual — manual Y only vertical marker that can be moved freely.

  Vertical markers are not allowed if the marker source is a digital input channel.
- TRACk — track waveform marker.

  A track waveform marker is a horizontal marker that can be moved freely. The waveform's vertical value at that horizontal time point is also marked (but cannot be moved).
- RF — track RF marker.

  Track RF markers are allowed only on frequency domain (FFT) waveform sources. Track RF markers show the frequency and vertical value associated with the marker's horizontal position.

There is an additional marker type, MEASurement, that cannot be specified with the :MARKer<K>:TYPE command. Measurement markers are turned on or off using the :MEASure:MARK command.

NOTE You cannot change a marker's type when it is enabled. Use the ":MARKer<K>:ENABle OFF" to disable a marker before changing its type.

<K> An integer, 1-60.

Query

```
:MARKer<K>:TYPE?
```

The :MARKer<K>:TYPE? query returns a particular marker's type.

If the marker was added as a measurement marker (using the :MEASure:MARK command), the query will return MEAS.

Returned Format

```
[:MARKer<K>:TYPE] <marker_type><NL>

<marker_type> ::= {XMAN | YMAN | TRAC | RF | MEAS}
```

See Also

- ":MEASure:MARK" on page 985
- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:X:POSition" on page 866
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:X:POSition

Command

```
:MARKer<K>:X:POSition <X_position>
```

The :MARKer<K>:X:POSition command specifies the horizontal position of a particular marker.

Whether this command is valid depends on the type of marker (see :MARKer<K>:TYPE). For example, you cannot set the X position of a manual Y only vertical marker.

<K> An integer, 1-60.

<X_position> Horizontal position of marker in NR3 format.

The horizontal position units are determined by the marker's source; they are typically seconds or Hertz.

Query

```
:MARKer<K>:X:POSition?
```

The :MARKer<K>:X:POSition? query returns a particular marker's horizontal position.

Returned Format

```
[:MARKer<K>:X:POSition] <X_position><NL>
```

If a horizontal position value is not appropriate for the type of marker (see :MARKer<K>:TYPE), for example querying the X position of a manual Y only vertical marker, the infinity representation value (9.99999E+37) is returned.

See Also

- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:Y:POSition" on page 867

History New in version 10.10.

:MARKer<K>:Y:POSition

Command

```
:MARKer<K>:Y:POSition <Y_position>
```

The :MARKer<K>:Y:POSition command specifies the vertical position of a particular marker.

Whether this command is valid depends on the type of marker (see :MARKer<K>:TYPE). For example, you cannot set the Y position of a manual X only vertical marker.

<K> An integer, 1-60.

<Y_position> The vertical position of marker in NR3 format.

The vertical position units are determined by the marker's source; they are typically Volts or dBm, but other options are available with frequency domain waveforms.

Query

```
:MARKer<K>:Y:POSition?
```

The :MARKer<K>:Y:POSition? query returns a particular marker's vertical position.

Returned Format

```
[:MARKer<K>:Y:POSition] <Y_position><NL>
```

If a vertical position value is not appropriate for the type of marker (see :MARKer<K>:TYPE), for example querying the Y position of a manual X only horizontal marker, the infinity representation value (9.99999E+37) is returned.

See Also

- ":MARKer<K>:DELTa" on page 860
- ":MARKer<K>:ENABle" on page 861
- ":MARKer<K>:NAME" on page 862
- ":MARKer<K>:SOURce" on page 863
- ":MARKer<K>:TYPE" on page 864
- ":MARKer<K>:X:POSition" on page 866

History New in version 10.10.

34 :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Jitter Analysis Software Commands

The following MEASure commands are available when the Jitter Analysis Software license is installed.

- ":MEASure:CTCDutycycle" on page 905
- ":MEASure:CTCJitter" on page 907
- ":MEASure:CTCNwidth" on page 909
- ":MEASure:CTCPwidth" on page 911
- ":MEASure:DATarate" on page 913
- ":MEASure:HOLDtime" on page 965
- ":MEASure:JITTer:HISTogram" on page 967
- ":MEASure:JITTer:MEASurement" on page 968
- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:TRENd" on page 979
- ":MEASure:NCJitter" on page 987
- ":MEASure:NPERiod" on page 1002
- ":MEASure:NUI" on page 1006
- ":MEASure:RJDJ:ALL?" on page 1087
- ":MEASure:RJDJ:APLength?" on page 1091
- ":MEASure:RJDJ:BER" on page 1093
- ":MEASure:RJDJ:CLOCk" on page 1095
- ":MEASure:RJDJ:EDGE" on page 1097
- ":MEASure:RJDJ:INTerpolate" on page 1098
- ":MEASure:RJDJ:PLENgth" on page 1102
- ":MEASure:RJDJ:SOURce" on page 1107
- ":MEASure:RJDJ:STATe" on page 1108
- ":MEASure:RJDJ:TJRJDJ?" on page 1109
- ":MEASure:RJDJ:UNITs" on page 1112
- ":MEASure:SETuptime" on page 1117
- ":MEASure:TIEClock2" on page 1174
- ":MEASure:TIEData2" on page 1176
- ":MEASure:UITouijitter" on page 1189
- ":MEASure:UNITinterval" on page 1194

FFT Commands

The :MEASure:FFT commands control the FFT measurements that are accessible through the Measure subsystem.

Measurement Sources Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The :MEASure:SOURce command lets you specify two sources. Most measurements are only made on a single source. Some measurements, such as the DELTatime measurement, require two sources.

For more information on waveform sources, see Chapter 50, “Waveform Sources,” starting on page 2011.

Measurement Setup To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope.

- For a period or frequency measurement, at least one and a half complete cycles must be displayed.
- For a pulse width measurement, the entire pulse must be displayed.
- For a rise time measurement, the leading (positive-going) edge of the waveform must be displayed.
- For a fall time measurement, the trailing (negative-going) edge of the waveform must be displayed.

In jitter mode with jitter statistics enabled, measurements are made on all data regardless of what is on screen.

User-Defined Thresholds If you choose to set user-defined thresholds, they must be set before actually sending the measurement command or query.

Measurement Error If a measurement cannot be made because of a lack of data, because the source waveform is not displayed, the requested measurement is not possible (for example, a period measurement on an FFT waveform), or for some other reason, the following results are returned:

- 9.99999E+37 is returned as the measurement result.
- If SENDvalid is ON, the error code is also returned as well as the questionable value.

Making Measurements If more than one period, edge, or pulse is displayed, time measurements are made on the first, left-most portion of the displayed waveform.

When any of the defined measurements are requested, the oscilloscope first determines the top (100%) and base (0%) voltages of the waveform. From this information, the oscilloscope determines the other important voltage values (10%, 90%, and 50% voltage values) for making measurements.

The 10% and 90% voltage values are used in the rise time and fall time measurements when standard thresholds are selected. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle with standard thresholds selected.

You can also make measurements using user-defined thresholds instead of the standard thresholds.

When the command form of a measurement is used, the oscilloscope is placed in the continuous measurement mode. The measurement result will be displayed on the front panel. There may be a maximum of 5 measurements running continuously. Use the SCRatch command to turn off the measurements.

When the query form of the measurement is used, the measurement is made one time, and the measurement result is returned.

- If the current acquisition is complete, the current acquisition is measured and the result is returned.
- If the current acquisition is incomplete and the oscilloscope is running, acquisitions will continue to occur until the acquisition is complete. The acquisition will then be measured and the result returned.
- If the current acquisition is incomplete and the oscilloscope is stopped, the measurement result will be 9.99999e+37 and the incomplete result state will be returned if SENDvalid is ON.

All measurements are made using the entire display, except for VAVerage and VRMS which allow measurements on a single cycle. Therefore, if you want to make measurements on a particular cycle, display only that cycle on the screen.

If the waveform is clipped, the measurement result may be questionable. In this case, the value returned is the most accurate value that can be made using the current scaling. You might be able to obtain a more accurate measurement by adjusting the vertical scale to prevent the waveform from being clipped.

Note that you can concatenate measurement queries for much faster throughput. For example:

```
:MEASure:VPP? CHANnel1;:MEASure:FREQuency? CHANnel2
```

When you do this, however, values are returned as a single query result, separated by semicolons.

:MEASure:AREA

Command

```
:MEASure:AREA [CYCLe[,<source>[,<direction>]]]
```

```
:MEASure:AREA [DISPlay[,<source>]]
```

The :MEASure:AREA command turns on the area measurement. The area measurement measures between the waveform, or a selected cycle of the waveform, and the waveform ground.

When measuring Area, it is sometimes useful to use the Subtract Math Operator to remove any dc offset from a waveform you want to measure.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the first CYCLe from the left side of the display grid is measured or the entire DISPlay is measured.

When the "Measure All Edges" mode is ON, all cycles in the acquisition are measured or the entire acquisition is measured.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

When the CYCLe option is used, the <direction> option specifies which edge the cycle begins and ends on. When <direction> is specified, the <source> parameter is required.

Example This example turns on the area measurement which measures between the waveform and ground. Only that portion of the waveform which is in the waveform viewing area is measured.

```
myScope.WriteString ":MEASure:AREA DISPlay"
```

Query

```
:MEASure:AREA? [CYCLe[,<source>[,<direction>]]]
```

```
:MEASure:AREA? [DISPlay[,<source>]]
```

The :MEASure:AREA? query returns the area measurement.

Returned Format

```
[:MEASure:AREA] <value>[,<result_state>]<NL>
```

Example This example places the current selection for the area to be measured in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String
myScope.WriteString ":MEASure:AREA?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

Version 5.70: Added a RISing or FALLing edge parameter when measuring a single cycle of the waveform.

:MEASure:BER

Command

```
:MEASure:BER <source>
```

When a pattern length and pattern can be determined (see the :ANALyze:SIGNal:PATTern:* commands), the :MEASure:BER command installs a cumulative bit error rate (BER) measurement of the specified PAM waveform into the user interface's measurement Results pane.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:BER? <source>
```

The :MEASure:BER? query returns the measured cumulative bit error rate value.

Returned Format

```
[:MEASure:BER] <value><NL>
```

```
<value> ::= the cumulative BER value in NR3 format.
```

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History New in version 5.60.

:MEASure:BERPeracq

Command

```
:MEASure:BERPeracq <source>
```

```
<pattern_length> ::= integer number of symbols.
```

When a pattern length and pattern can be determined (see the :ANALyze:SIGNal:PATTern:* commands), the :MEASure:BERPeracq command installs a bit error rate (BER) per acquisition measurement of the specified PAM waveform into the user interface's measurement Results pane.

<source>

{CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:BERPeracq? <source>
```

The :MEASure:BERPeracq? query returns the measured bit error rate per acquisition value.

Returned Format

```
[:MEASure:BERPeracq] <value><NL>
```

```
<value> ::= the BER per acquisition value in NR3 format.
```

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:SER" on page 1115
- ":MEASure:SERPeracq" on page 1116

History

New in version 5.60.

:MEASure:BINTerval

Command

```
:MEASure:BINTerval <source>, <idle time>
```

The :MEASure:BINTerval command measures the amount of time between the end of a burst and beginning of the next burst. The idle time is the minimum time between bursts.

The Burst Interval measurement builds on top of the Burst Width measurement. Two Burst Widths must be identified in order to measure the interval between bursts. To measure a Burst Width, idles are required before and after the burst. Therefore, to measure a Burst Interval, an idle-burst-idle-burst-idle sequence must be captured by the oscilloscope.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<idle time> Minimum amount of idle time between bursts.

Example This example measures the burst interval between two bursts on channel 4 (and with an idle time of 5 microseconds)

```
myScope.WriteString ":MEASure:BINTerval CHAN4, 5e-6"
```

Query

```
:MEASure:BINTerval? <source>, <idle time>
```

The :MEASure:BINTerval? query returns the burst interval time.

See Also

- ":MEASure:BWIDth" on page 885

History Legacy command (existed before version 3.10).

:MEASure:BPERiod

Command

```
:MEASure:BPERiod <source>, <idle time>
```

The :MEASure:BPERiod command measures the time between the beginning of a burst and the beginning of the next burst. The idle time is the minimum time between bursts.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<idle time> Minimum amount of idle time between bursts.

Example This example measures the burst period between two bursts on channel 4 (and with an idle time of 5 microseconds)

```
myScope.WriteString ":MEASure:BPERiod CHAN4, 5e-6"
```

Query

```
:MEASure:BPERiod? <source>, <idle time>
```

The :MEASure:BPERiod? query returns the burst period time.

History Legacy command (existed before version 3.10).

:MEASure:BWIDth

Command

```
:MEASure:BWIDth <source>,<idle_time>
```

The :MEASure:BWIDth command measures the width of bursts in your waveform. The idle time is the minimum time between bursts.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MSPectrum | MTRend | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<idle_time> Amount of idle time between bursts.

Example This example measures the width of bursts for the waveform on channel one and sets the idle time to 1 microsecond.

```
myScope.WriteString ":MEASure:BWIDth CHANnel1,1E-6"
```

Query

```
:MEASure:BWIDth? <source>,<idle_time>
```

The :MEASure:BWIDth? query returns the width of the burst being measured.

Returned Format

```
[:MEASure:BWIDth ]<burst_width><NL>
```

Example This example returns the width of the burst being measured, in the string variable, strBurstwidth, then prints the contents of the variable to the computer's screen.

```
Dim strBurstwidth As String
myScope.WriteString ":MEASure:BWIDth? CHANnel1,1E-6"
strBurstwidth = myScope.ReadString
Debug.Print strBurstwidth
```

See Also

- ":MEASure:BINTerval" on page 883

History Legacy command (existed before version 3.10).

:MEASure:CDRRate

Command

```
:MEASure:CDRRate <source>
```

The :MEASure:CDRRate command determines the data rate (clock recovery rate) from the clock recovery method being used. It yields one data point per acquisition so trending cannot be performed on this measurement.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MSPectrum | MTRend | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the clock recovery rate of channel 1.

```
myScope.WriteString ":MEASure:CDRRate CHANnel1"
```

Query

```
:MEASure:CDRRate? <source>
```

The :MEASure:CDRRate? query returns the data rate (clock recovery rate) for the source waveform.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:CDRRate] <cdr_rate><NL>
```

Example This example places the current data rate of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CDRRate? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:CGRade:CROSsing

Command

```
:MEASure:CGRade:CROSsing [<source>]
```

The :MEASure:CGRade:CROSsing command enables the crossing level percent measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the crossing level measurement will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the crossing level.

```
myScope.WriteString ":MEASure:CGRade:CROSsing"
```

Query

```
:MEASure:CGRade:CROSsing? [<source>]
```

The :MEASure:CGRade:CROSsing? query returns the crossing level percent measurement of the current eye diagram on the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:CROSsing]<value>[,<result_state>]<NL>
```

<value> The crossing level.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current crossing level in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:CROSsing?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade crossing level percent is measured.

:MEASure:CGRade:DCDistortion

Command

```
:MEASure:CGRade:DCDistortion <format> [,<source>]
```

The :MEASure:CGRade:DCDistortion command enables the duty cycle distortion measurement on the current eye pattern. The parameter specifies the format for reporting the measurement. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<format> {TIME | PERCent}

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the duty cycle distortion measurement will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the duty cycle distortion.

```
myScope.WriteString ":MEASure:CGRade:DCDistortion TIME"
```

Query

```
:MEASure:CGRade:DCDistortion? <format> [,<source>]
```

The :MEASure:CGRade:DCDistortion query returns the duty cycle distortion measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:DCDistortion]<value>[,<result_state>]<NL>
```

<value> The duty cycle distortion.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current duty cycle distortion in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CGRade:DCDistortion? PERCent"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade duty cycle distortion is measured.

:MEASure:CGRade:EHEight

Command

```
:MEASure:CGRade:EHEight <algorithm>[,<source>[,<threshold>]]
```

The :MEASure:CGRade:EHEight command enables the eye height measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<algorithm> {MEASured | EXTRapolated}

- MEASured — depending on the location setting (see :MEASure:CGRade:ELOCation), the eye height will be measured either within a window (see :MEASure:CGRade:EWINdow) or at the location of the recovered clock edge.

  When measured within a window, the smallest eye height within the window is reported.
- EXTRapolated — is optional because it is the default if you do not specify an algorithm. Extrapolated will estimate the eye height based upon the mean and standard deviation of the eye top and base.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L> | XT<X> | INPut | CORRected}

If <source> is omitted, the eye height measurement will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<threshold> When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the <threshold> parameter is an integer that specifies which eye to measure, and the <algorithm> parameter is ignored.

- For PAM-3, the <threshold> may be from 0-1.
- For PAM-4, the <threshold> may be from 0-2.
- For PAM-6, the <threshold> may be from 0-4.
- For PAM-8, the <threshold> may be from 0-6.

Example This example enables the eye height measurement.

```
myScope.WriteString ":MEASure:CGRade:EHEight"
```

Query

```
:MEASure:CGRade:EHEight? <algorithm>[,<source>[,<threshold>]]
```

The :MEASure:CGRade:EHEight? query returns the eye height measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:EHEight] <value>[,<result_state>]<NL>
```

<value> The eye height.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current eye height in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:EHEight?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:CGRade:ELOCation" on page 891
- ":MEASure:CGRade:EWINdow" on page 895
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:PAM:EYE:PROBability" on page 1021

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade eye height is measured.

Version 5.50: When the signal type is PAM-4, an additional <threshold> parameter is used to specify which eye, and the <algorithm> parameter is ignored.

Version 11.20: The <threshold> parameter now supports PAM-6 and PAM-8 eyes.

:MEASure:CGRade:ELOCation

Command

```
:MEASure:CGRade:ELOCation {FIND | CLOCk}[, <source>]
```

The :MEASure:CGRade:ELOCation command specifies the eye height measurement location:

- FIND — measures eye height within a specified window of the eye.

  The window is defined using the :MEASure:CGRade:EWINdow command.
- CLOCk — measure eye height at the location of the recovered clock edge.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L> | XT<X> | INPut | CORRected}

If <source> is omitted, the location setting is applied to the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:CGRade:ELOCation? [<source>]
```

The :MEASure:CGRade:ELOCation? query returns the specified location of the eye height measurement.

Returned Format

```
[:MEASure:CGRade:ELOCation] {FIND | CLOC}<NL>
```

See Also

- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:CGRade:EWINdow" on page 895

History New in version 10.10.

:MEASure:CGRade:EWIDth

Command

```
:MEASure:CGRade:EWIDth <algorithm>[,<source>[,<threshold>[,<units>]]]
```

The :MEASure:CGRade:EWIDth command enables the eye width measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<algorithm> {MEASured | EXTRapolated} EXTRapolated is optional because it is the default if you do not specify an algorithm.

MEASured will measure the eye width measurement within the window (see CGRade:EWINdow) of the current data. The smallest eye width is reported. Extrapolated will estimate the eye width based upon the mean and standard deviation of the crossings.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the eye width will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<threshold> When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the <threshold> parameter is an integer that specifies which eye to measure, and the <algorithm> parameter is ignored.

- For PAM-3, the <threshold> may be from 0-1.
- For PAM-4, the <threshold> may be from 0-2.
- For PAM-6, the <threshold> may be from 0-4.
- For PAM-8, the <threshold> may be from 0-6.

For NRZ (non-return-to-zero) signals, this value should be 0.

<units> {SECond | UNITinterval}

Lets you choose the measurement units. If <units> is omitted, the last specified units are used.

If the eye is not a real-time eye, that is, if the eye is constructed by triggering on a data waveform without using clock recovery, trying to specify UNITinterval units will result in an error because the unknown data rate cannot be converted to UI.

Example This example measures the eye width.

```
myScope.WriteString ":MEASure:CGRade:EWIDth"
```

Query

```
:MEASure:CGRade:EWIDth? <algorithm>[,<source>[,<threshold>[,<units>]]]
```

The :MEASure:CGRade:EWIDth? query returns the eye width measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:EWIDth]<value>[,<result_state>]<NL>`

<value> The eye width.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current eye width in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:EWIDth?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:CGRade:EWIDth:THReshold" on page 894
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:PAM:EYE:PROBability" on page 1021

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade eye width is measured.

Version 5.50: When the signal type is PAM-4, an additional <threshold> parameter is used to specify which eye, and the <algorithm> parameter is ignored.

Version 11.10: Added the ability to select units in UNITinterval or SECond (which was the default before).

Version 11.20: The <threshold> parameter now supports PAM-6 and PAM-8 eyes.

:MEASure:CGRade:EWIDth:THReshold

Command `:MEASure:CGRade:EWIDth:THReshold {AUTomatic | SPECified}[, <source>]`

The :MEASure:CGRade:EWIDth:THReshold command specifies the threshold voltage level used in measuring the eye width:

- AUTomatic — Eye widths are measured at the threshold voltage level of the widest eye opening.
- SPECified — Eye widths are measured at the measurement threshold voltage level.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query `:MEASure:CGRade:EWIDth:THReshold? [<source>]`

The :MEASure:CGRade:EWIDth:THReshold? query returns the eye width threshold voltage option setting.

Returned Format `[:MEASure:CGRade:EWIDth:THReshold] {AUT | SPEC}<NL>`

See Also

- ":MEASure:CGRade:EWIDth" on page 892

History New in version 10.20.

:MEASure:CGRade:EWINdow

Command

```
:MEASure:CGRade:EWINdow <start>,<stop>[,<start_after>][,<source>]
```

The :MEASure:CGRade:EWINdow command is used to change the starting point and the stopping point of the window used to make the eye pattern measurements of eye height, eye crossing %, and eye q-factor. In addition, the number of waveform hits can be set to ensure that enough data has been collected to make accurate measurements.

<start> An integer from 1 to 100 for horizontal starting point. (Default value is 40%.)

<stop> An integer from 1 to 100 for horizontal stopping point. (Default value is 60%.)

<start_after> An integer from 1 to 63,488 for number of hits to acquire before making measurements. (Default value is 1.)

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the eye window will be applied to all sources.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the eye window starting point to 2%, the stopping point to 75% and the start after to 5,000 hits.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:EWINdow 2,75,5000"
```

Query

```
:MEASure:CGRade:EWINdow? [<source>]
```

The :MEASure:CGRade:EWINdow query returns the starting point, the ending point, and the start after setting for the eye pattern measurements.

On the query, the eye window of channel 1 will be returned.

Returned Format

```
[:MEASure:CGRade:EWIDdow] <start>,<stop>,<start_after> <NL>
```

The following example returns the values for the eye window.

Example

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:EWINdow?"
varStart,Stop,Startafter = myScope.ReadNumber
Debug.Print FormatNumber(varStart,Stop,Startafter, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade eye window is applied.

:MEASure:CGRade:JITTer

Command

```
:MEASure:CGRade:JITTer <format>[,<source>[,<units>]]
```

The :MEASure:CGRade:JITTer measures the jitter at the eye diagram crossing point. The parameter specifies the format, peak-to-peak or RMS, of the returned results. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

<format> {PP | RMS}

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the jitter will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<units> {SECond | UNITinterval}

Lets you choose the measurement units. If <units> is omitted, the last specified units are used.

If the eye is not a real-time eye, that is, if the eye is constructed by triggering on a data waveform without using clock recovery, trying to specify UNITinterval units will result in an error because the unknown data rate cannot be converted to UI.

Example This example measures the jitter.

```
myScope.WriteString ":MEASure:CGRade:JITTer RMS"
```

Query

```
:MEASure:CGRade:JITTer? <format>[,<source>[,<units>]]
```

The :MEASure:CGRade:JITTer? query returns the jitter measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:JITTer]<value>[,<result_state>]<NL>
```

<value> The jitter.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current jitter in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CGRade:JITTer? RMS"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade jitter is measured.

Version 11.10: Added the ability to select units in UNITinterval or SECond (which was the default before).

:MEASure:CGRade:OLEVel

Command

```
:MEASure:CGRade:OLEVel [<source>]
```

The :MEASure:CGRade:OLEVel command installs an Eye One Level measurement into the user interface's measurement Results pane. Eye one level is a measure of the mean value of the logical 1 of an eye diagram.

Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected}

If <source> is omitted, the Q-factor will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:CGRade:OLEVel? [<source>]
```

The :MEASure:CGRade:OLEVel? query returns the measured Eye One Level.

Returned Format

```
[:MEASure:CGRade:OLEVel] <value>[,<result_state>]<NL>
```

<value> The measured Eye One Level value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:CGRade:ZLEVel" on page 901
- ":MEASure:ERATio" on page 928
- ":MEASure:OPOWer" on page 1011
- ":MEASure:OMAMplitude" on page 1009

History New in version 5.70.

:MEASure:CGRade:QFACtor

Command

```
:MEASure:CGRade:QFACtor [<source>]
```

The :MEASure:CGRade:QFACtor command measures the Q factor. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected}

If <source> is omitted, the Q-factor will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the Q factor.

```
myScope.WriteString ":MEASure:CGRade:QFACtor"
```

Query

```
:MEASure:CGRade:QFACtor? [<source>]
```

The :MEASure:CGRade:QFACtor? query returns the Q factor measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format

```
[:MEASure:CGRade:QFACtor]<value>[,<result_state>]<NL>
```

<value> The Q factor.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the Q factor in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CGRade:QFACtor"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which the color grade Q factor is measured.

:MEASure:CGRade:ZLEVel

Command

```
:MEASure:CGRade:ZLEVel [<source>]
```

The :MEASure:CGRade:ZLEVel command installs an Eye Zero Level measurement into the user interface's measurement Results pane. Eye zero level is a measure of the mean value of the logical 0 of an eye diagram.

Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected}

If <source> is omitted, the Q-factor will be performed on the first waveform that has color grade enabled.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:CGRade:ZLEVel? [<source>]
```

The :MEASure:CGRade:ZLEVel? query returns the measured Eye Zero Level.

Returned Format

```
[:MEASure:CGRade:ZLEVel] <value>[,<result_state>]<NL>
```

<value> The measured Eye Zero Level value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:CGRade:OLEVel" on page 899
- ":MEASure:ERATio" on page 928
- ":MEASure:OPOWer" on page 1011
- ":MEASure:OMAMplitude" on page 1009

History New in version 5.70.

:MEASure:CHARge

Command

```
:MEASure:CHARge [<primary_channel_source>]
```

When N2820A/N2821A high-sensitivity current probes are connected, the :MEASure:CHARge command adds the Charge measurement to the Measurements tab.

This measurement determines the total current consumption over time with the results listed in ampere-hours (Ah).

When both the primary and secondary cables of a N2820A probe are used, the measurement includes the area under the curve across both Zoomed-In and Zoomed-Out waveforms.

<primary_channel_source> {CHANnel<N>}

<N> An integer, 1 to the number of analog input channels, and should be the primary channel of the N2820A/N2821A probe.

Example This example turns on the Charge measurement and adds it to the Measurements tab.

```
myScope.WriteString ":MEASure:CHARge CHANnel1"
```

Query

```
:MEASure:CHARge?
```

The :MEASure:CHARge? query returns the measured Charge value in Amp-hours.

Returned Format

```
[:MEASure:CHARge]<value>[,<result_state>]<NL>
```

Example This example places the measured Charge value in the string variable, strCharge, then prints the contents of the variable to the computer's screen.

```
Dim strCharge As String
myScope.WriteString ":MEASure:CHARge?"
strCharge = myScope.ReadString
Debug.Print strCharge
```

See Also

- ":MEASure:WINDow" on page 1212
- ":CHANnel<N>:PROBe:PRIMary" on page 497

History New in version 4.20.

:MEASure:CLEar

Command

```
:MEASure:{CLEar | SCRatch}
```

The :MEASure:CLEar command clears the measurement results from the screen and disables all previously enabled measurements.

Example This example clears the current measurement results from the screen.

```
myScope.WriteString ":MEASure:CLEar"
```

History Legacy command (existed before version 3.10).

:MEASure:CROSsing

Command

```
:MEASure:CROSsing <source1>,<source2>
```

The :MEASure:CROSsing command adds the crossing measurement to the screen. The crossing measurement is the voltage where two signals cross (uses edges closest to the center of the screen)

<source1>, <source2> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<hysteresis> a real number

Example This example measures the voltage where channel 1 and 2 cross.

```
myScope.WriteString ":MEASure:CROSsing CHANnel1, CHANnel2"
```

Query

```
:MEASure:CROSsing? [<source1>, <source2>]
```

The :MEASure:CROSsing? query returns the crossing measurement value.

If the <source> parameters are not specified, the two sources specified by the :MEASure:SOURce command are used.

Returned Format

```
[:MEASure:CROSsing] <value><NL>
```

<value> The voltage value where the signals cross.

Example This example places the crossing voltage value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CROSsing? CHANnel1, CHANnel2"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:SOURce" on page 1124

History Legacy command (existed before version 3.10).

:MEASure:CTCDutycycle

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:CTCDutycycle <source>,<direction>
```

The :MEASure:CYCDutycycle command measures the cycle-to-cycle duty cycle jitter (%) of the waveform. Another name for this measurement is "duty cycle - duty cycle".

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example This example measures the cycle-to-cycle duty cycle on the rising edge of channel 1.

```
myScope.WriteString ":MEASure:CTCDutycycle CHANnel1,RISing"
```

Query

```
:MEASure:CTCDutycycle? <source>,<direction>
```

The :MEASure:CTCDutycycle? query returns the cycle-to-cycle duty cycle jitter (%) measurement.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:CTCDutycycle <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle duty cycle jitter (%) of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle duty cycle of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CTCDutycycle CHANnel1,RISing"
```

```
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332
- ":MEASure:TIEClock2" on page 1174
- ":MEASure:CTCJitter" on page 907
- ":MEASure:NCJitter" on page 987
- ":MEASure:CTCPwidth" on page 911
- ":MEASure:CTCNwidth" on page 909

History Legacy command (existed before version 3.10).

:MEASure:CTCJitter

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:CTCJitter <source>,<direction>
```

The :MEASure:CYCJitter command measures the cycle-to-cycle jitter of the waveform. Another name for this measurement is "period-period", where the number of cycles is one.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example This example measures the cycle-to-cycle jitter on the rising edge of channel 1.

```
myScope.WriteString ":MEASure:CTCJitter CHANnel1,RISing"
```

Query

```
:MEASure:CTCJitter? <source>,<direction>
```

The :MEASure:CTCJitter? query returns the cycle-to-cycle jitter time measurement.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:CTCJitter <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle jitter time of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle jitter of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CTCJitter CHANnel1,RISing"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332
- ":MEASure:TIEClock2" on page 1174
- ":MEASure:NCJitter" on page 987
- ":MEASure:CTCPwidth" on page 911
- ":MEASure:CTCNwidth" on page 909
- ":MEASure:CTCDutycycle" on page 905

History Legacy command (existed before version 3.10).

:MEASure:CTCNwidth

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:CTCNwidth [<source>]
```

The :MEASure:CTCNwidth command measures the cycle-to-cycle -width jitter of the waveform. Another name for this measurement is "-width - -width".

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the cycle-to-cycle -width of channel 1.

```
myScope.WriteString ":MEASure:CTCNwidth CHANnel1"
```

Query

```
:MEASure:CTCNwidth? [<source>]
```

The :MEASure:CTCNwidth? query returns the cycle-to-cycle -width jitter measurement.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:CTCNwidth <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle - width jitter of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle - width of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CTCNwidth CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332
- ":MEASure:TIEClock2" on page 1174

- ":MEASure:CTCJitter" on page 907
- ":MEASure:NCJitter" on page 987
- ":MEASure:CTCPwidth" on page 911
- ":MEASure:CTCDutycycle" on page 905

History Legacy command (existed before version 3.10).

:MEASure:CTCPwidth

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:CTCPwidth [<source>]
```

The :MEASure:CYCPwidth command measures the cycle-to-cycle +width jitter of the waveform. Another name for this measurement is "+width - +width".

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the cycle-to-cycle +width of channel 1.

```
myScope.WriteString ":MEASure:CTCPwidth CHANnel1"
```

Query

```
:MEASure:CTCPwidth? [<source>]
```

The :MEASure:CTCPwidth? query returns the cycle-to-cycle +width jitter measurement.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:CTCPwidth <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle +width jitter of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle + width of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CTCPwidth CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332
- ":MEASure:TIEClock2" on page 1174

- ":MEASure:CTCJitter" on page 907
- ":MEASure:NCJitter" on page 987
- ":MEASure:CTCNwidth" on page 909
- ":MEASure:CTCDutycycle" on page 905

History Legacy command (existed before version 3.10).

:MEASure:DATarate

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:DATarate <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:DATarate command measures the data rate in bits per second for the selected source. Use the :MEASure:UNITinterval command/query to measure the unit interval of the source

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<data_rate> A real number specifying the data rate.

Example This example measures the data rate of channel 1.

```
myScope.WriteString ":MEASure:DATarate CHANnel1"
```

Query

```
:MEASure:DATarate? <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:DATarate? query returns the measured data rate.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:DATarate] <value>[,<result_state>]<NL>
```

<value> Data rate frequency in bits per second for the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current data rate of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:DATarate? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:DCAI

Command

```
:MEASure:DCAI [<source>]
```

The :MEASure:DCAI command installs a duty cycle adjustment measurement for the I phase of a DDR5 DQS (strobe) signal. The I phase is the first rising edge to first falling edge.

Duty cycle adjustment measurements are edge-edge measurements that have been modified so that each measurement instance runs on the appropriate UI only.

The duty cycle adjustment measurements are designed to be run on a single acquisition in Measure All Edges mode so that statistics are accumulated across all on-screen UIs.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:DCAI? [<source>]
```

The :MEASure:DCAI? query returns the measured duty cycle adjustment value for the I phase of a DDR5 DQS (strobe) signal.

Returned Format

```
[:MEASure:DCAI] <value>[,<result_state>]<NL>
```

<value> The duty cycle adjustment for I value in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:DCAQ" on page 916
- ":MEASure:DCIPrime" on page 917
- ":MEASure:DCQPrime" on page 918

History New in version 11.25.

:MEASure:DCAQ

Command

```
:MEASure:DCAQ [<source>]
```

The :MEASure:DCAQ command installs a duty cycle adjustment measurement for the Q phase of a DDR5 DQS (strobe) signal. The Q phase is the first falling edge to second rising edge.

Duty cycle adjustment measurements are edge-edge measurements that have been modified so that each measurement instance runs on the appropriate UI only.

The duty cycle adjustment measurements are designed to be run on a single acquisition in Measure All Edges mode so that statistics are accumulated across all on-screen UIs.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:DCAQ? [<source>]
```

The :MEASure:DCAQ? query returns the measured duty cycle adjustment value for the Q phase of a DDR5 DQS (strobe) signal.

Returned Format

```
[:MEASure:DCAQ] <value>[,<result_state>]<NL>
```

<value> The duty cycle adjustment for Q value in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:DCAI" on page 915
- ":MEASure:DCIPrime" on page 917
- ":MEASure:DCQPrime" on page 918

History New in version 11.25.

:MEASure:DCIPrime

Command

```
:MEASure:DCIPrime [<source>]
```

The :MEASure:DCIPrime command installs a duty cycle adjustment measurement for the I' (I prime) phase of a DDR5 DQS (strobe) signal. The I' phase is the second rising edge to second falling edge.

Duty cycle adjustment measurements are edge-edge measurements that have been modified so that each measurement instance runs on the appropriate UI only.

The duty cycle adjustment measurements are designed to be run on a single acquisition in Measure All Edges mode so that statistics are accumulated across all on-screen UIs.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:DCIPrime? [<source>]
```

The :MEASure:DCIPrime? query returns the measured duty cycle adjustment value for the I' (I prime) phase of a DDR5 DQS (strobe) signal.

Returned Format

```
[:MEASure:DCIPrime] <value>[,<result_state>]<NL>
```

<value> The duty cycle adjustment for I' value in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:DCAI" on page 915
- ":MEASure:DCAQ" on page 916
- ":MEASure:DCQPrime" on page 918

History New in version 11.25.

:MEASure:DCQPrime

Command `:MEASure:DCQPrime [<source>]`

The :MEASure:DCQPrime command installs a duty cycle adjustment measurement for the Q' (Q prime) phase of a DDR5 DQS (strobe) signal. The Q' phase is the second falling edge to the next rising edge. After the Q' phase, the next set of I, Q, I', and Q' phases begin.

Duty cycle adjustment measurements are edge-edge measurements that have been modified so that each measurement instance runs on the appropriate UI only.

The duty cycle adjustment measurements are designed to be run on a single acquisition in Measure All Edges mode so that statistics are accumulated across all on-screen UIs.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query `:MEASure:DCQPrime? [<source>]`

The :MEASure:DCQPrime? query returns the measured duty cycle adjustment value for the Q' (Q prime) phase of a DDR5 DQS (strobe) signal.

Returned Format `[:MEASure:DCQPrime] <value>[,<result_state>]<NL>`

<value> The duty cycle adjustment for Q' value in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:DCAI" on page 915
- ":MEASure:DCAQ" on page 916
- ":MEASure:DCIPrime" on page 917

History New in version 11.25.

:MEASure:DEEMphasis

Command `:MEASure:DEEMphasis [<source>]`

When the Jitter and Vertical Noise Analysis Software is licensed, the Deemphasis serial data measurement becomes available.

The :MEASure:DEEMphasis command adds the deemphasis measurement.

The de-emphasis measurement relies on the clock recovery to recover a clock for each bit in the data waveform. You need to configure clock recovery appropriately for your signal.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DEEMphasis command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example adds the deemphasis measurement on the channel 1 source.

```
myScope.WriteString ":MEASure:DEEMphasis CHANnel1"
```

Query `:MEASure:DEEMphasis? [<source>]`

The :MEASure:DEEMphasis? query returns the measured deemphasis value of the specified source.

Due to random noise, many bits need to be averaged together to average out the noise. Therefore, the current value has little importance and the mean should be used. See ":MEASure:STATistics" on page 1125.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format `[:MEASure:DEEMphasis] <value>[,<result_state>]<NL>`

<value> For every de-emphasis bit in the waveform, a value is computed using:

$$20 * \log_{10}(\text{de-emphasis voltage / transition voltage})$$

Where:

- Transition voltage is the voltage at the clock location of the preceding transition bit.

- De-emphasis voltage is the voltage at the clock location of de-emphasis bits following a transition bit.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value for deemphasis in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:DEEMphasis? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also
- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:DELTatime

Command

```
:MEASure:DELTatime [<source>[,<source>]]
```

The :MEASure:DELTatime command measures the delta time between two edges. If one source is specified, the delta time from the leading edge of the specified source to the trailing edge of the specified source is measured. If two sources are specified, the delta time from the leading edge on the first source to the trailing edge on the second source is measured.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DELTatime command. The rest of the parameters for this command are specified with the :MEASure:DEFine command.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the delta time between channel 1 and channel 2.

```
myScope.WriteString ":MEASure:DELTatime CHANnel1,CHANnel2"
```

Query

```
:MEASure:DELTatime? [<source>[,<source>]]
```

The :MEASure:DELTatime? query returns the measured delta time value.

Returned Format

```
[:MEASure:DELTatime] <value>[,<result_state>]<NL>
```

<value> Delta time from the first specified edge on one source to the next specified edge on another source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of delta time in the numeric variable, varValue, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:DELTatime?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:DELTatime:DEFine

Command

```
:MEASure:DELTatime:DEFine <start_edge_direction>,<start_edge_number>,
    <start_edge_position>,<stop_edge_direction>,<stop_edge_number>,
    <stop_edge_position>
```

The :MEASure:DELTatime:DEFine command sets the type of direction, the number of the edge, and the edge position for the delta time measurement.

<start_edge_direction> {RISing | FALLing | EITHer} for start directions.

<start_edge_number> An integer from 1 to 65534 for start edge numbers.

<start_edge_position> {UPPer | MIDDle | LOWer} for start edge positions.

<stop_edge_direction> {RISing | FALLing | EITHer} for stop directions.

<stop_edge_number> An integer from 1 to 65534 for stop edge numbers.

<stop_edge_position> {UPPer | MIDDle | LOWer} for stop edge positions.

Example This example sets the delta time starting edge to a rising edge on the 5th edge at the middle position and the stopping edge to falling on the 50th edge at the lower position.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString _
    ":MEASure:DELTatime:DEFine RISing,5,MIDDle,FALLing,50,LOWer"
```

Query

```
:MEASure:DELTatime:DEFine?
```

The :MEASure:DELTatime:DEFine? query returns the measured delta time value.

Returned Format

```
[:MEASure:DELTatime:DEFine] <start_edge_direction>,<start_edge_number>,
    <start_edge_position>,<stop_edge_direction>,<stop_edge_number>,
    <stop_edge_position><NL>
```

Example This example places the current value of delta time definition in the string variable, strValue, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
Dim strValue As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:DELTatime:DEFine?"
strValue = myScope.ReadString
Debug.Print strValue
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:DUTYcycle

Command

```
:MEASure:DUTYcycle [<source>[,<direction>]]
```

The :MEASure:DUTYcycle command measures the ratio (%) of the positive pulse width to the period.

Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter following the :MEASure:DUTYcycle command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether the duty cycle is measured from rising edge to rising edge or from falling edge to falling edge. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether whether the duty cycle is measured from rising edge to rising edge or from falling edge to falling edge throughout the acquisition.

Example This example measures the duty cycle of the channel 1 waveform.

```
myScope.WriteString ":MEASure:DUTYcycle CHANnel1"
```

Query

```
:MEASure:DUTYcycle? [<source>[,<direction>]]
```

The :MEASure:DUTYcycle? query returns the measured duty cycle (%) of the specified source.

Returned Format

```
[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>
```

<value> The ratio (%) of the positive pulse width to the period.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current duty cycle of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:DUTYcycle? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:EDGE

Command

```
:MEASure:EDGE [<source>[,<direction>]]
```

The :MEASure:EDGE command measures the time of edges, relative to the timebase reference location.

Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing | BOTH}

Specifies the edge whose time is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), BOTH means whichever edge is nearest to the timebase reference location is used.

When the "Measure All Edges" mode is ON, BOTH specifies that both rising and falling edge times are measured throughout the acquisition.

Example This example measures the edge times of the channel 1 waveform.

```
myScope.WriteString ":MEASure:EDGE CHANnel1"
```

Query

```
:MEASure:EDGE? [<source>[,<direction>]]
```

The :MEASure:EDGE? query returns the measured edge time of the specified source.

Returned Format

```
[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>
```

<value> The measured edge time.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current duty cycle of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:EDGE? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History New in version 3.10.

:MEASure:ERATio

Command

```
:MEASure:ERATio [<source>[,{RATio | DB | PERCent}]]
```

The :MEASure:ERATio command measures the ratio of the one level and the zero level of an eye diagram of an optical signal.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected}

For more information on <source> parameters, see "Measurement Sources" on page 877.

{RATio | DB | PERCent} Specifies the extinction-ratio correction factor units in Ratio, Decibel, or percentage.

Query

```
:MEASure:ERATio? [<source>[,{RATio | DB | PERCent}]]
```

The :MEASure:ERATio? query returns the measured Extinction Ratio.

Returned Format

```
[:MEASure:ERATio] <value>[,<result_state>]<NL>
```

<value> The measured Extinction Ratio value.

NOTE The Extinction Ratio measurement will give a question mark ("?") result if:

- The dark calibration has not been performed at all.
- The vertical sensitivity, offset, or sample rate has changed since the dark calibration was run.
- The probe temperature has changed by > 2 degrees C.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:OMAMplitude" on page 1009
- ":MEASure:OPOWer" on page 1011
- ":MEASure:CGRade:OLEVel" on page 899
- ":MEASure:CGRade:ZLEVel" on page 901

History New in version 5.70.

:MEASure:ETAEdges

Command

```
:MEASure:ETAEdges <source>[, <n-pulses>[, <direction>]]
```

The :MEASure:ETAEdges command measures the time between edges (RISing, FALLing, or BOTH) within a certain number of pulses (N) across all groups of N pulses in the acquired waveform. At the end of the waveform, the time between edges in the smaller-than-N remaining pulse groups are also measured.

Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<n-pulses> An integer from 1 to 1000 in NR1 format.

<direction> {RISing | FALLing | BOTH}

Query

```
:MEASure:ETAEdges? <source>[, <n-pulses>[, <direction>]]
```

The :MEASure:ETAEdges? query returns a value of 0.0 seconds.

NOTE **The Edge to All Edges measurement is useful when a histogram is applied.**

Returned Format

```
[:MEASure:ETAEdges] <value>[,<result_state>]<NL>
```

<value> A value of 0.0 seconds is returned.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":HISTogram:MODE" on page 698

History New in version 10.20.

:MEASure:ETOedge

Command

```
:MEASure:ETOedge <source_1>,<direction_1>,<position_1>,<next_prev>,
    <relative_edge_number>,<source_2>,<direction_2>,<position_2>
```

The :MEASure:ETOedge command measures the delta time between two edges. It is similar to the delta time measurement, but can be applied to the measurement trend. It also enables you to set whether the measurement is between an edge before or after a specific edge and the number of edges to move forward or backwards.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

The Edge-Edge measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

When this measurement is tracked with markers, markers are displayed at the measurement nearest to the timebase reference location.

<source_1>, <source_2> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction_1>, <direction_2> May be RISing, FALLing, or BOTH

<position_1>, <position_2> May be UPPer, MIDDle, or LOWer

<next_prev> May be NEXT or PREVious

<relative_edge_number> An integer that is the relative number of the second edge.

Query

```
:MEASure:ETOedge? <source_1>,<direction_1>,<position_1>,<next_prev>,
    <relative_edge_number>,<source_2>,<direction_2>,<position_2>
```

The :MEASure:ETOedge? query returns the delta time between the two specified edges.

Returned Format

```
[:MEASure:ETOedge] <value>[,<result_state>]<NL>
```

<value> The measured delta time between two edges value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value for delta time in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:ETOedge? CHAN1,RIS,UPP,NEXT,2,CHAN2,RIS,UP
P"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:FALLtime

Command `:MEASure:FALLtime [<source>[,<start_level>,<stop_level>]]`

The :MEASure:FALLtime command measures the time at the upper threshold of the falling edge, measures the time at the lower threshold of the falling edge, then calculates the fall time.

Fall time = time at lower threshold point - time at upper threshold point.

The first displayed falling edge is used for the fall-time measurement.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FALLtime command.

Four or more sample points on the falling edge of the waveform are required to make this measurement (one above the upper threshold, one below the lower threshold, and two between the thresholds).

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<start_level>, <stop_level> When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the <start_level> and <stop_level> parameters are integers that identify the edge to measure. For PAM-4, the levels may be from 0-3.

For PAM fall time measurements, "Measure All Edges" must be turned on (see :ANALyze:AEDGes).

Example This example measures the fall time of the channel 1 waveform.

```
myScope.WriteString ":MEASure:FALLtime CHANnel1"
```

Query `:MEASure:FALLtime? [<source>[,<start_level>,<stop_level>]]`

The :MEASure:FALLtime? query returns the fall time of the specified source.

Returned Format `[:MEASure:FALLtime] <value>[,<result_state>]<NL>`

<value> Time at lower threshold - time at upper threshold.

A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement. Check the <result_state>. If the required four or more sample points are not present on the falling edge of the waveform, you can use a faster sampling rate or more interpolation points.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See Table 16 for a list of values and descriptions of the result state value.

Example This example places the current value for fall time in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:FALLtime? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:AEDGes" on page 332

History

Legacy command (existed before version 3.10).

Version 5.50: With PAM signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.

Version 11.20: With PAM6 and PAM8 signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.

:MEASure:FFT:CPOWer

Command `:MEASure:FFT:CPOWer <source>,<center_freq>,<meas_bw>`

The :MEASure:FFT:CPOWer command installs a channel power measurement into the user interface's measurement Results pane.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<R> | MSPectrum}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<center_freq> The center frequency used in the measurement in NR3 format.

<meas_bw> The width of FFT band in NR3 format.

Query `:MEASure:FFT:CPOWer? <source>,<center_freq>,<meas_bw>`

The :MEASure:FFT:CPOWer? query returns the measured channel power.

Returned Format `[:MEASure:FFT:CPOWer] <channel_power_value><NL>`

`<channel_power_value> ::= in dBm in NR3 format.`

See Also

- ":MEASure:FFT:OBW" on page 943
- ":MEASure:FFT:PSD" on page 944

History New in version 5.70.

:MEASure:FFT:DFRequency

Command `:MEASure:FFT:DFRequency <source>,<peak1_number>,<peak2_number>,<level>`

The :MEASure:FFT:DFRequency command installs a measurement of the frequency difference between two FFT peaks. Peaks are numbered from low-to-high frequency. Only peaks above the specified threshold level are numbered.

For this command/query to work, the source must be a function that is set to FFT or a waveform memory that contains an FFT.

<source> {FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<peak1_number>, <peak2_number> An integer (in NR1 format) that specifies the FFT peak number.

<level> A decimal number (in NR3 format) that specifies the peak threshold level in dBm.

Query `:MEASure:FFT:DFRequency? <source>,<peak1_number>,<peak2_number>,<level>`

The :MEASure:FFT:DFRequency? query returns the delta frequency value between two FFT peaks.

Returned Format `[:MEASure:FFT:DFRequency] <delta_frequency>[,<result_state>]<NL>`

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example sets up an FFT Magnitude math function on channel 4 and measures the delta frequency between FFT peaks 2 and 4 using a threshold level of -47.0 dBm.

```
' Response headers off:
myScope.WriteString ":SYSTem:HEADer OFF"

' Set up FFT Magnitude math function on channel 4:
myScope.WriteString ":FUNCtion4:FFTMagnitude CHANnel4"

' Display the FFT:
myScope.WriteString ":FUNCtion4:DISPlay ON"

' Install the FFT delta frequency measurement between peaks 2 and 4
' using a peak threshold level of -47 dBm:
myScope.WriteString ":MEASure:FFT:DFRequency FUNCtion4,2,4,-47.0"

' Get the FFT delta frequency measurement result value:
myScope.WriteString ":MEASure:FFT:DFRequency? FUNCtion4,2,4,-47.0"
varDeltaFreq = myScope.ReadNumber
Debug.Print FormatNumber(varDeltaFreq, "Scientific")
```

See Also

- ":MEASure:FFT:DMAGnitude" on page 937

- ":MEASure:FFT:FREQuency" on page 939
- ":MEASure:FFT:MAGNitude" on page 941

History Legacy command (existed before version 3.10).

Version 6.20: The command and query now include peak number and level parameters.

:MEASure:FFT:DMAGnitude

Command

```
:MEASure:FFT:DMAGnitude <source>,<peak1_number>,<peak2_number>,<level>
```

The :MEASure:FFT:DMAGnitude command installs a measurement of the magnitude difference between two FFT peaks. Peaks are numbered from low-to-high frequency. Only peaks above the specified threshold level are numbered.

For this command/query to work, the source must be a function that is set to FFT or a waveform memory that contains an FFT.

<source> {FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<peak1_number>, <peak2_number> An integer (in NR1 format) that specifies the FFT peak number.

<level> A decimal number (in NR3 format) that specifies the peak threshold level in dBm.

Query

```
:MEASure:FFT:DMAGnitude? <source>,<peak1_number>,<peak2_number>,<level>
```

The :MEASure:FFT:DMAGnitude? query returns the delta magnitude value between two FFT peaks.

Returned Format

```
[:MEASure:FFT:DMAGnitude] <delta_magnitude>[,<result_state>]<NL>
```

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example sets up an FFT Magnitude math function on channel 4 and measures the delta magnitude between FFT peaks 2 and 4 using a threshold level of -47.0 dBm.

```
' Response headers off:
myScope.WriteString ":SYSTem:HEADer OFF"

' Set up FFT Magnitude math function on channel 4:
myScope.WriteString ":FUNCtion4:FFTMagnitude CHANnel4"

' Display the FFT:
myScope.WriteString ":FUNCtion4:DISPlay ON"

' Install the FFT delta magnitude measurement between peaks 2 and 4
' using a peak threshold level of -47 dBm:
myScope.WriteString ":MEASure:FFT:DMAGnitude FUNCtion4,2,4,-47.0"

' Get the FFT delta magnitude measurement result value:
myScope.WriteString ":MEASure:FFT:DMAGnitude? FUNCtion4,2,4,-47.0"
varDeltaMag = myScope.ReadNumber
Debug.Print FormatNumber(varDeltaMag, "Scientific")
```

See Also

- ":MEASure:FFT:DFRequency" on page 935

- ":MEASure:FFT:FREQuency" on page 939
- ":MEASure:FFT:MAGNitude" on page 941

History Legacy command (existed before version 3.10).

Version 6.20: The command and query now include peak number and level parameters.

:MEASure:FFT:FREQuency

Command

```
:MEASure:FFT:FREQuency <source>,<peak_number>,<level>
```

The :MEASure:FFT:FREQuency command installs a measurement of the frequency of an FFT peak. Peaks are numbered from low-to-high frequency. Only peaks above the specified threshold level are numbered.

For this command/query to work, the source must be a function that is set to FFT or a waveform memory that contains an FFT.

<source> {FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<peak_number> An integer (in NR1 format) that specifies the FFT peak number.

<level> A decimal number (in NR3 format) that specifies the peak threshold level in dBm.

Query

```
:MEASure:FFT:FREQuency? <source>,<peak_number>,<level>
```

The :MEASure:FFT:FREQuency? query returns the frequency value of the FFT peak.

Returned Format

```
[:MEASure:FFT:FREQuency] <frequency>[,<result_state>]<NL>
```

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example sets up an FFT Magnitude math function on channel 4 and measures the frequency of FFT peak 2 using a threshold level of -47.0 dBm.

```
' Response headers off:
myScope.WriteString ":SYSTem:HEADer OFF"

' Set up FFT Magnitude math function on channel 4:
myScope.WriteString ":FUNCtion4:FFTMagnitude CHANnel4"

' Display the FFT:
myScope.WriteString ":FUNCtion4:DISPlay ON"

' Install the FFT frequency measurement on peak number 2 using a peak
' threshold level of -47 dBm:
myScope.WriteString ":MEASure:FFT:FREQuency FUNCtion4,2,-47.0"

' Get the FFT frequency measurement result value:
myScope.WriteString ":MEASure:FFT:FREQuency? FUNCtion4,2,-47.0"
varFrequency = myScope.ReadNumber
Debug.Print FormatNumber(varFrequency, "Scientific")
```

See Also

- ":MEASure:FFT:DFRequency" on page 935
- ":MEASure:FFT:DMAGnitude" on page 937
- ":MEASure:FFT:MAGNitude" on page 941

History Legacy command (existed before version 3.10).

Version 6.20: The command and query now include peak number and level parameters.

:MEASure:FFT:MAGNitude

Command

```
:MEASure:FFT:MAGNitude <source>,<peak_number>,<level>
```

The :MEASure:FFT:MAGNitude command installs a measurement of the magnitude of an FFT peak. Peaks are numbered from low-to-high frequency. Only peaks above the specified threshold level are numbered.

For this command/query to work, the source must be a function that is set to FFT or a waveform memory that contains an FFT.

<source> {FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<peak_number> An integer (in NR1 format) that specifies the FFT peak number.

<level> A decimal number (in NR3 format) that specifies the peak threshold level in dBm.

Query

```
:MEASure:FFT:MAGNitude? <source>,<peak_number>,<level>
```

The :MEASure:FFT:MAGNitude? query returns the magnitude value of the FFT peak.

Returned Format

```
[:MEASure:FFT:FMAGNitude] <magnitude>[,<result_state>]<NL>
```

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example sets up an FFT Magnitude math function on channel 4 and measures the magnitude of FFT peak 2 using a threshold level of -47.0 dBm.

```
' Response headers off:
myScope.WriteString ":SYSTem:HEADer OFF"

' Set up FFT Magnitude math function on channel 4:
myScope.WriteString ":FUNCtion4:FFTMagnitude CHANnel4"

' Display the FFT:
myScope.WriteString ":FUNCtion4:DISPlay ON"

' Install the FFT magnitude measurement on peak number 2 using a peak
' threshold level of -47 dBm:
myScope.WriteString ":MEASure:FFT:MAGNitude FUNCtion4,2,-47.0"

' Get the FFT magnitude measurement result value:
myScope.WriteString ":MEASure:FFT:MAGNitude? FUNCtion4,2,-47.0"
varMagnitude = myScope.ReadNumber
Debug.Print FormatNumber(varMagnitude, "Scientific")
```

See Also

- ":MEASure:FFT:DFRequency" on page 935
- ":MEASure:FFT:DMAGnitude" on page 937
- ":MEASure:FFT:FREQuency" on page 939

History Legacy command (existed before version 3.10).

Version 6.20: The command and query now include peak number and level parameters.

:MEASure:FFT:OBW

Command

```
:MEASure:FFT:OBW <source>,<occupied_bw_pct>
```

The :MEASure:FFT:OBW command installs an occupied bandwidth measurement into the user interface's measurement Results pane.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<R> | MSPectrum}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<occupied_bw_pct> The percentage of the power of the FFT on screen.

Query

```
:MEASure:FFT:OBW? <source>,<occupied_bw_pct>
```

The :MEASure:FFT:OBW? query returns the measured occupied bandwidth.

Returned Format

```
[:MEASure:FFT:OBW] <bandwidth><NL>

<bandwidth> ::= in Hz in NR3 format.
```

See Also

- ":MEASure:FFT:CPOWer" on page 934
- ":MEASure:FFT:PSD" on page 944

History New in version 5.70.

:MEASure:FFT:PSD

Command

```
:MEASure:FFT:PSD <source>,<center_freq>,<meas_bw>
```

The :MEASure:FFT:PSD command installs a power spectral density measurement into the user interface's measurement Results pane.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<R> | MSPectrum}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<center_freq> The center frequency used in the measurement in NR3 format.

<meas_bw> The width of FFT band in NR3 format.

Query

```
:MEASure:FFT:PSD? <source>,<center_freq>,<meas_bw>
```

The :MEASure:FFT:PSD? query returns the measured power spectral density.

Returned Format

```
[:MEASure:FFT:PSD] <psd_value><NL>

<psd_value> ::= in dBm/Hz in NR3 format.
```

See Also

- ":MEASure:FFT:CPOWer" on page 934
- ":MEASure:FFT:OBW" on page 943

History New in version 5.70.

:MEASure:FFT:THDistortion

Command

```
:MEASure:FFT:THDistortion <source>,{MANual,<fund_freq>
    | PEAK,<level>[,<peak_number>]}
```

The :MEASure:FFT:THDistortion command installs a Total Harmonic Distortion (THD) measurement into the user interface's measurement Results pane.

THD measures the magnitude of the fundamental frequency and as many harmonics as are on the screen.

THD is calculated using the following formula:

$$\mathrm{THD} = 100 \times \frac{\sqrt{X_2{}^2 + X_3{}^2 + X_n{}^2 + \ldots}}{X_1}$$

Where:

- X_n = voltage or current of each harmonic
- X_1 = fundamental frequency voltage or current value

The fundamental frequency of the input signal can be specified manually (using `MANual,<fund_freq>`), or you can let the oscilloscope determine the fundamental frequency automatically based on the first peak above a certain magnitude threshold (using `PEAK,<level>[,<peak_number>]`).

<source> {FUNCtion<F> | WMEMory<R> | MSPectrum}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<fund_freq> The fundamental frequency used in the measurement in NR3 format.

<level> A decimal number (in NR3 format) that specifies the peak threshold level in dBm.

<peak_number> An integer (in NR1 format) that is the index of the peak corresponding to the fundamental frequency (typically 1).

Query

```
:MEASure:FFT:THDistortion? <source>,{MANual,<fund_freq>
    | PEAK,<level>[,<peak_number>]}
```

The :MEASure:FFT:THDistortion? query returns the measured Total Harmonic Distortion (THD).

Returned Format

```
[:MEASure:FFT:THDistortion] <THD_value><NL>

<THD_value> ::= percent in NR3 format.
```

See Also

- ":MEASure:FFT:FREQuency" on page 939
- ":MEASure:FFT:MAGNitude" on page 941
- ":MEASure:FFT:DFRequency" on page 935
- ":MEASure:FFT:DMAGnitude" on page 937

- ":MEASure:FFT:CPOWer" on page 934
- ":MEASure:FFT:OBW" on page 943
- ":MEASure:FFT:PSD" on page 944

History New in version 11.20.

:MEASure:FREQuency

Command

```
:MEASure:FREQuency [<source>[,<direction>]]
```

The :MEASure:FREQuency command measures the frequency of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels if standard thresholds are selected).

The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FREQuency command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   frequency = 1/(second rising edge time - first rising edge time)
else
   frequency = 1/(second falling edge time - first falling edge time)
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether the frequency is measured from rising edge to rising edge or from falling edge to falling edge. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether whether the frequency is measured from rising edge to rising edge or from falling edge to falling edge throughout the acquisition.

Example This example measures the frequency of the channel 1 waveform.

```
myScope.WriteString ":MEASure:FREQuency CHANnel1"
```

Query

```
:MEASure:FREQuency? [<source>[,<direction>]]
```

The :MEASure:FREQuency? query returns the measured frequency.

Returned Format

```
[:MEASure:FREQuency] <value>[,<result_state>]<NL>
```

<value> The frequency value in Hertz of the first complete cycle on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current frequency of the waveform in the numeric variable, varFreq, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:FREQuency? CHANnel1"
varFreq = myScope.ReadNumber
Debug.Print FormatNumber(varFreq, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:HISTogram:FWHM

Command

```
:MEASure:HISTogram:FWHM
```

When a histogram is on, the :MEASure:HISTogram:FWHM command installs a "Full-Width at Half Max" histogram measurement into the user interface's measurement Results pane.

Query

```
:MEASure:HISTogram:FWHM?
```

The :MEASure:HISTogram:FWHM? query returns the measured histogram "Full-Width at Half Max" value.

Returned Format

```
<value><NL>

<value> ::= width of histogram at half max height in NR3 format
```

See Also

- ":MEASure:HISTogram:HITS" on page 950
- ":MEASure:HISTogram:M1S" on page 951
- ":MEASure:HISTogram:M2S" on page 952
- ":MEASure:HISTogram:M3S" on page 953
- ":MEASure:HISTogram:MAX" on page 954
- ":MEASure:HISTogram:MEAN" on page 955
- ":MEASure:HISTogram:MEDian" on page 956
- ":MEASure:HISTogram:MIN" on page 957
- ":MEASure:HISTogram:MODE" on page 959
- ":MEASure:HISTogram:PEAK" on page 961
- ":MEASure:HISTogram:PP" on page 962
- ":MEASure:HISTogram:RESolution" on page 963
- ":MEASure:HISTogram:STDDev" on page 964

History

New in version 6.10.

:MEASure:HISTogram:HITS

Command
```
:MEASure:HISTogram:HITS [<source>]
```

The :MEASure:HISTogram:HITS command places the histogram hits measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the number of hits within the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:HITS WMEMory1"
```

Query
```
:MEASure:HISTogram:HITS? [<source>]
```

The :MEASure:HISTogram:HITS? query returns the number of hits within the histogram.

Returned Format
```
[:MEASure:HISTogram:HITS]<value>[,<result_state>]<NL>
```

<value> The number of hits in the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits within the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:HITS? WMEMory1"
varHisthits = myScope.ReadNumber
Debug.Print FormatNumber(varHisthits, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:M1S

Command

```
:MEASure:HISTogram:M1S [<source>]
```

The :MEASure:HISTogram:M1S command places the histogram percentage of points within one standard deviation of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example measures the percentage of points that are within one standard deviation of the mean of the histogram of the data stored in waveform memory 3.

```
myScope.WriteString ":MEASure:HISTogram:M1S WMEMory3"
```

Query

```
:MEASure:HISTogram:M1S? [<source>]
```

The :MEASure:HISTogram:M1S? query returns the measurement of the percentage of points within one standard deviation of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M1S]<value>[,<result_state>]<NL>
```

<value> The percentage of points within one standard deviation of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within one standard deviation of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M1S? WMEMory1"
varHistm1s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm1s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:M2S

Command

```
:MEASure:HISTogram:M2S [<source>]
```

The :MEASure:HISTogram:M2S command places the histogram percentage of points within two standard deviations of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example measures the percentage of points that are within two standard deviations of the mean of the histogram whose source is specified using the MEASure:SOURce command.

```
myScope.WriteString ":MEASure:HISTogram:M2S WMEMory1"
```

Query

```
:MEASure:HISTogram:M2S? [<source>]
```

The :MEASure:HISTogram:M2S? query returns the measurement of the percentage of points within two standard deviations of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M2S]<value>[,<result_state>]<NL>
```

<value> The percentage of points within two standard deviations of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within two standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M2S? WMEMory1"
varHistm2s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm2s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:M3S

Command

```
:MEASure:HISTogram:M3S [<source>]
```

The :MEASure:HISTogram:M2S command places the histogram percentage of points within two standard deviations of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example measures the percentage of points that are within three standard deviations of the mean of the histogram.

```
myScope.WriteString ":MEASure:HISTogram:M3S HISTogram"
```

Query

```
:MEASure:HISTogram:M3S? [<source>]
```

The :MEASure:HISTogram:M3S? query returns the measurement of the percentage of points within three standard deviations of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:M3S] <value>[,<result_state>]<NL>
```

<value> The percentage of points within three standard deviations of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. For a list of the result states, refer to the MEASure:RESults command.

Example This example returns the percentage of points within three standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M3S? WMEMory1"
varHistm3s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm3s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MAX

Command

```
:MEASure:HISTogram:MAX [<source>]
```

The :MEASure:HISTogram:MAX command places the histogram maximum value measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the maximum value of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MAX WMEMory1"
```

Query

```
:MEASure:HISTogram:MAX? [<source>]
```

The :MEASure:HISTogram:MAX? query returns the measurement of the maximum value of the histogram.

Returned Format

```
[:MEASure:HISTogram:MAX]<value>[,<result_state>]<NL>
```

<value> The maximum value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the maximum value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MAX?"
varHistmax = myScope.ReadNumber
Debug.Print FormatNumber(varHistmax, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MEAN

Command

```
:MEASure:HISTogram:MEAN [<source>]
```

The :MEASure:HISTogram:MEAN command places the histogram mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the mean of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MEAN WMEMory1"
```

Query

```
:MEASure:HISTogram:MEAN? [<source>]
```

The :MEASure:HISTogram:MEAN? query returns the measurement of the mean of the histogram.

Returned Format

```
[:MEASure:HISTogram:MEAN]<value>[,<result_state>]<NL>
```

<value> The mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MEAN? WMEMory1"
varHistmean = myScope.ReadNumber
Debug.Print FormatNumber(varHistmean, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MEDian

Command

```
:MEASure:HISTogram:MEDian [<source>]
```

The :MEASure:HISTogram:MEDian command places the histogram median measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the median of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MEDian WMEMory1"
```

Query

```
:MEASure:HISTogram:MEDian? [<source>]
```

The :MEASure:HISTogram:MEDian? query returns the measurement of the median of the histogram.

Returned Format

```
[:MEASure:HISTogram:MEDian]<value>[,<result_state>]<NL>
```

<value> The median of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the median of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MEDian? WMEMory1"
varHistmed = myScope.ReadNumber
Debug.Print FormatNumber(varHistmed, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MIN

Command `:MEASure:HISTogram:MIN [<source>]`

The :MEASure:HISTogram:MIN command places the histogram minimum measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the minimum the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MIN WMEMory1"
```

Query `:MEASure:HISTogram:MIN? [<source>]`

The :MEASure:HISTogram:MIN? query returns the measurement of the minimum value of the histogram.

Returned Format `[:MEASure:HISTogram:MIN]<value>[,<result_state>]<NL>`

<value> The minimum value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the minimum value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MIN?"
varHistmin = myScope.ReadNumber
Debug.Print FormatNumber(varHistmin, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MM3S

Command

```
:MEASure:HISTogram:MM3S [<source>]
```

The :MEASure:HISTogram:MM3S command installs the **μ-3σ** mean minus three standard deviations measurement in the Measurement Results area the graphical user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Query

```
:MEASure:HISTogram:MM3S? [<source>]
```

The :MEASure:HISTogram:MM3S? query returns the measured value of the mean minus three standard deviations.

Returned Format

```
[:MEASure:HISTogram:MM3S] <value>[,<result_state>]<NL>
```

<value> The mean minus three standard deviations value in NR3 format.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. For a list of the result states, refer to the :MEASure:RESults command.

See Also

- ":MEASure:SOURce" on page 1124
- ":MEASure:HISTogram:MP3S" on page 960
- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698
- ":MEASure:RESults?" on page 1081

History New in version 10.25.

:MEASure:HISTogram:MODE

Command

```
:MEASure:HISTogram:MODE [<source>]
```

The :MEASure:HISTogram:MODE command places the histogram mode measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the mode of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MODE WMEMory1"
```

Query

```
:MEASure:HISTogram:MODE? [<source>]
```

The :MEASure:HISTogram:MODE? query returns the measurement histogram's Mode value.

Returned Format

```
[:MEASure:HISTogram:MODE]<value>[,<result_state>]<NL>
```

<value> The Mode value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the Mode value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MODE? WMEMory1"
varHistMode = myScope.ReadNumber
Debug.Print FormatNumber(varHistMode, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History New in version 3.11.

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:MP3S

Command

```
:MEASure:HISTogram:MP3S [<source>]
```

The :MEASure:HISTogram:MP3S command installs the **μ+3σ** mean plus three standard deviations measurement in the Measurement Results area the graphical user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands apply only to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Query

```
:MEASure:HISTogram:MP3S? [<source>]
```

The :MEASure:HISTogram:MP3S? query returns the measured value of the mean plus three standard deviations.

Returned Format

```
[:MEASure:HISTogram:MP3S] <value>[,<result_state>]<NL>
```

<value> The mean plus three standard deviations value in NR3 format.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. For a list of the result states, refer to the :MEASure:RESults command.

See Also

- ":MEASure:SOURce" on page 1124
- ":MEASure:HISTogram:MM3S" on page 958
- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698
- ":MEASure:RESults?" on page 1081

History New in version 10.25.

:MEASure:HISTogram:PEAK

Command

```
:MEASure:HISTogram:PEAK [<source>]
```

The :MEASure:HISTogram:PEAK command places the histogram number of hits in the greatest peak measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the number of hits in the greatest peak of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:PEAK WMEMory1"
```

Query

```
:MEASure:HISTogram:PEAK? [<source>]
```

The :MEASure:HISTogram:PEAK? query returns the number of hits in the greatest peak of the histogram measurement.

Returned Format

```
[:MEASure:HISTogram:PEAK]<value>[,<result_state>]<NL>
```

<value> The number of hits in the histogram peak.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits in the greatest peak of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:PEAK? WMEMory1"
varHistpeak = myScope.ReadNumber
Debug.Print FormatNumber(varHistpeak, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:PP

Command

```
:MEASure:HISTogram:PP [<source>]
```

The :MEASure:HISTogram:PP command places the histogram width measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the width of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:PP WMEMory1"
```

Query

```
:MEASure:HISTogram:PP? [<source>]
```

The :MEASure:HISTogram:PP? query returns the measurement of the width of the histogram.

Returned Format

```
[:MEASure:HISTogram:PP]<value>[,<result_state>]<NL>
```

<value> The width of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the width of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:PP? WMEMory1"
varHistpp = myScope.ReadNumber
Debug.Print FormatNumber(varHistpp, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HISTogram:RESolution

Command

```
:MEASure:HISTogram:RESolution [<source>]
```

The :MEASure:HISTogram:RESolution command places the histogram bin width measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the bin width of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:RESolution WMEMory1"
```

Query

```
:MEASure:HISTogram:RESolution? [<source>]
```

The :MEASure:HISTogram:RES? query returns the measurement of the bin width of the histogram.

Returned Format

```
[:MEASure:HISTogram:RESolution]<value>[,<result_state>]<NL>
```

<value> The width of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the width of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:RESolution? WMEMory1"
varHistpp = myScope.ReadNumber
Debug.Print FormatNumber(varHistpp, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History New in version 3.50.

:MEASure:HISTogram:STDDev

Command

```
:MEASure:HISTogram:STDDev [<source>]
```

The :MEASure:HISTogram:STDDev command places the histogram standard deviation measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<R> | HISTogram}

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example places into the Measurements tab the standard deviation of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:STDDev WMEMory1"
```

Query

```
:MEASure:HISTogram:STDDev? [<source>]
```

The :MEASure:HISTogram:STDDev? query returns the measurement of standard deviation of the histogram.

Returned Format

```
[:MEASure:HISTogram:STDDev]<value>[,<result_state>]<NL>
```

<value> The standard deviation of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the standard deviation of the histogram whose source is specified using the MEASure:SOURce command and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:STDDEV? WMEMory1"
varHiststtd = myScope.ReadNumber
Debug.Print FormatNumber(varHiststtd, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667
- ":HISTogram:MODE" on page 698

History Legacy command (existed before version 3.10).

Version 3.50: Can now use this command with Meas Histogram math functions.

:MEASure:HOLDtime

Command

```
:MEASure:HOLDtime
   [<data_source>,<data_source_dir>,<clock_source>,<clock_ source_dir>]
```

The :MEASure:HOLDtime command measures the hold time between the specified clock and data sources.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<data_source>, <clock_source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<data_source _dir>

{RISing | FALLing | BOTH}

Selects the direction of the data source edge.

<clock_source _dir>

{RISing | FALLing}

Selects the direction of the clock source edge.

Example

This example measures the hold time from the rising edge of channel 1 to the rising edge of channel 2.

```
myScope.WriteString ":MEASure:HOLDtime CHAN1,RIS,CHAN2,RIS"
```

Query

```
:MEASure:HOLDtime?
   [<data_source>,<data_source_dir>,<clock_source>,<clock_ source_dir>]
```

The :MEASure:HOLDtime? query returns the measured hold time between the specified clock and data source.

The necessary waveform edges must be present on the display. Also, the "Measure All Edges" mode must be set (use the :ANALyze:AEDGes command or :MEASure:HOLDtime command before the query).

The query will return 9.99999E+37 if the necessary edges are not displayed or if the "Measure All Edges" mode is not currently set.

Returned Format

```
{:MEASure:SETuptime] <value><NL>
```

<value>

Hold time in seconds.

Example

This example places the current value of hold time in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HOLDtime? CHAN1,RIS,CHAN2,RIS"
```

```
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

See Also Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:JITTer:HISTogram

Command

NOTE This command is available only when the Jitter Analysis Software license is installed. Note, however, that you can also use the :FUNCtion<F>:MHIStogram command to display a measurement histogram.

```
:MEASure:JITTer:HISTogram {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:HISTogram command turns the measurement histogram display on or off when a jitter measurement is displayed.

Example This example turns the jitter measurement histogram display on.

```
myScope.WriteString ":MEASure:JITTer:HISTogram ON"
```

Query

```
:MEASure:JITTer:HISTogram?
```

The :MEASure :JITTer:HISTogram? query returns the state of measurement histogram display.

Returned Format

```
[:MEASure:JITTer:HISTogram] {1 | 0}
```

Example This example places the current setting of the jitter spectrum mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:HISTogram?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 667

History Legacy command (existed before version 3.10).

:MEASure:JITTer:MEASurement

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:MEASurement {MEASurement<N>}
```

The :MEASure :JITTer:MEASurement command selects which measurement displayed on the oscilloscope you are performing the jitter analysis on. MEASurement1 is the most recently added measurement.

<N> An integer, 1-40.

NOTE When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.

Example This example assigns measurement 2 to the jitter measurement analysis.

```
myScope.WriteString ":MEASure:JITTer:MEASurement MEASurement2"
```

Query

```
:MEASure:JITTer:MEASurement?
```

The :MEASure :JITTer:MEASurement? query returns the measurement number you are performing the jitter analysis on. If no measurements are being displayed on the oscilloscope, the query will return a null string.

Returned Format

```
[:MEASure:JITTer:MEASurement MEASurement<N>]
```

Example This example places the current measurement number that you are performing jitter analysis on in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:MEASurement?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASure:JITTer:SPECtrum

Command

NOTE This command is available only when the Jitter Analysis Software license is installed. Note, however, that you can also use the :FUNCtion<F>:MTRend command to display a measurement trend waveform and then the :FUNCtion<F>:FFTMagnitude command to display the spectrum of the measurement trend waveform.

```
:MEASure:JITTer:SPECtrum {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:SPECtrum command turns the jitter spectrum display on or off when a jitter measurement is displayed.

Example This example turns the jitter measurement spectrum display on.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum ON"
```

Query

```
:MEASure:JITTer:SPECtrum?
```

The :MEASure :JITTer:SPECtrum? query returns the state of jitter spectrum display.

Returned Format

```
[:MEASure:JITTer:SPECtrum] {1 | 0}
```

Example This example places the current setting of the jitter spectrum mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

See Also

- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978
- ":FUNCtion<F>:MTRend" on page 671
- ":FUNCtion<F>:FFTMagnitude" on page 645

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:HORizontal

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal {MANual}
```

The :MEASure:JITTer:SPECtrum:HORizontal command sets the jitter spectrum horizontal mode to manual. The AUTO mode, which automatically selected the horizontal scaling and center frequency, is no longer available.

In manual mode, you set your own horizontal scaling and center frequency values.

Example This example sets the jitter spectrum horizontal mode to automatic.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal MANual"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal?
```

The :MEASure:JITTer:SPECtrum:HORizontal? query returns the current jitter spectrum horizontal mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal] {MAN}
```

Example This example places the current setting of the jitter trend horizontal mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:HORizontal:POSition

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition <position>
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition command sets the jitter spectrum horizontal center frequency position.

<position> A real number for the center frequency position in Hertz.

Example This example sets the jitter spectrum horizontal center frequency position to 250 kHz.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal:POSition 250E3"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition?
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition? query returns the current jitter spectrum horizontal center frequency position setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal:POSition] <value><NL>
```

<value> The jitter spectrum horizontal center frequency setting.

Example This example places the current setting of the jitter trend horizontal center frequency position in the variable varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal:POSition?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:HORizontal:RANGe

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe command sets the jitter spectrum horizontal range.

<range> A real number for the horizontal frequency range in Hertz.

Example This example sets the jitter spectrum horizontal range to 10 GHz (1 GHz/div).

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal:RANGe 10E9"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe?
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe? query returns the current jitter spectrum horizontal range setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal:RANGe] <value><NL>
```

<value> The jitter spectrum horizontal range setting.

Example This example places the current setting of the jitter trend horizontal range in the variable varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:HORizontal:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:RESolution

Query

```
:MEASure:JITTer:SPECtrum:RESolution?
```

The :MEASure:JITTer:SPECtrum:RESolution? query returns returns the resolution bandwidth of the measurement analysis spectrum FFT.

Returned Format

```
<value><NL>

<value> ::= resolution BW in NR3 format
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History New in version 6.20.

:MEASure:JITTer:SPECtrum:VERTical

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:VERTical {MANual}
```

The :MEASure:JITTer:SPECtrum:VERTical command sets the jitter spectrum vertical mode to manual. The AUTO mode, which automatically selected the vertical scaling and offset, is no longer available.

Example This example sets the jitter spectrum vertical mode to manual.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical MANual"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical?
```

The :MEASure:JITTer:SPECtrum:VERTical? query returns the current jitter spectrum vertical mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical] {MAN}
```

Example This example places the current setting of the jitter spectrum vertical mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:VERTical:OFFSet

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:SPECtrum:VERTical:OFFSet command sets the jitter spectrum vertical offset.

<offset> A real number for the vertical offset of the jitter measurement spectrum.

Example This example sets the jitter spectrum vertical offset to 2 ns.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:OFFSet 10E-9"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet?
```

The :MEASure:JITTer:SPECtrum:VERTical:OFFSet? query returns the jitter spectrum vertical offset time.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:OFFSet] <value> [,<result_state>]<NL>
```

<value> The jitter vertical spectrum offset time setting.

Example This example places the current value of jitter spectrum vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:VERTical:RANGe

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:VERTical:RANGe command sets the jitter spectrum vertical range.

<range> A real number for the full-scale vertical range for the jitter measurement spectrum.

Example This example sets the jitter spectrum vertical range to 4 ns (500 ps/div X 8 div).

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:RANGe 4E-9"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe?
```

The :MEASure:JITTer:SPECtrum:VERTical:RANGe? query returns the jitter spectrum range time setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:RANGe] <value> [,<result_state>]<NL>
```

<value> The jitter spectrum vertical range setting.

Example This example places the current value of jitter spectrum vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History Legacy command (existed before version 3.10).

:MEASure:JITTer:SPECtrum:VERTical:TYPE

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:TYPE {LINear | LOGarithmic}
```

The :MEASure:JITTer:SPECtrum:VERTical:TYPE command lets you select either a LINear or a LOGarithmic vertical scale for the jitter spectrum plot.

Example This example sets a linear vertical scale for the jitter spectrum plot.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:TYPE LINear"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:TYPE?
```

The :MEASure:JITTer:SPECtrum:VERTical:TYPE? query returns the current jitter spectrum plot vertical scale setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:TYPE] {LINear | LOGarithmic}
```

Example This example places the current jitter spectrum plot vertical scale setting in the string variable strType, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:TYPE?"
strType = myScope.ReadString
Debug.Print strType
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:WINDow" on page 978

History New in version 3.10.

:MEASure:JITTer:SPECtrum:WINDow

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:SPECtrum:WINDow {RECTangular | HANNing | FLATtop
                                | BHARris | HAMMing}
```

The :MEASure:JITTer:SPECtrum:WINDow command sets the jitter spectrum window mode. For a description of the window modes, see ":FUNCtion<F>:FFT:WINDow" on page 643.

Example This example sets the jitter spectrum window mode to Hanning.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:WINDow HANNing"
```

Query

```
:MEASure:JITTer:SPECtrum:WINDow?
```

The :MEASure:JITTer:SPECtrum:WINDow? query returns the current jitter spectrum window mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:WINDow] {RECTangular | HANNing | FLATtop
                                  | BHARris | HAMMing}<NL>
```

Example This example places the current setting of the jitter spectrum window mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:WINDow?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:JITTer:SPECtrum" on page 969
- ":MEASure:JITTer:SPECtrum:HORizontal" on page 970
- ":MEASure:JITTer:SPECtrum:HORizontal:POSition" on page 971
- ":MEASure:JITTer:SPECtrum:HORizontal:RANGe" on page 972
- ":MEASure:JITTer:SPECtrum:RESolution" on page 973
- ":MEASure:JITTer:SPECtrum:VERTical" on page 974
- ":MEASure:JITTer:SPECtrum:VERTical:OFFSet" on page 975
- ":MEASure:JITTer:SPECtrum:VERTical:RANGe" on page 976
- ":MEASure:JITTer:SPECtrum:VERTical:TYPE" on page 977

History Legacy command (existed before version 3.10).

Version 3.11: Added the HAMMing window mode selection.

:MEASure:JITTer:TRENd

Command

NOTE **This command is available only when the Jitter Analysis Software license is installed. Note, however, that you can also use the :FUNCtion<F>:MTRend command to display a measurement trend waveform.**

```
:MEASure:JITTer:TRENd {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd command turns the jitter measurement trend display on or off. When on, trend plots measurement results time correlated to the waveform being measured.

Example This example turns the jitter measurement trend display on.

```
myScope.WriteString ":MEASure:JITTer:TRENd ON"
```

Query

```
:MEASure:JITTer:TRENd?
```

The :MEASure :JITTer:TRENd? query returns the state of jitter trend display.

Returned Format

```
[:MEASure:JITTer:TRENd] {1 | 0}
```

Example This example places the current setting of the jitter trend mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:TRENd?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":FUNCtion<F>:MTRend" on page 671

History Legacy command (existed before version 3.10).

:MEASure:JITTer:TRENd:SMOoth

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:TRENd:SMOoth {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd:SMOoth command sets jitter trend smoothing to on or off. When on, smoothing creates a running average smoothed by the number of points set by the :JITTer:TRENd:SMOoth:POINts command.

Example This example sets the jitter trend smoothing mode to on.

```
myScope.WriteString ":MEASure:JITTer:TRENd:SMOoth ON"
```

Query

```
:MEASure:JITTer:TRENd:SMOoth?
```

The :MEASure:JITTer:TRENd:SMOoth? query returns the current jitter trend smoothing mode setting.

Returned Format

```
[:MEASure:JITTer:TRENd:SMOoth] {1 | 0}
```

Example This example places the current setting of the jitter trend smoothing mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:TRENd:SMOoth?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:MEASure:JITTer:TRENd:SMOoth:POINts

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:TRENd:SMOoth:POINts <points>
```

The :MEASure:JITTer:TRENd:SMOoth:POINts command sets the number of points as a set size for the data smoothing feature.

<points> odd integers, 3 to 100001. If out of range, the number will be rounded to nearest lower odd integer.

Example This example sets the jitter trend smoothing points to 7.

```
myScope.WriteString ":MEASure:JITTer:TRENd:SMOoth:POINts 7"
```

Query

```
:MEASure:JITTer:TRENd:SMOoth:POINts?
```

The :MEASure:JITTer:TRENd:SMOoth:POINts? query returns the current setting for jitter trend smoothing points.

Returned Format

```
[:MEASure:JITTer:TRENd:SMOoth:POINts] <value><NL>
```

<value> The jitter offset smoothing points setting.

Example This example places the current value of jitter trend smoothing points in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:JITTer:TRENd:SMOoth:POINts?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:JITTer:TRENd:VERTical

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:JITTer:TRENd:VERTical {AUTO | MANual}
```

The :MEASure:JITTer:TRENd:VERTical command sets the jitter trend vertical mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the jitter trend vertical mode to automatic.

```
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical AUTO"
```

Query

```
:MEASure:JITTer:TRENd:VERTical?
```

The :MEASure:JITTer:TRENd:VERTical? query returns the current jitter trend vertical mode setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical] {AUTO | MANual}
```

Example This example places the current setting of the jitter trend vertical mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:MEASure:JITTer:TRENd:VERTical:OFFSet

Command

NOTE **This command is available only when the Jitter Analysis Software license is installed.**

```
:MEASure:JITTer:TRENd:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:TRENd:VERTical:OFFSet command sets the jitter trend vertical offset.

<offset> A real number for the vertical offset for the jitter measurement trend.

Example This example sets the jitter trend vertical offset to 100 ps.

```
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical:OFFSet 100E-12"
```

Query

```
:MEASure:JITTer:TRENd:VERTical:OFFSet?
```

The :MEASure:JITTer:TRENd:VERTical:OFFSet? query returns the jitter trend vertical offset setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical:OFFSet] <value><NL>
```

<value> The jitter vertical trend offset setting.

Example This example places the current value of jitter trend vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:JITTer:TRENd:VERTical:RANGe

Command

NOTE **This command is available only when the Jitter Analysis Software license is installed.**

```
:MEASure:JITTer:TRENd:VERTical:RANGe <range>
```

The :MEASure:JITTer:TRENd:VERTical:RANGe command sets the jitter trend vertical range.

<range> A real number for the full-scale vertical range for the jitter measurement trend.

Example This example sets the jitter trend vertical range to 4 ns (500 ps/div X 8 div).

```
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical:RANGe 4E-9"
```

Query

```
:MEASure:JITTer:TRENd:VERTical:RANGe?
```

The :MEASure:JITTer:TRENd:VERTical:RANGe? query returns the jitter trend vertical range setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical:RANGe] <value><NL>
```

<value> The jitter trend vertical range setting.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of jitter trend vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:JITTer:TRENd:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:MARK

Command

```
:MEASure:MARK <meas_name>,{{0 | OFF} | {1 | ON}}
```

The :MEASure:MARK command turns on or off "track measurement" markers for a specified measurement. "Track measurement" markers show you where the oscilloscope is making an automatic measurement.

<meas_name> This is the measurement name as returned by :MEASurement<N>:NAME? query or as returned by the :MEASure:RESults? query.

Query

```
:MEASure:MARK? <meas_name>
```

The :MEASure:MARK? query returns returns the "track measurement" marker results as comma-separated values.

Returned Format

```
<meas_marker_results><NL>

<meas_marker_results> ::= <marker_state>,<first_X>,<first_Y>,
   <second_X>,<second_Y>

<marker_state> ::= {0 | 1}

<first_X> ::= first horizontal marker value in NR3 format

<first_Y> ::= first vertical marker value in NR3 format

<second_X> ::= second horizontal marker value in NR3 format

<second_Y> ::= second vertical marker value in NR3 format
```

If the marker measurement does not apply or cannot be made or if the marker state is 0 (OFF), the infinity representation value (9.99999E+37) is returned.

See Also

- ":MEASurement<N>:NAME" on page 1218
- ":MEASure:RESults?" on page 1081

History New in version 10.10.

:MEASure:NAME

Command `:MEASure:NAME {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20},<name>`

The :MEASure:NAME commands sets the name of the specified measurement to whatever string is given to <name>. This enables you to give specific names to measurements displayed on the oscilloscope's screen.

<name> a quoted string

Query `:MEASure:NAME? {MEAS1 | MEAS2 | MEAS3 | ... | MEAS20}`

The :MEASure:NAME? query returns the name of the corresponding measurement.

History Legacy command (existed before version 3.10).

Version 5.00: Now 20 measurements to choose from.

:MEASure:NCJitter

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:NCJitter <source>,<direction>,<N>,<start>
```

The :MEASure:NCJitter command measures the N cycle jitter of the waveform. Another name for this measurement is "N period-period", where N is the number of cycles in the period.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}, specifies direction of waveform edge to make measurement.

<N> An integer, 1 to 99, the number of cycles in a group.

<start> An integer, 1 to <N> - 1, typically 1, the cycle to start measuring.

Example This example measures the N cycle jitter on channel 1, rising edge, 5 cycles in a group, starting on the first cycle of the waveform.

```
myScope.WriteString ":MEASure:NCJitter CHANnel1,RISing,5,1"
```

Query

```
:MEASure:NCJitter? <source>,<direction>,<N>,<start>
```

The :MEASure:NCJitter? query returns the measured N cycle jitter time of the waveform.

Returned Format

```
[:MEASure:NCJitter] <value>[,<result_state>]<NL>
```

<value> The N cycle jitter time of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of N cycle jitter in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NCJitter? CHANnel1,RIS,5,1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:TIEClock2" on page 1174
- ":MEASure:CTCJitter" on page 907
- ":MEASure:CTCPwidth" on page 911

- ":MEASure:CTCNwidth" on page 909
- ":MEASure:CTCDutycycle" on page 905

History Legacy command (existed before version 3.10).

:MEASure:NOISe

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe <source>, {VOLT | UNIT}, {ZERO | ONE | BOTH}
```

The :MEASure:NOISe command adds a Noise measurement to the oscilloscope display.

The parameters specify the input source to be measured, the units (in volts or unit amplitude), and whether "zeros", "ones", or both "zeros" and "ones" should be measured.

This command is the equivalent of adding a noise measurement via **Measure > Data > Noise** in the front panel user interface.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example adds a "ones" Noise measurement on channel 1, in volt units, to the oscilloscope display. The measurement results appear in the Measurements tab.

```
myScope.WriteString ":MEASure:NOISe CHANnel1,VOLT,ONE"
```

Query

```
:MEASure:NOISe? <source>, {VOLT | UNIT}, {ZERO | ONE | BOTH}
```

The :MEASure:NOISe? query returns the measured noise value.

NOTE This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEAS:NOIS] <measured_value><NL>
```

<measured_value> The measured "zeros", "ones", or both noise value in volts or unit amplitude.

Example This example places the measurement result in the varMeasuredNoise variable.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe? CHANnel1,VOLT,ONE"
varMeasuredNoise = myScope.ReadNumber
Debug.Print FormatNumber(varMeasuredNoise, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332

History New in version 3.50.

:MEASure:NOISe:ALL?

Query

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:ALL? <level>
```

The :MEASure:NOISe:ALL? query returns the NOISe measurement results for the specified level.

<level> The <level> syntax is:

- For NRZ: {ZERO | ONE}
- For PAM-3: {ZERO | ONE | TWO}
- For PAM-4: {ZERO | ONE | TWO | THRee}
- For PAM-6: {ZERO | ONE | TWO | THRee | FOUR | FIVE}
- For PAM-8: {ZERO | ONE | TWO | THRee | FOUR | FIVE | SIX | SEVen}

Returned Format These values are returned as comma separated values using the following format:

```
[:MEASure:NOISe:ALL<space>]
TI(<format>),<result>,<state>,
RN(<format>),<result>,<state>,
DI(<format>),<result>,<state>,
PI(<format>),<result>,<state>,
ABUI(<format>),<result>,<state>,
BUI(<format>),<result>,<state>,
ISI(<format>),<result>,<state>,
Count,<number_of_bits>,<state>,
Level,<nominal_level>,<state>,
Eye Height(<format>),<result>,<state>,<NL>
```

NOTE Whether some of these values are included or not depends on the setting of :MEASure:NOISe:METHod and :MEASure:NOISe:REPort.

For example, when :MEASure:NOISe:REPort or :MEASure:NOISe:METHod is SPECtral, the BUI and ABUI values are not returned, and there are two PI values (one "rms" and one "dd").

<space> White space (ASCII 32) character.

<format> The format value tells you something about how the measurement is made. For instance, TI(1E-12) means that the TI measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (dd) means the measurement uses a dual-Dirac delta model to derive the measurement. A format of (pp) means the measurement is a peak-to-peak measurement.

<result> The measured results for the NOISe measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<state> The measurement result state. See Table 16 for a list of values and descriptions of the result state value.

<number_of_ bits> The number of waveform bits that have been measured.

<nominal_level> The Level line returns the nominal one or zero level. The unit amplitude = the nominal one level – nominal zero level.

Example This example places the noise measurement result for "ones" in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:ALL? ONE"
strResults = myScope.ReadString
Debug.Print strResults
```

See Also

- ":MEASure:NOISe:METHod" on page 995
- ":MEASure:NOISe:REPort" on page 996

History New in version 3.50.

Version 4.10: New results can be returned depending on the :MEASure:NOISe:METHod and :MEASure:NOISe:REPort settings.

Version 5.50: The parameters TWO and THRee are available for PAM-4 signals.

Version 11.20: Parameters are added for levels in the PAM-6 and PAM-8 signal types.

:MEASure:NOISe:BANDwidth

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:BANDwidth {NARRow | WIDE}
```

The :MEASure:NOISe:BANDwidth command sets the type of filtering used to separate the data dependent noise from the random noise and the periodic noise.

Example This example sets the RN bandwidth to WIDE.

```
myScope.WriteString ":MEASure:NOISe:BANDwidth WIDE"
```

Query

```
:MEASure:NOISe:BANDwidth?
```

The :MEASure:NOISe:BANDwidth? query returns the RN bandwidth filter setting.

Returned Format

```
[:MEASure:NOISe:BANDwidth] {NARRow | WIDE}<NL>
```

Example This example places the RN filter setting the strFilter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:BANDwidth?"
strFilter = myScope.ReadString
Debug.Print strFilter
```

History New in version 3.50.

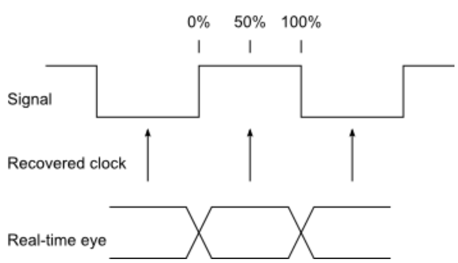
:MEASure:NOISe:LOCation

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:LOCation <location>
```

The :MEASure:NOISe:LOCation command specifies the measurement location within the bit where 0% is the beginning of the bit, 50% is the middle of the bit, and 100% is the end of the bit.



You can specify a location value from 5% to 95%.

Example This example sets the measurement location to 60%.

```
myScope.WriteString ":MEASure:NOISe:LOCation 60"
```

Query

```
:MEASure:NOISe:LOCation?
```

The :MEASure:NOISe:LOCation? query returns the measurement location setting.

Returned Format

```
[:MEASure:NOISe:LOCation] <location><NL>
```

Example This example places the measurement location setting the varLocation variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:LOCation?"
varLocation = myScope.ReadNumber
Debug.Print FormatNumber(varLocation, 0)
```

History New in version 3.50.

:MEASure:NOISe:METHod

Command

> **NOTE** This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:METHod {SPECtral | BOTH}
```

The :MEASure:NOISe:METHod command lets you select the method for random noise (RN) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.

When analyzing noise with crosstalk or ground bounce effects present in your signal, select BOTH. When this option is selected, the deterministic interference (DI) that is uncorrelated to the data pattern, also known as bounded uncorrelated interference (BUI), is separated into periodic interference (PI) and aperiodic bounded uncorrelated interference (ABUi). ABUi is caused by crosstalk and ground bounce effects.

When there are no crosstalk or ground bounce effects present in your signal, you can select the SPECtral method in order to run faster. When this option is selected, the deterministic interference (DI) that is uncorrelated to the data pattern is all reported as periodic interference (PI).

Example This example sets NOISe method to BOTH the spectral and tail fit analysis.

```
myScope.WriteString ":MEASure:NOISe:METHod BOTH"
```

Query

```
:MEASure:NOISe:METHod?
```

The :MEASure:NOISe:METHod? query returns the selected NOISe method.

Returned Format

```
[:MEASure:NOISe:METHod] {SPEC | BOTH}<NL>
```

Example This example places the NOISe method setting the strNoiseMethod variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:METHod?"
strNoiseMethod = myScope.ReadString
Debug.Print strNoiseMethod
```

See Also

- ":MEASure:NOISe:REPort" on page 996

History New in version 4.10.

:MEASure:NOISe:REPort

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:REPort {SPECtral | TAILfit}
```

When the :MEASure:NOISe:METHod BOTH command selects both the spectral and tail fit methods for random noise analysis, the :MEASure:NOISe:REPort command specifies which method is used for the reports in the noise graphs / histograms and Noise tab measurements.

Example This example specifies that the NOISe report include measurements from both the spectral and tail fit analysis (including aperiodic bounded uncorrelated interference ABUI measurements).

```
myScope.WriteString ":MEASure:NOISe:REPort TAILfit"
```

Query

```
:MEASure:NOISe:REPort?
```

The :MEASure:NOISe:REPort? query returns the report setting.

Returned Format

```
[:MEASure:NOISe:REPort] {SPEC | TAIL}<NL>
```

Example This example places the report setting in the strReportSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe:REPort?"
strReportSetting = myScope.ReadString
Debug.Print strReportSetting
```

See Also

- ":MEASure:NOISe:METHod" on page 995

History New in version 4.10.

:MEASure:NOISe:RN

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:RN {ON, <RNrms_level>,... | OFF}
```

The :MEASure:NOISe:RN command can specify a known amount of random noise. When used, the remaining amount of the total noise measured is reported as periodic interference (PI).

This command is used in situations when crosstalk aggressors influence the random noise measured on a signal. If the random noise on a signal is measured without the aggressor signal crosstalk, this known amount of random noise can be specified when measuring the noise again with the crosstalk aggressors.

- ON, <RNrms_level>,... — Enables a specified amount of random noise for each level in the signal type.
- OFF — Disables the specification of random noise amounts.

Specified amounts of random noise is shown in the noise measurement results (see page 991) as "RN(rms specified)".

<RNrms_level>,... For these signal types, specify values for the levels:

- For NRZ: <RNrms_zero>, <RNrms_one>
- For PAM-3: <RNrms_zero>, <RNrms_one>, <RNrms_two>
- For PAM-4: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>
- For PAM-6: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>, <RNrms_four>, <RNrms_five>
- For PAM-8: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>, <RNrms_four>, <RNrms_five>, <RNrms_six>, <RNrms_seven>

Example This example specifies 100 µV of known "zeros" random noise and 200 µV of known "ones" random noise.

```
myScope.WriteString ":MEAS:NOISE:RN ON, 100e-6, 200e-6"
```

Query

```
:MEASure:NOISe:RN?
```

The :MEASure:NOISe:RN? query returns the specified RN settings.

Returned Format

```
[:MEASure:NOISe:RN] {ON, <RNrms_level>,... | OFF}<NL>
```

Example This example places the specified RN settings in the strKnownRandomNoise variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:RN?"
```

```
strKnownRandomNoise = myScope.ReadString
Debug.Print strKnownRandomNoise
```

History New in version 3.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:NOISe:SCOPe:RN

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:SCOPe:RN {ON, <RNrms_level>,... | OFF}
```

The :MEASure:NOISe:SCOPe:RN command can specify the removal of the oscilloscope's calibrated random noise from the reported RN.

- ON, <RNrms_level>,... — Enables the removal of the oscilloscope's calibrated random noise from the reported RN for each level in the signal type.
- OFF — Disables the removal of the oscilloscope's calibrated random noise from the reported RN.

Running the **Calibrate scope jitter / noise** from the front panel user interface will set random noise to the measured values; however, the measures values can be changed by this command.

<RNrms_level>,... For these signal types, specify values for the levels:

- For NRZ: <RNrms_zero>, <RNrms_one>
- For PAM-3: <RNrms_zero>, <RNrms_one>, <RNrms_two>
- For PAM-4: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>
- For PAM-6: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>, <RNrms_four>, <RNrms_five>
- For PAM-8: <RNrms_zero>, <RNrms_one>, <RNrms_two>, <RNrms_three>, <RNrms_four>, <RNrms_five>, <RNrms_six>, <RNrms_seven>

Example This example specifies 100 µV of oscilloscope "zeros" random noise and 200 µV of oscilloscope "ones" random noise.

```
myScope.WriteString ":MEAS:NOISE:SCOPE:RN ON, 100e-6, 200e-6"
```

Query

```
:MEASure:NOISe:SCOPe:RN?
```

The :MEASure:NOISe:SCOPe:RN? query returns the oscilloscope RN settings.

Returned Format

```
[:MEASure:NOISe:SCOPe:RN] {ON, <RNrms_level>,... | OFF}<NL>
```

Example This example places the oscilloscope RN settings in the strScopeRandomNoise variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:SCOPe:RN?"
strScopeRandomNoise = myScope.ReadString
Debug.Print strScopeRandomNoise
```

History New in version 3.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:NOISe:STATe

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:STATe {ON | OFF}
```

The :MEASure:NOISe:STATe command enables or disables the NOISe measurements.

Example This example sets the NOISe state to on.

```
myScope.WriteString ":MEASure:NOISe:STATe ON"
```

Query

```
:MEASure:NOISe:STATe?
```

The :MEASure:NOISe:STATe? query returns the state of the NOISe measurements.

Returned Format

```
[:MEASure:NOISe:STATe] {1 | 0}<NL>
```

Example This example places the current state of the NOISe measurements in the varState variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:STATe?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

History New in version 3.50.

:MEASure:NOISe:UNITs

Command

NOTE This command is available only when the Jitter and Vertical Noise Analysis Software license is installed.

```
:MEASure:NOISe:UNITs {VOLT | UNIT}
```

The :MEASure:NOISe:UNITs command sets the unit of measure for NOISe measurements to volts or unit amplitude.

Example This example sets the NOISe units to unit amplitude.

```
myScope.WriteString ":MEASure:NOISe:UNITs UNIT"
```

Query

```
:MEASure:NOISe:UNITs?
```

The :MEASure:NOISe:UNITs? query returns the units of measure being used for the NOISe measurements.

Returned Format

```
[:MEASure:NOISe:UNITs] {VOLT | UNIT}<NL>
```

Example This example places the current units of measure for the NOISe measurements in the strUnits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NOISe:UNITs?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

History New in version 3.50.

:MEASure:NPERiod

Command

```
:MEASure:NPERiod <source>, <slope>, <N>
```

The :MEASure:NPERiod command measures the span of time of N consecutive periods. The measurement then moves over one period and measures the span of time of the next N consecutive periods.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<source> The source on which the measurement is made.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<slope> {RISing | FALLing}

<N> An integer greater than or equal to 1.

Example This example measures the time span of 3 consecutive periods on channel 1 (rising edge).

```
myScope.WriteString ":MEASure:NPERiod CHANnel1,RISing, 3"
```

Query

```
:MEASure:NPERiod?
```

History Legacy command (existed before version 3.10).

:MEASure:NPULses

Command

```
:MEASure:NPULses <source>
```

The :MEASure:NPULses measures the number of negative pulses on the screen.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<source> The source on which the measurement is made.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the number of negative pulses on channel 1.

```
myScope.WriteString ":MEASure:NPULses CHAN1"
```

Query :MEASure:NPULses?

This query returns the result for the NPULses measurement.

History Legacy command (existed before version 3.10).

:MEASure:NSIGma

Command

```
:MEASure:NSIGma [<source>[, {IEEE | PCIE | FILE[, <file path>]}
    [, {L0 | L1 | L2 | L3 | ALL}]]]
```

The :MEASure:NSIGma command installs a Sigma-n measurement on a PAM-4 waveform.

The Sigma-n measurement computes the noise parameter, σ_n (Sigma-n), for PAM-4 waveforms according to the PCIe 6.0 and IEEE 802.3 (PRBS13Q) standards. The parameter σ_n measures the uncorrelated RMS amplitude noise of the specified symbol level or all symbol levels (including random noise and bounded uncorrelated noise effects), while not including ISI (inter-symbol interference) and jitter effects.

Sigma-n (σ_n) is a component of Signal to Noise and Distortion Ratio (SNDR) measurements.

- IEEE (PRBS13Q) — This selection uses the technique outlined in the IEEE 802.3 specification clause 120D.3.1.6. The algorithm looks for six or more consecutive identical symbols and places the voltage measurement at the center of the 3rd UI. The variance of all those measurements is taken and then the standard deviations are averaged to get σ_n.

  The PRBS13Q pattern provides runs of at least six consecutive identical PAM-4 symbols at each of the PAM-4 levels.
- PCIE — This selection uses a process described in the PCIe 6.0 standard: See the oscilloscope's online help for more information.
- FILE[, <file_path>] — This selection allows for custom measurement parameters loaded from a file.

  The format of this file is proprietary. Contact Keysight Technical Support for more information.
- [, {L0 | L1 | L2 | L3 | ALL}] — This option specifies the symbol level on which the Sigma-n measurement is performed. ALL specifies that the measurement return a combined result for all four levels (using a root-square sum).

See the oscilloscope's online help for more information on the Sigma-n PAM-4 measurement.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L> | INPut | CORRected | ERRor | LFPR}

<file_path> A full path string referring to an XML file that defines the Sigma-n measurement custom parameters.

Query

```
:MEASure:NSIGma? [<source>[, {IEEE | PCIE | FILE[, <file path>]}
    [, {L0 | L1 | L2 | L3 | ALL}]]]
```

The :MEASure:NSIGma? query returns the value of a Sigma-n measurement on a PAM-4 waveform.

Returned Format `[:MEASure:NSIGma] <value>[,<result_state>]<NL>`

<value> A real number in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789
- ":MEASure:NSIGma" on page 1004
- ":MEASure:SLEVel" on page 1119
- ":MEASure:SLEVel" on page 1119
- ":MEASure:SNDR" on page 1122
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029

History New in version 11.15.

Version 11.25: Results can now be returned per symbol level.

:MEASure:NUI

Command

```
:MEASure:NUI <source>, <N>
```

The :MEASure:NUI command measures N consecutive unit intervals. The measurement then moves over one unit interval and measures the span of time of the next N consecutive unit intervals.

<source> The source on which the measurement is made.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<N> An integer greater than or equal to 1.

Example This example measures the time span of 3 consecutive unit intervals on channel 1.

```
myScope.WriteString ":MEASure:NUI CHAN1, 3"
```

Query

```
:MEASure:NUI?
```

The :MEASure:NUI? query returns the measured N-UI jitter.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:NWIDth

Command

```
:MEASure:NWIDth [<source>]
```

The :MEASure:NWIDth command measures the width of the first negative pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard threshold selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:NWIDth command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   nwidth = time at the second rising edge - time at the first
   falling edge
else
   nwidth = time at the first rising edge - time at the first
   falling edge
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the width of the first negative pulse on the screen.

```
myScope.WriteString ":MEASure:NWIDth CHANnel1"
```

Query

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query returns the measured width of the first negative pulse of the specified source.

Returned Format

```
[:MEASure:NWIDth] <value>[,<result_state>]<NL>
```

<value> The width of the first negative pulse on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current width of the first negative pulse on the screen in the numeric variable, varWidth, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:NWIDth? CHANnel1"
varWidth = myScope.ReadNumber
Debug.Print FormatNumber(varWidth, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:OERatio

Command

```
:MEASure:OERatio [<source>[, {RATio | DB | PERCent}]]
```

On a PAM-4 signal, through an optical probe, that has a pattern with at least seven consecutive 3s and six consecutive 0s, the :MEASure:OERatio command installs an outer Extinction Ratio measurement on a PAM-4 signal.

Outer ER is the ratio of a PAM-4 optical signal eye diagram's level 3 and level 0 symbols.

The level 3 and level 0 run requirements allow the signal level to settle. Be sure to check any relevant standard for its level 3 and level 0 run requirements.

For the central 2 UI of the run, the window for measuring the level's RMS value is determined. The level 0 and level 3 amplitude measurements are made over the central 2 UI of the run. The average amplitude for level 0 and for level 3 are calculated. The Outer ER measurement is the ratio of the average level 3 value to the average level 0 value.

<source> {CHANnel<N>}

Query

```
:MEASure:OERatio? [<source>[, {RATio | DB | PERCent}]]
```

The :MEASure:OERatio? query returns the outer Extinction Ratio measurement value in the specified units.

Returned Format

```
<value><NL>

<value> ::= a real number in NR3 format
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:OOMA" on page 1010

History New in version 10.25.

:MEASure:OMAMplitude

Command

```
:MEASure:OMAMplitude [<source>[,{WATT | DBM}]]
```

The :MEASure:OMAMplitude command installs an Optical Modulation Amplitude (OMA) measurement into the user interface's measurement Results pane.

Optical Modulation Amplitude (OMA) is the measure of the difference between the optical power of an NRZ (non-return-to-zero) one pulse and the optical power of an NRZ zero pulse. It requires an NRZ pattern and is designed to be used with a square wave made of consecutive zeros followed by consecutive ones. Be sure to check any relevant standard for one and zero run requirements. All instances are measured if Measure All Edges is selected. Otherwise, the edges closest to the timebase reference are measured.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

{WATT | DBM} Specifies the measurement units in Watts or dBm.

Query

```
:MEASure:OMAMplitude? [<source>[,{WATT | DBM}]]
```

The :MEASure:OMAMplitude? query returns the measured Optical Modulation Amplitude (OMA).

Returned Format

```
[:MEASure:OMAMplitude] <value>[,<result_state>]<NL>
```

<value> The measured Optical Modulation Amplitude (OMA) value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:ERATio" on page 928
- ":MEASure:OPOWer" on page 1011
- ":MEASure:CGRade:OLEVel" on page 899
- ":MEASure:CGRade:ZLEVel" on page 901

History New in version 5.70.

:MEASure:OOMA

Command

```
:MEASure:OOMA [<source>][, {WATT | DBM}]
```

On a PAM-4 signal, through an optical probe, that has a pattern with at least seven consecutive 3s and six consecutive 0s, the :MEASure:OOMA command installs an outer OMA (Optical Modulation Amplitude) measurement.

Outer OMA is the measure of the difference between the optical power of a PAM-4 signal's level 3 and level 0 symbols.

The level 3 and level 0 run requirements allow the signal level to settle. Be sure to check any relevant standard for its level 3 and level 0 run requirements.

The level measurements are made over the central 2 UI of the run. The average amplitude for level 3 and for level 0 are calculated. The Outer OMA measurement is calculated by subtracting the average level 0 value from the average level 3 value.

All instances are measured if Measure All Edges is selected. Otherwise, the edges closest to the timebase reference are measured.

<source> {CHANnel<N>}

Query

```
:MEASure:OOMA? [<source>][, {WATT | DBM}]
```

The :MEASure:OOMA? query returns the measured outer OMA (Optical Modulation Amplitude) value in the specified units.

Returned Format

```
<value><NL>

<value> ::= a real number in NR3 format
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:OERatio" on page 1008

History New in version 10.25.

:MEASure:OPOWer

Command `:MEASure:OPOWer [<source>[,{WATT | DBM}]]`

The :MEASure:OPOWer command installs an Optical Average Power measurement into the user interface's measurement Results pane.

Optical average power is a measure of the true average component of an optical signal. If markers are tracking this measurement, the marker is placed on the optical power Watts. This measurement is commonly used when identifying the fundamental parameters of a lightwave transmitter. However, it differs from other measurements because it does not rely on the waveform display to determine the measurement. The analog-to-digital converter is in the probe itself, independent of the waveform displayed on the screen. You can measure the optical power of an eye diagram.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

{WATT | DBM} Specifies the measurement units in Watts or dBm.

Query `:MEASure:OPOWer? [<source>[,{WATT | DBM}]]`

The :MEASure:OPOWer? query returns the measured Optical Average Power.

Returned Format `[:MEASure:OMAMplitude] <value>[,<result_state>]<NL>`

<value> The measured Optical Average Power value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:ERATio" on page 928
- ":MEASure:OMAMplitude" on page 1009
- ":MEASure:CGRade:OLEVel" on page 899
- ":MEASure:CGRade:ZLEVel" on page 901

History New in version 5.70.

:MEASure:OVERshoot

Command

```
:MEASure:OVERshoot [<source>[,<direction>]]
```

The :MEASure:OVERshoot command measures the overshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:OVERshoot command.

The algorithm is:

```
If the first edge on the screen is rising,
then
    overshoot = (Local Vmax - Vtop) / Vamplitude
else
    overshoot = (Vbase - Local Vmin) / Vamplitude
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether rising edge overshoot or falling edge overshoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether rising edge overshoot or falling edge overshoot is measured throughout the acquisition.

Example This example measures the overshoot of the first edge on the screen.

```
myScope.WriteString ":MEASure:OVERshoot CHANnel1"
```

Query

```
:MEASure:OVERshoot? [<source>[,<direction>]]
```

The :MEASure:OVERshoot? query returns the measured overshoot of the specified source.

Returned Format

```
[:MEASure:OVERshoot] <value>[,<result_state>]<NL>
```

<value> Ratio of overshoot to amplitude, in percent.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of overshoot in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:OVERshoot? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:PAM:ELEVel

Command

```
:MEASure:PAM:ELEVel [<source>[,<threshold>]]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:PAM:ELEVel command installs a vertical center measurement of the specified PAM eye into the user interface's measurement Results pane.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<threshold>

Specifies eye to measure as an integer.

- For PAM-3, this may be from 0-1.
- For PAM-4, this may be from 0-2.
- For PAM-6, this may be from 0-4.
- For PAM-8, this may be from 0-6.

Query

```
:MEASure:PAM:ELEVel? [<source>[,<threshold>]]
```

The :MEASure:PAM:ELEVel? query returns returns the measured vertical center value of the specified PAM eye.

Returned Format

```
[:MEASure:PAM:ELEVel] <value><NL>

<value> ::= the vertical center value of the specified PAM eye in NR3 fo
rmat.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137

- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Eyes for PAM-6 and PAM-8 are now supported.

:MEASure:PAM:ESKew

Command

```
:MEASure:PAM:ESKew [<source>[,<threshold>[,<units>]]]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:PAM:ESKew command installs a horizontal center skew measurement of the specified PAM eye into the user interface's measurement Results pane.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<threshold> Specifies eye to measure as an integer.

- For PAM-3, this may be from 0-1.
- For PAM-4, this may be from 0-2.
- For PAM-6, this may be from 0-4.
- For PAM-8, this may be from 0-6.

<units> {SECond | UNITinterval}

Lets you choose the measurement units. If <units> is omitted, the last specified units are used.

Query

```
:MEASure:PAM:ESKew? [<source>[,<threshold>[,<units>]]]
```

The :MEASure:PAM:ESKew? query returns the measured horizontal center skew value of the specified PAM eye.

Returned Format

```
[:MEASure:PAM:ESKew] <value><NL>
```

<value> The horizontal center skew value of the specified PAM eye in NR3 format.

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031

- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.10: Added the ability to select units in UNITinterval or SECond. This control has been in the graphical user interface and is now available in the remote SCPI command.

Version 11.20: Eyes for PAM-6 and PAM-8 are now supported.

:MEASure:PAM:EYE:ELMethod

Command

```
:MEASure:PAM:EYE:ELMethod <method>
```

The :MEASure:PAM:EYE:ELMethod command selects the basis for determining the location of an eye's center on the waveform.

<method> {MEWidth | MEHeight}

- MEWidth — The eye's center is located at the eye's maximum width midway between the eye's inside left and right edges. This is the default setting.
- MEHeight — The eye's center is located at the eye's maximum height midway between the eye's inside top and bottom edges.

Query

```
:MEASure:PAM:EYE:ELMethod?
```

The :MEASure:PAM:EYE:ELMethod? query returns the selected method for determining the location of an eye's center.

Returned Format

```
<method><NL>

<method> ::= {MEW | MEH}
```

See Also

- ":MEASure:PAM:EYE:ESTiming" on page 1019
- ":MEASure:PAM:EYE:PPERcent" on page 1020
- ":MEASure:PAM:EYE:TIME:LTDefinition" on page 1022
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 6.10.

:MEASure:PAM:EYE:ESTiming

Command

```
:MEASure:PAM:EYE:ESTiming <timing>
```

The :MEASure:PAM:EYE:ESTiming command configures the timing for sampling the data. Use this setting to match the timing method used by a receiver. For signals with skewed eyes, measurements results will vary depending on this setting.

<timing> {CEYE | PELevel}

- CEYE — Sampling timing is based on the location of the center eye (of the three stacked eyes). This is the default setting.
- PELevel — Sampling timing is independently based on the location of each of the three eyes.

Query

```
:MEASure:PAM:EYE:ESTiming?
```

The :MEASure:PAM:EYE:ESTiming? query returns the selected "timing for sampling" setting.

Returned Format

```
<timing><NL>

<timing> ::= {CEYE | PEL}
```

See Also

- ":MEASure:PAM:EYE:ELMethod" on page 1018
- ":MEASure:PAM:EYE:PPERcent" on page 1020
- ":MEASure:PAM:EYE:TIME:LTDefinition" on page 1022
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 6.10.

:MEASure:PAM:EYE:PPERcent

Command `:MEASure:PAM:EYE:PPERcent <percentage>`

The :MEASure:PAM:EYE:PPERcent command defines the time span over which an eye's amplitude level is measured. The time span is specified as a percentage of the symbol period.

<percentage> Percentage of symbol period in NR3 format. The default setting is 10%.

Query `:MEASure:PAM:EYE:PPERcent?`

The :MEASure:PAM:EYE:PPERcent? query returns the "eye level width" time span setting.

Returned Format `<percentage><NL>`

See Also

- ":MEASure:PAM:EYE:ELMethod" on page 1018
- ":MEASure:PAM:EYE:ESTiming" on page 1019
- ":MEASure:PAM:EYE:TIME:LTDefinition" on page 1022
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 6.10.

:MEASure:PAM:EYE:PROBability

Command

```
:MEASure:PAM:EYE:PROBability {ZHITs | PRObability,<probability>}
```

When making PAM eye height or eye width measurements, the :MEASure:PAM:EYE:PROBability command specifies whether eye boundaries (from the center of each eye) are based on zero hits (ZHITs) or at an eye opening BER (Bit Error Ratio) probability (PROBability,<probability>).

The "at probability" setting defines the ratio of total hits in the waveform database column that can occur in the eye's opening. The eye opening probability can be set from 1.0E-01 to 1.0E-09. The default probability is 1.0E-02. No extrapolation is used to determine Eye Height or Eye Width at a specified probability.

Because the eye center time is determined by the measured eye height, Eye Skew measurements are indirectly affected by the eye measurement boundary setting.

<probability> The "at probability" value in NR3 format.

Query

```
:MEASure:PAM:EYE:PROBability?
```

The :MEASure:PAM:EYE:PROBability? query returns the eye measurement boundary setting.

Returned Format

```
<setting><NL>

<setting> ::= {ZHIT | PROB,<probability>}
```

See Also

- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:EYE:ELMethod" on page 1018
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 10.10.

:MEASure:PAM:EYE:TIME:LTDefinition

Command

```
:MEASure:PAM:EYE:TIME:LTDefinition <method>
```

The :MEASure:PAM:EYE:TIME:LTDefinition command specifies the method used to locate the time at which to measure an eye's level.

<method> {MRMS | ECENter}

- ECENter — The time of a level is the average of the centers of the adjacent eyes. This is the default setting.
- MRMS — The time is located within the eye's level width at the minimum level thickness (RMS values).

Query

```
:MEASure:PAM:EYE:TIME:LTDefinition?
```

The :MEASure:PAM:EYE:TIME:LTDefinition? query returns the specified method used to locate the time at which to measure an eye's level.

Returned Format

```
<method><NL>

<method> ::= {MRMS | ECEN}
```

See Also

- ":MEASure:PAM:EYE:ELMethod" on page 1018
- ":MEASure:PAM:EYE:ESTiming" on page 1019
- ":MEASure:PAM:EYE:PPERcent" on page 1020
- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 6.10.

:MEASure:PAM:EYE:VEC

Command

```
:MEASure:PAM:EYE:VEC <source>
```

The :MEASure:PAM:EYE:VEC command installs a VEC (Vertical Eye Closure) measurement on a PAM-4 eye.

For a PRBS13Q signal, vertical eye closure (in dB) is defined by the equation:

$$VEC = 20 \log_{10} \max \frac{AV_{upp}}{V_{upp}}, \frac{AV_{mid}}{V_{mid}}, \frac{AV_{low}}{V_{low}}$$

Where:

- V_{upp}, V_{mid}, V_{low} = the upper, middle, lower eye height at the eye opening probability specified by the :MEASure:PAM:EYE:PROBability command (the default is 10^{-5}).
- AV_{upp} = the amplitude of the upper eye, equal to VM3-VM2.
- AV_{mid} = the amplitude of the middle eye, equal to VM2-VM1.
- AV_{low} = the amplitude of the lower eye, equal to VM1-VM0.
- VM3 = the mean of the signal above VCupp within 0.025 UI of TCmid.
- VM2 = the mean of the signal between VCupp and VCmid within 0.025 UI of TCmid.
- VM1 = the mean of the signal between VCmid and VClow within 0.025 UI of TCmid.
- VM0 = the mean of the signal below VClow within 0.025 UI of TCmid.
- VCupp = the voltage center of the upper eye.
- VCmid = the voltage center of the middle eye.
- VClow = the voltage center of the lower eye.
- TCmid = the time center of the middle eye.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L>}

Query

```
:MEASure:PAM:EYE:VEC? <source>
```

The :MEASure:PAM:EYE:VEC? query returns the measured VEC (Vertical Eye Closure) in dB.

Returned Format

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

```
<value> ::= a real number in NR3 format
```

A "?" on the result indicates that the specified eye opening probability has not been achieved yet.

See Also

- ":MEASure:PAM:EYE:PROBability" on page 1021
- ":MEASure:PAM:EYE:ELMethod" on page 1018
- ":MEASure:PAM:EYE:ESTiming" on page 1019
- ":MEASure:PAM:EYE:PPERcent" on page 1020
- ":MEASure:PAM:EYE:TIME:LTDefinition" on page 1022

History New in version 10.25.

:MEASure:PAM:EYE:WSDeviation

Command

```
:MEASure:PAM:EYE:WSDeviation <std_dev_percent>
```

When a Gaussian window shape is selected (see :MEASure:PAM:EYE:WSHape), the :MEASure:PAM:EYE:WSDeviation command specifies the standard deviation (σ) of the Gaussian shape as a percent of the UI (Unit Interval). It specifies the overall width of the Gaussian shape.

The selected window shape specifies how data is weighted with respect to the recovered clock location (which is presumably the center of the eye). The window shape is used when calculating PAM-4 eye height and VEC (Vertical Eye Closure).

<std_dev_percent> Percent of UI value in NR3 format.

Query

```
:MEASure:PAM:EYE:WSDeviation?
```

The :MEASure:PAM:EYE:WSDeviation? query returns the standard deviation (σ) of the Gaussian shape as a percent of the UI (Unit Interval).

Returned Format

```
<std_dev_percent><NL>
```

See Also

- ":MEASure:PAM:EYE:WSHape" on page 1026
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:PAM:EYE:VEC" on page 1023

History New in version 11.30.

:MEASure:PAM:EYE:WSHape

Command

```
:MEASure:PAM:EYE:WSHape {BOXCar | GAUSsian}
```

The :MEASure:PAM:EYE:WSHape command specifies the window shape of how data is weighted with respect to the recovered clock location when calculating PAM-4 eye height and VEC (Vertical Eye Closure).

- BOXCar — Specifies that all data is weighted evenly.
- GAUSsian — Specifies that data closer to the recovered clock location is weighted more significantly than data farther away, according to a Gaussian shape.

  When GAUSsian is selected, the :MEASure:PAM:EYE:WSDeviation command lets you specify the standard deviation (σ) of the Gaussian shape as a percent of the UI (Unit Interval). The standard deviation value specifies the overall width of the Gaussian shape.

Query

```
:MEASure:PAM:EYE:WSHape?
```

The :MEASure:PAM:EYE:WSHape? query returns the specified window shape.

Returned Format

```
<window_shape><NL>

<window_shape> ::= {BOXC | GAUS}
```

See Also

- ":MEASure:PAM:EYE:WSDeviation" on page 1025
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:PAM:EYE:VEC" on page 1023

History

New in version 11.30.

:MEASure:PAM:LEVel

Command

```
:MEASure:PAM:LEVel [<source>[,<level>]]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:PAM:LEVel command installs a mean voltage measurement of the specified PAM level into the user interface's measurement Results pane.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<level> Specifies the PAM level to measure as an integer.

- For PAM-3, this may be from 0-2.
- For PAM-4, this may be from 0-3.
- For PAM-6, this may be from 0-5.
- For PAM-8, this may be from 0-7.

If omitted, the last specified level is used.

Query

```
:MEASure:PAM:LEVel? [<source>[,<level>]]
```

The :MEASure:PAM:LEVel? query returns the measured mean voltage value of the specified PAM level.

Returned Format

```
[:MEASure:PAM:LEVel] <value><NL>
```

```
<value> ::= the mean voltage value of the specified PAM level in NR3 for
mat.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135

- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:PAM:LRMS

Command

```
:MEASure:PAM:LRMS [<source>[,<level>]]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:PAM:LRMS command installs a RMS voltage measurement of the specified PAM level into the user interface's measurement Results pane.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<level> Specifies the PAM level to measure as an integer:

- For PAM-3, this may be from 0-2.
- For PAM-4, this may be from 0-3.
- For PAM-6, this may be from 0-5.
- For PAM-8, this may be from 0-7.

If omitted, the last specified level is used.

Query

```
:MEASure:PAM:LRMS? [<source>[,<level>]]
```

The :MEASure:PAM:LRMS? query returns the measured RMS voltage value of the specified PAM level.

Returned Format

```
[:MEASure:PAM:LRMS] <value><NL>

<value> ::= the RMS voltage value of the specified PAM level in NR3 form
at.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135

- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:PAM:LTHickness

Command

```
:MEASure:PAM:LTHickness [<source>[,<level>]]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:PAM:LTHickness command installs an eye diagram level thickness measurement of the specified PAM level into the user interface's measurement Results pane.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<level> Specifies the PAM level to measure as an integer:

- For PAM-3, this may be from 0-2.
- For PAM-4, this may be from 0-3.
- For PAM-6, this may be from 0-5.
- For PAM-8, this may be from 0-7.

If omitted, the last specified level is used.

Query

```
:MEASure:PAM:LTHickness? [<source>[,<level>]]
```

The :MEASure:PAM:LTHickness? query returns the measured thickness of the specified PAM level.

Returned Format

```
[:MEASure:PAM:LTHickness] <value><NL>

<value> ::= the thickness value of the specified PAM level in NR3 format
.
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129

- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:PAM:PRBS13q:COUNt

Command

```
:MEASure:PAM:PRBS13q:COUNt <report_count>
```

The :MEASure:PAM:PRBS13q:COUNt command lets you change the PRBS13Q edge (12-edge) jitter measurement count.

The IEEE 802.3bs standard specifies PRBS13Q edge (12-edge) jitter measurements must be made 10,000 times, but you can use this command to change the count that is actually used.

<report_count> Number of acquisitions to measure from 200 to 10000 in NR1 format.

Query

```
:MEASure:PAM:PRBS13q:COUNt?
```

The :MEASure:PAM:PRBS13q:COUNt? query returns the PRBS13Q edge (12-edge) jitter measurement count setting.

Returned Format

```
<report_count><NL>
```

See Also

- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048

History New in version 10.20.

:MEASure:PAM:PRBS13q:EDGE:EOJ

Query

```
:MEASure:PAM:PRBS13q:EDGE:EOJ?
```

When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled (by the :MEASure:PAM:PRBS13q:STATe command), the :MEASure:PAM:PRBS13q:EDGE:EOJ? query returns the measured PRBS13Q even-odd jitter (EOJ) values.

Returned Format

```
<comma-separated_values><NL>
```

The returned comma-separated values contain:

- A composite measurement value.
- Values for individual rising and falling edges (R13, F21, F30, R03, F10, R02, R12, R23, R01, F20, F32, and F31).
- A count of the number of edges measured.

  If the count is less than (<) the Jrms/J4u Report Count (see :MEASure:PAM:PRBS13q:COUNt), the returned value is the number of edges measured so far.

  If the count is greater than or equal to (>=) the Jrms/J4u Report Count, the returned value is the number of edges reflected in the measurement results.
- If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned. See Table 16 for the meaning of the result state codes.

The complete list of comma-separated values is:

(Composite[,<result state>], R13[,<result state>], F21[,<result state>], F30[,<result state>], R03[,<result state>], F10[,<result state>], R02[,<result state>], R12[,<result state>], R23[,<result state>], R01[,<result state>], F20[,<result state>], F32[,<result state>], F31[,<result state>], Count)

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048
- ":MEASure:SENDvalid" on page 1114

History New in version 10.20.

:MEASure:PAM:PRBS13q:EDGE:J3U

Query `:MEASure:PAM:PRBS13q:EDGE:J3U?`

When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled (by the :MEASure:PAM:PRBS13q:STATe command), the :MEASure:PAM:PRBS13q:EDGE:J3U? query returns the measured PRBS13Q J3u values.

Returned Format `<comma-separated_values><NL>`

The returned comma-separated values contain:

- A composite measurement value.
- Values for individual rising and falling edges (R13, F21, F30, R03, F10, R02, R12, R23, R01, F20, F32, and F31).
- A count of the number of edges measured.

  If the count is less than (<) the Jrms/J4u Report Count (see :MEASure:PAM:PRBS13q:COUNt), the returned value is the number of edges measured so far.

  If the count is greater than or equal to (>=) the Jrms/J4u Report Count, the returned value is the number of edges reflected in the measurement results.
- If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned. See Table 16 for the meaning of the result state codes.

The complete list of comma-separated values is:

(Composite[,<result state>], R13[,<result state>], F21[,<result state>], F30[,<result state>], R03[,<result state>], F10[,<result state>], R02[,<result state>], R12[,<result state>], R23[,<result state>], R01[,<result state>], F20[,<result state>], F32[,<result state>], F31[,<result state>], Count)

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048
- ":MEASure:SENDvalid" on page 1114

History New in version 10.20.

:MEASure:PAM:PRBS13q:EDGE:J4U

Query `:MEASure:PAM:PRBS13q:EDGE:J4U?`

When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled (by the :MEASure:PAM:PRBS13q:STATe command), the :MEASure:PAM:PRBS13q:EDGE:J4U? query returns the measured PRBS13Q J4u values.

Returned Format `<comma-separated_values><NL>`

The returned comma-separated values contain:

- A composite measurement value.
- Values for individual rising and falling edges (R13, F21, F30, R03, F10, R02, R12, R23, R01, F20, F32, and F31).
- A count of the number of edges measured.

  If the count is less than (<) the Jrms/J4u Report Count (see :MEASure:PAM:PRBS13q:COUNt), the returned value is the number of edges measured so far.

  If the count is greater than or equal to (>=) the Jrms/J4u Report Count, the returned value is the number of edges reflected in the measurement results.
- If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned. See Table 16 for the meaning of the result state codes.

The complete list of comma-separated values is:

(Composite[,<result state>], R13[,<result state>], F21[,<result state>], F30[,<result state>], R03[,<result state>], F10[,<result state>], R02[,<result state>], R12[,<result state>], R23[,<result state>], R01[,<result state>], F20[,<result state>], F32[,<result state>], F31[,<result state>], Count)

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048
- ":MEASure:SENDvalid" on page 1114

History New in version 10.20.

:MEASure:PAM:PRBS13q:EDGE:J6U

Query `:MEASure:PAM:PRBS13q:EDGE:J6U?`

When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled (by the :MEASure:PAM:PRBS13q:STATe command), the :MEASure:PAM:PRBS13q:EDGE:J6U? query returns the measured PRBS13Q J6U values.

Returned Format `<comma-separated_values><NL>`

The returned comma-separated values contain:

- A composite measurement value.
- Values for individual rising and falling edges (R13, F21, F30, R03, F10, R02, R12, R23, R01, F20, F32, and F31).
- A count of the number of edges measured.

  If the count is less than (<) the Jrms/J6u Report Count (see :MEASure:PAM:PRBS13q:COUNt), the returned value is the number of edges measured so far.

  If the count is greater than or equal to (>=) the Jrms/J6u Report Count, the returned value is the number of edges reflected in the measurement results.
- If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned. See Table 16 for the meaning of the result state codes.

The complete list of comma-separated values is:

(Composite[,<result state>], R13[,<result state>], F21[,<result state>], F30[,<result state>], R03[,<result state>], F10[,<result state>], R02[,<result state>], R12[,<result state>], R23[,<result state>], R01[,<result state>], F20[,<result state>], F32[,<result state>], F31[,<result state>], Count)

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048
- ":MEASure:SENDvalid" on page 1114

History New in version 11.15.

:MEASure:PAM:PRBS13q:EDGE:JRMS

Query `:MEASure:PAM:PRBS13q:EDGE:JRMS?`

When the signal type is PAM-4 and PRBS13Q edge (12-edge) jitter measurements are enabled (by the :MEASure:PAM:PRBS13q:STATe command), the :MEASure:PAM:PRBS13q:EDGE:JRMS? query returns the measured PRBS13Q Jrms values.

Returned Format `<comma-separated_values><NL>`

The returned comma-separated values contain:

- A composite measurement value.
- Values for individual rising and falling edges (R13, F21, F30, R03, F10, R02, R12, R23, R01, F20, F32, and F31).
- A count of the number of edges measured.

  If the count is less than (<) the Jrms/J4u Report Count (see :MEASure:PAM:PRBS13q:COUNt), the returned value is the number of edges measured so far.

  If the count is greater than or equal to (>=) the Jrms/J4u Report Count, the returned value is the number of edges reflected in the measurement results.
- If :MEASure:SENDvalid is ON, a result state for the composite value and each of the rising and falling edge values is returned. See Table 16 for the meaning of the result state codes.

The complete list of comma-separated values is:

(Composite[,<result state>], R13[,<result state>], F21[,<result state>], F30[,<result state>], R03[,<result state>], F10[,<result state>], R02[,<result state>], R12[,<result state>], R23[,<result state>], R01[,<result state>], F20[,<result state>], F32[,<result state>], F31[,<result state>], Count)

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048
- ":MEASure:SENDvalid" on page 1114

History New in version 10.20.

:MEASure:PAM:PRBS13q:HUNits

Command

```
:MEASure:PAM:PRBS13q:HUNits <graph_scale>
```

The :MEASure:PAM:PRBS13q:HUNits command specifies the PRBS13Q edge (12-edge) jitter measurement graph scale (either Linear or Logarithmic).

<graph_scale> {LINear | LOG}

Query

```
:MEASure:PAM:PRBS13q:HUNits?
```

The :MEASure:PAM:PRBS13q:HUNits? query returns the PRBS13Q edge (12-edge) jitter measurement graph scale setting.

Returned Format

```
<graph_scale><NL>
<graph_scale> ::= {LIN | LOG}
```

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048

History New in version 10.20.

:MEASure:PAM:PRBS13q:PATTern

Command

```
:MEASure:PAM:PRBS13q:PATTern {P13Q | P9Q | PCI6P1 | PCI6P2 | PCI6P3
   | PCI6P4 | FILE}
```

The :MEASure:PAM:PRBS13q:PATTern command specifies the edge definition for the 12-edge jitter measurements' data pattern:

- P13Q — uses the known edge definition for the PRBS13Q data pattern.
- P9Q — uses the known edge definition for the PRBS9Q data pattern.
- PCI6P1 — uses the 52-UI pattern 1 for making edge jitter measurements on the first combination of PCIe Gen6 edges.
- PCI6P2 — uses the 52-UI pattern 2 for making edge jitter measurements on the second combination of PCIe Gen6 edges.
- PCI6P3 — uses the 52-UI pattern 3 for making edge jitter measurements on the third combination of PCIe Gen6 edges.
- PCI6P4 — uses the 52-UI pattern 4 for making edge jitter measurements on the fourth combination of PCIe Gen6 edges.
- FILE — uses the edge definition specified in a file for a custom data pattern.

When the FILE option is selected, use the :MEASure:PAM:PRBS13q:PFILe command to specify the full-path location of the edge definition file.

Query

```
:MEASure:PAM:PRBS13q:PATTern?
```

The :MEASure:PAM:PRBS13q:PATTern? query returns the edge definition being used.

Returned Format

```
<option><NL>

<option> ::= {P13Q | P9Q | PCI6P1 | PCI6P2 | PCI6P3 | PCI6P4 | FILE}
```

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048

History

New in version 11.15.

Version 11.25: The previous PCI6 pattern is replaced by the PCI6P1, PCI6P2, PCI6P3, and PCI6P4 patterns.

:MEASure:PAM:PRBS13q:PFILe

Command `:MEASure:PAM:PRBS13q:PFILe <full-path_string>`

When a custom data pattern is being used, the :MEASure:PAM:PRBS13q:PFILe command specifies the full-path location of the edge definition file for that pattern.

The :MEASure:PAM:PRBS13q:PATTern command specifies whether edge definitions for PRBS13Q, PCIE6 (52 symbols), or a custom data pattern should be used.

<full-path_string> A quoted string of the full path of the edge definition file for a custom data pattern.

Query `:MEASure:PAM:PRBS13q:PFILe?`

The :MEASure:PAM:PRBS13q:PFILe? query returns the edge definition file's full-path quoted string.

Returned Format `<full-path_string><NL>`

`<full-path_string> ::= A quoted string.`

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:STATe" on page 1047
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048

History New in version 11.15.

:MEASure:PAM:PRBS13q:STATe

Command

```
:MEASure:PAM:PRBS13q:STATe <source>, {{ON | 1} | {OFF | 0}}
```

The :MEASure:PAM:PRBS13q:STATe command enables or disables the PRBS13Q edge (12-edge) jitter measurements on a source waveform.

Before you can enable PRBS13Q edge (12-edge) jitter measurements, the PAM4 signal type must be specified for the source waveform using the :ANALyze:SIGNal:TYPE command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L>}

Query

```
:MEASure:PAM:PRBS13q:STATe? <source>
```

The :MEASure:PAM:PRBS13q:STATe? query returns whether PRBS13Q edge (12-edge) jitter measurements are enabled for the source waveform.

Returned Format

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

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:UNITs" on page 1048

History New in version 10.20.

:MEASure:PAM:PRBS13q:UNITs

Command

```
:MEASure:PAM:PRBS13q:UNITs <units>
```

The :MEASure:PAM:PRBS13q:UNITs command specifies the PRBS13Q edge (12-edge) jitter measurement units (either Seconds or Unit Interval).

<units> {SECond | UNITinterval}

Query

```
:MEASure:PAM:PRBS13q:UNITs?
```

The :MEASure:PAM:PRBS13q:UNITs? query returns the PRBS13Q edge (12-edge) jitter measurement units setting.

Returned Format

```
<units><NL>

<units> ::= {SEC | UNIT}
```

See Also

- ":MEASure:PAM:PRBS13q:COUNt" on page 1033
- ":MEASure:PAM:PRBS13q:EDGE:EOJ" on page 1034
- ":MEASure:PAM:PRBS13q:EDGE:J3U" on page 1036
- ":MEASure:PAM:PRBS13q:EDGE:J4U" on page 1038
- ":MEASure:PAM:PRBS13q:EDGE:J6U" on page 1040
- ":MEASure:PAM:PRBS13q:EDGE:JRMS" on page 1042
- ":MEASure:PAM:PRBS13q:HUNits" on page 1044
- ":MEASure:PAM:PRBS13q:PATTern" on page 1045
- ":MEASure:PAM:PRBS13q:PFILe" on page 1046
- ":MEASure:PAM:PRBS13q:STATe" on page 1047

History New in version 10.20.

:MEASure:PAMPlitude

Command

```
:MEASure:PAMPlitude [<source>, <width>, <direction>]
```

The :MEASure:PAMPlitude command measures the pulse amplitude around the specified edge. There is only a single width applied to the top and base for the amplitude measurement.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<width> width to measure at the top and base of the pulse (in percent, 0-100)

<direction> the edge direction to measure (RISing or FALLing). The pulse measured is to the left and right of the specified edge.

Example This example measures the pulse amplitude around a rising edge (width set to 50%)

```
myScope.WriteString ":MEASure:PAMPlitude CHAN1, 50, RISing"
```

Query

```
:MEASure:PAMPlitude? [<source>, <width>, <direction>]
```

The :MEASure:PAMPlitude? query returns the pulse amplitude around the specified edge.

History Legacy command (existed before version 3.10).

:MEASure:PBASe

Command

```
:MEASure:PBASe <source>, <pulse width percent>
```

The :MEASure:PBASe command measures the average of the data of a negative pulse within the pulse window. The pulse window is a range of data centered within the pulse width using the specified percentage of the data as measured as the middle threshold level. For example, a 50% window would not include in the average the first or last 25% of the pulse width as measured at the middle threshold level. A 100% window would measure the average of the entire positive or negative pulse. In measure all edges mode and EZJIT, these measurements can be trended, histogrammed, etc.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<pulse width percent> pulse width percent to use in average (in percent, 0-100)

Example This example measures the average of the data of a negative pulse within the pulse window (width set to 50%)

```
myScope.WriteString ":MEASure:PBASe CHAN1, 50"
```

Query

```
:MEASure:PBASe? <source>, <pulse width percent>
```

The :MEASure:PBASe? query returns the average pulse base of the data of a negative pulse within the specified pulse window.

History Legacy command (existed before version 3.10).

:MEASure:PERiod

Command

```
:MEASure:PERiod [<source>[,<direction>]]
```

The :MEASure:PERiod command measures the period of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected).

The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PERiod command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   period = second rising edge time - first rising edge time
else
   period = second falling edge time - first falling edge time
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether the period is measured from rising edge to rising edge or from falling edge to falling edge. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether whether the period is measured from rising edge to rising edge or from falling edge to falling edge throughout the acquisition.

Example This example measures the period of the waveform.

```
myScope.WriteString ":MEASure:PERiod CHANnel1"
```

Query

```
:MEASure:PERiod? [<source>[,<direction>]]
```

The :MEASure:PERiod? query returns the measured period of the specified source.

Returned Format

```
[:MEASure:PERiod] <value>[,<result_state>]<NL>
```

<value> Period of the first complete cycle on the screen.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current period of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:PERiod? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:PHASe

Command

```
:MEASure:PHASe [<source>[,<source>[,<direction>]]]
```

The :MEASure:PHASe command measures the phase in degrees between two edges. If two sources are specified, the phase from the specified edge of the first source to the specified edge of the second source is measured. If one source is specified, the phase is always 0.0E0.00°.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies direction of edge to measure. When <direction> is specified, the <source> parameter is required.

Example This example measures the phase between channel 1 and channel 2.

```
myScope.WriteString ":MEASure:PHASe CHANnel1,CHANnel2"
```

Query

```
:MEASure:PHASe? [<source>[,<source>[,<direction>]]]
```

The :MEASure:PHASe? query returns the measured phase angle value.

The necessary waveform edges must be present on the display. Also, the "Measure All Edges" mode must be set (use the :ANALyze:AEDGes command or :MEASure:PHASe command before the query).

The query will return 9.99999E+37 if the necessary edges are not displayed or if the "Measure All Edges" mode is not currently set.

Returned Format

```
[:MEASure:PHASe] <value>[,result_state]<NL>
```

<value> Phase angle from the first edge on the first source to the first edge on the second source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current phase angle value between channel 1 and channel 2 in the variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:PHASe? CHANnel1,CHANnel2"
```

```
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:PJITter

Command

```
:MEASure:PJITter PNOise, <start_freq>, <stop_freq>[, {SRMS | DBC}]
```

The :MEASure:PJITter command adds a Phase Jitter measurement on the phase noise single-sideband (SSB) frequency offset FFT plot.

The SRMS or DBC option lets you specify the measurement units in s(rms) or dBc, respectively. If you do not include this option, the most recent selection is used again.

<start_freq> Start frequency in Hz in NR3 format.

<stop_freq> Stop frequency in Hz in NR3 format.

Query

```
:MEASure:PJITter? PNOise, <start_freq>, <stop_freq>[, {SRMS | DBC}]
```

The :MEASure:PJITter? query returns the measured Phase Jitter value.

Returned Format

```
<measured_value><NL>

<measured_value> ::= phase jitter value in seconds in NR3 format
```

See Also

- ":MEASure:PN:STATe" on page 1068

History New in version 10.10.

Version 10.20: Added the SRMS or DBC option for specifying the measurement units in s(rms) or dBc, respectively.

:MEASure:PLENgth

Command

```
:MEASure:PLENgth <source>
```

The :MEASure:PLENgth command installs a Pattern Length measurement into the user interface's measurement Results pane.

The Pattern Length measurement looks for at least two error-free copies of an identical repeating bit pattern in acquisition memory. If a repeating bit pattern is found, its length is reported in the measurement results.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L> | INPut | CORRected | ERRor | LFPR}

Query

```
:MEASure:PLENgth? <source>
```

The :MEASure:PLENgth? query returns the measured pattern length.

Returned Format

```
<pattern_length><NL>
```

See Also

- ":MEASure:DATarate" on page 913
- ":MEASure:CDRRate" on page 886

History New in version 10.10.

:MEASure:PN:CORRelations

Command

```
:MEASure:PN:CORRelations <number>
```

When two clock sources permit the *two-channel cross-correlation technique* to be used, the :MEASure:PN:CORRelations command specifies the number of correlations that will be accumulated between phase noise analysis plot averages.

<number> The number of correlations from 1 to 65535 in NR1 format.

Query

```
:MEASure:PN:CORRelations?
```

The :MEASure:PN:CORRelations? query returns the specified number of correlations.

Returned Format

```
<number><NL>
```

See Also

- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:DESKew

Command `:MEASure:PN:DESKew`

When two channels are used (see :MEASure:PN:SOURce), the :MEASure:PN:DESKew command will automatically remove time skew due to different length cables connecting the device under test (DUT) to the oscilloscope. This automatic deskew can remove up to half a period of skew for clock-type signals. Greater skews than that must be removed manually (see :CHANnel<N>:PROBe:SKEW).

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071
- ":CHANnel<N>:PROBe:SKEW" on page 499

History New in version 11.20.

:MEASure:PN:EDGE

Command

```
:MEASure:PN:EDGE {RISing | FALLing | EITHer}
```

The :MEASure:PN:EDGE command specifies the clock edge direction on which to measure phase noise.

Query

```
:MEASure:PN:EDGE?
```

The :MEASure:PN:EDGE? query returns the specified clock edge direction.

Returned Format

```
<direction><NL>

<direction> ::= {RIS | FALL | EITH}
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:HORizontal:STARt

Command

```
:MEASure:PN:HORizontal:STARt <start_offset>
```

For the phase noise analysis single-sideband (SSB) frequency offset plot, the :MEASure:PN:HORizontal:STARt command specifies the left side of the horizontal log frequency scale.

<start_offset> Start offset frequency in Hz in NR3 format.

Query

```
:MEASure:PN:HORizontal:STARt?
```

The :MEASure:PN:HORizontal:STARt? query returns the phase noise analysis plot's left-side start offset frequency setting.

Returned Format

```
[:MEASure:PN:HORizontal:STARt] <start_offset><NL>
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:HORizontal:STOP

Command

```
:MEASure:PN:HORizontal:STOP <stop_offset>
```

For the phase noise analysis single-sideband (SSB) frequency offset plot, the :MEASure:PN:HORizontal:STOP command specifies the right side of the horizontal log frequency scale.

<stop_offset> Stop offset frequency in Hz in NR3 format.

Query

```
:MEASure:PN:HORizontal:STOP?
```

The :MEASure:PN:HORizontal:STOP? query returns the phase noise analysis plot's right-side stop offset frequency setting.

Returned Format

```
[:MEASure:PN:HORizontal:STOP] <stop_offset><NL>
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:INFO

Query `:MEASure:PN:INFO?`

The :MEASure:PN:INFO? query returns information about the number of averages and number of correlations that have occurred in the phase noise analysis.

Returned Format

```
<number_of_averages>,<number_of_correlations><NL>

<number_of_averages> ::= An integer in NR1 format

<number_of_correlations> ::= An integer in NR1 format
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 11.15.

:MEASure:PN:RSSC

Command `:MEASure:PN:RSSC {{0 | OFF} | {1 | ON}}`

If your clock source uses spread-spectrum clocking (SSC) and the FLATtop FFT windowing function is selected, you can use the :MEASure:PN:RSSC command to enable or disable the removal of the SSC effects from the phase noise analysis results.

Query `:MEASure:PN:RSSC?`

The :MEASure:PN:RSSC? query returns the "remove SSC" setting.

Returned Format `[:MEASure:PN:RSSC] <setting><NL>`

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

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:SOURce

Command

```
:MEASure:PN:SOURce <source1>[, <source2>]
```

The :MEASure:PN:SOURce command specifies the clock source(s) on which the phase noise analysis is performed.

<source1> {CHANnel<N> | FUNCtion<F>}

<source2> {CHANnel<N> | FUNCtion<F> | NONE}

The oscilloscope phase noise measurement floor is reduced by using two input channels (see "Two-Channel Cross-Correlation Lowers the Oscilloscope Noise Floor" in the oscilloscope's online help). You can split a single-ended signal into two copies or you can use both polarities of a differential signal.

If you do not want to use two input channels, select NONE. With only one input channel, the analysis is not able to lower the oscilloscope's phase noise measurement floor using two-channel cross-correlation.

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

Query

```
:MEASure:PN:SOURce?
```

The :MEASure:PN:SOURce? query returns the phase noise analysis clock source setup.

Returned Format

```
[:MEASure:PN:SOURce] <options><NL>

<options> ::= <source1>[, <source2>]

<source1> ::= {CHAN<N> | FUNC<F>}

<source2> ::= {CHAN<N> | FUNC<F> | NONE}
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070

- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:SPURs

Command

```
:MEASure:PN:SPURs {NORMalized | OMIT | POWer}
```

The :MEASure:PN:SPURs command specifies how to display spurs in the phase noise analysis single-sideband (SSB) frequency offset plot:

- NORMalized — This is the default setting. Spurs are displayed in the same normalized (dBc/Hz) scale as the rest of the phase noise analysis plot.
- OMIT — According to the :MEASure:PN:SSENsitivity setting, spurs are removed from the phase noise analysis plot.
- POWer — According to the :MEASure:PN:SSENsitivity setting, spurs are displayed in the power (dBc) setting. This shows a better representation of the energy at the spurs.

Query

```
:MEASure:PN:SPURs?
```

The :MEASure:PN:SPURs? query returns the current setting.

Returned Format

```
[:MEASure:PN:SPURs] <option><NL>

<option> ::= {NORM | OMIT | POW}
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:SSENsitivity

Command `:MEASure:PN:SSENsitivity <spur_sensitivity>`

When omitting spurs from the phase noise analysis single-sideband (SSB) frequency offset plot, or when displaying them in power (dBc) instead of the default normalized (dBc/Hz) scale, the :MEASure:PN:SSENsitivity command specifies the sensitivity used in identifying spurs.

<spur_sensitivity> 0.1 to 10.0 in NR3 format.

Query `:MEASure:PN:SSENsitivity?`

The :MEASure:PN:SSENsitivity? query returns the phase noise analysis spur sensitivity setting.

Returned Format `[:MEASure:PN:SSENsitivity] <spur_sensitivity><NL>`

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:STATe

Command

```
:MEASure:PN:STATe {{0 | OFF} | {1 | ON}}
```

The :MEASure:PN:STATe command turns the phase noise analysis feature on or off.

Query

```
:MEASure:PN:STATe?
```

The :MEASure:PN:STATe? query returns the phase noise analysis state setting.

Returned Format

```
[:MEASure:PN:STATe] <setting><NL>

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

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071
- ":MEASure:PJITter" on page 1055

History

New in version 10.10.

:MEASure:PN:VERTical:REFerence

Command `:MEASure:PN:VERTical:REFerence <level>`

The :MEASure:PN:VERTical:REFerence command specifies the dBc/Hz value at the top of the phase noise analysis single-sideband (SSB) frequency offset plot.

<level> Reference level in dBc/Hz in NR3 format.

Query `:MEASure:PN:VERTical:REFerence?`

The :MEASure:PN:VERTical:REFerence? query returns the phase noise analysis plot vertical reference setting.

Returned Format `[:MEASure:PN:VERTical:REFerence] <level><NL>`

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:VERTical:SCALe

Command

```
:MEASure:PN:VERTical:SCALe <scale_value>
```

The :MEASure:PN:VERTical:SCALe command specifies the height in dBc/Hz of each vertical division in the phase noise analysis single-sideband (SSB) frequency offset plot.

<scale_value> Scale in dBc/Hz per division in NR3 format.

Query

```
:MEASure:PN:VERTical:SCALe?
```

The :MEASure:PN:VERTical:SCALe? query returns the phase noise analysis plot vertical scale setting.

Returned Format

```
[:MEASure:PN:VERTical:SCALe] <scale_value><NL>
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.10.

:MEASure:PN:WINDow

Command

```
:MEASure:PN:WINDow <window_type>

<window_type> ::= {RECTangular | HANNing | FLATtop | BHARris | HAMMing}
```

The :MEASure:PN:WINDow command specifies the FFT windowing function used in the phase noise analysis.

The FLATtop window generally gives the best results.

Query

```
:MEASure:PN:WINDow?
```

The :MEASure:PN:WINDow? query returns the phase noise analysis FFT windowing function setting.

Returned Format

```
[:MEASure:PN:WINDow] <window_type><NL>

<window_type> ::= {RECT | HANN | FLAT | BHAR | HAMM}
```

See Also

- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:INFO" on page 1062
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":FUNCtion<F>:FFT:WINDow" on page 643

History

New in version 10.10.

:MEASure:PPContrast

Command

```
:MEASure:PPContrast [<source>]
```

The :MEASure:PPContrast command measures the peak-to-peak contrast (also known as Michelson contrast or modulation). This is the relation between the spread and the sum of two luminances.

Peak-to-Peak Contrast = (Lmax - Lmin) / (Lmax + Lmin)

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PPContrast command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the peak-to-peak contrast of channel 1.

```
myScope.WriteString ":MEASure:PPContrast CHANnel1"
```

Query

```
:MEASure:PPContrast? [<source>]
```

The :MEASure:PPContrast? query returns the measured peak-to-peak contrast value.

Returned Format

```
[:MEASure:PPContrast] <value>[,<result_state>]<NL>
```

<value> Peak-to-peak contrast of the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current peak-to-peak voltage in the numeric variable, varPPContrast, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:PPContrast? CHANnel1"
varPPContrast = myScope.ReadNumber
Debug.Print FormatNumber(varPPContrast, 0)
```

History New in version 5.60.

:MEASure:PPULses

Command `:MEASure:PPULses <source>`

The :MEASure:PPULses measures the number of positive pulses on the screen.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<source> The source on which the measurement is made.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the number of positive pulses on channel 1.

```
myScope.WriteString ":MEASure:PPULses CHANnel1"
```

Query :MEASure:PPULses?

This query returns the result for the PPULses measurement.

History Legacy command (existed before version 3.10).

:MEASure:PREShoot

Command

```
:MEASure:PREShoot [<source>[,<direction>]]
```

The :MEASure:PREShoot command measures the preshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PREShoot command.

The algorithm is:

```
If the first edge on the screen is rising,
then
    preshoot = (Vbase - Local Vmin) / Vamplitude
else
    preshoot = (Local Vmax - Vtop) / Vamplitude
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether rising edge preshoot or falling edge preshoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether rising edge preshoot or falling edge preshoot is measured throughout the acquisition.

Example This example measures the preshoot of the waveform on the screen.

```
myScope.WriteString ":MEASure:PREShoot CHANnel1"
```

Query

```
:MEASure:PREShoot? [<source>[,<direction>]]
```

The :MEASure:PREShoot? query returns the measured preshoot of the specified source.

Returned Format

```
[:MEASure:PREShoot] <value>[,<result state>]<NL>
```

<value> Ratio of preshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of preshoot in the numeric variable, varPreshoot, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:PREShoot? CHANnel1"
varPreshoot = myScope.ReadNumber
Debug.Print FormatNumber(varPreshoot, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:PTOP

Command

```
:MEASure:PTOP <source>, <pulse width percent>
```

The :MEASure:PTOP command measures the average of the data of a positive pulse within the pulse window. The pulse window is a range of data centered within the pulse width using the specified percentage of the data as measured as the middle threshold level. For example, a 50% window would not include in the average the first or last 25% of the pulse width as measured at the middle threshold level. A 100% window would measure the average of the entire positive or negative pulse. In measure all edges mode and EZJIT, these measurements can be trended, histogrammed, etc.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<pulse width percent>

pulse width percent to use in average (in percent, 0-100)

Example

This example measures the average of the data of a positive pulse within the pulse window (width set to 50%)

```
myScope.WriteString ":MEASure:PTOP CHANnel1, 50"
```

Query

```
:MEASure:PTOP? <source>, <pulse width percent>
```

The :MEASure:PTOP? query returns the average of the data of a positive pulse within the specified pulse window.

History

Legacy command (existed before version 3.10).

:MEASure:PWIDth

Command

```
:MEASure:PWIDth [<source>]
```

The :MEASure:PWIDth command measures the width of the first positive pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PWIDth command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   pwidth = time at the first falling edge - time at the
   first rising edge
else
   pwidth = time at the second falling edge - time at the
   first rising edge
```

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the width of the first positive pulse on the screen.

```
myScope.WriteString ":MEASure:PWIDth CHANnel1"
```

Query

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query returns the measured width of the first positive pulse of the specified source.

Returned Format

```
[:MEASure:PWIDth] <value>[,<result_state>]<NL>
```

<value> Width of the first positive pulse on the screen in seconds.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the width of the first positive pulse on the screen in the numeric variable, varWidth, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:PWIDth? CHANnel1"
varWidth = myScope.ReadNumber
Debug.Print FormatNumber(varWidth, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:QUALifier<M>:CONDition

Command

```
:MEASure:QUALifier<M>:CONDition {HIGH | LOW |
INSide | OUTSide}
```

The :MEASure:QUALifier<M>:CONDition command sets the condition when valid timing measurements are made

- Above Middle Threshold (HIGH)
- Below Middle Threshold (LOW)
- Between Upper, Lower Thresholds (INSide)
- Not Between Thresholds (OUTSide)

<M> An integer, 1-3.

Example This example sets the level qualifier 2 condition to HIGH.

```
myScope.WriteString ":MEASure:QUALifier2:CONDition HIGH"
```

Query

```
:MEASure:QUALifier<M>:CONDition?
```

The :MEASure:QUALifier<M>:CONDition? query returns the condition being used of the level qualifier.

Returned Format

```
[:MEASure:QUALifier<M>:CONDition] <source><NL>
```

Example This example places the current condition of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:QUALifier2:CONDition?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:QUALifier<M>:SOURce

Command

NOTE The channel being selected must not be used to make a timing measurement and must be turned on.

```
:MEASure:QUALifier<M>:SOURce <source>
```

The :MEASure:QUALifier<M>:SOURce command sets the source of the level qualify for timing measurements.

<source> CHANnel<N>

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 1-3.

Example This example sets the level qualifier 2 source to the channel 1 waveform.

```
myScope.WriteString ":MEASure:QUALifier2:SOURce CHANnel1"
```

Query

```
:MEASure:QUALifier<M>:SOURce?
```

The :MEASure:QUALifier<M>:SOURce? query returns the source being used of the level qualifier for timing measurements.

Returned Format

```
[:MEASure:QUALifier<M>:SOURce] <source><NL>
```

Example This example places the current source of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:QUALifier2:SOURce?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:QUALifier<M>:STATe

Command

```
:MEASure:QUALifier<M>:STATe {{ON | 1} | {OFF | 0}}
```

The :MEASure:QUALifier<M>:STATe command enables or disables level qualifying for timing measurements.

<M> An integer, 1-3.

Example This example sets the level qualifier 2 state to ON.

```
myScope.WriteString ":MEASure:QUALifier2:STATe ON"
```

Query

```
:MEASure:QUALifier<M>:STATe?
```

The :MEASure:QUALifier<M>:STATe? query returns the state of the level qualifier for timing measurements.

Returned Format

```
[:MEASure:QUALifier<M>:SOURce] {1 | 0}<NL>
```

Example This example places the current state of the level qualifier for timing measurements in the state variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:QUALifier2:STATe?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RESults?

Query `:MEASure:RESults? [AORDered | GORDered]`

The :MEASure:RESults? query returns the results of the measurements displayed on the front panel graphical user interface (GUI). Results are returned as a list of comma-separated values. If :MEASure:SENDvalid is ON, the result state is also returned.

If more than one measurement is displayed, the values for each measurement are returned according to the option:

- AORDered (or no option) — As measurements are added, either by a remote program or using the front panel GUI, they are displayed in the Results pane with the most recently added measurement at the top. The AORDered option (or no option) returns results in the inverse order that measurements were added. Reordering measurements on the front panel will not change this order.
- GORDered — This option always returns measurement results in the Results pane top-to-bottom order, even after reordering measurements on the front panel.

Up to 20 measurements can be displayed.

Returned Format `[:MEASure:RESults] <result_list><NL>`

<result_list> A list of the measurement results separated with commas. The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measure ment label	current	result state	min	max	mean	std dev	# of meas

Min, max, mean, std dev, and # of meas are returned only if :MEASure:STATistics is ON. The result state is returned only if :MEASure:SENDvalid is ON. See Table 16 for the meaning of the result state codes.

If the :MEASure:STATistics is set to CURRENT, MAX, MEAN, MIN, or STDDEV only that particular statistic value is returned for each displayed measurement.

Example This example places the current results of the measurements in the string variable, strResult, then prints the contents of the variable to the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":MEASure:RESults?"
strResult = myScope.ReadString
Debug.Print strResult
```

Table 16 Result States

Code	Description
0	Result correct. No problem found.
1	Result questionable but could be measured.
2	Result less than or equal to value returned.
3	Result greater than or equal to value returned.
4	Result returned is invalid.
5	Result invalid. Required edge not found.
6	Result invalid. Max not found.
7	Result invalid. Min not found.
8	Result invalid. Requested time not found.
9	Result invalid. Requested voltage not found.
10	Result invalid. Top and base are equal.
11	Result invalid. Measurement zone too small.
12	Result invalid. Lower threshold not on waveform.
13	Result invalid. Upper threshold not on waveform.
14	Result invalid. Upper and lower thresholds are too close.
15	Result invalid. Top not on waveform.
16	Result invalid. Base not on waveform.
17	Result invalid. Completion criteria not reached.
18	Result invalid. Measurement invalid for this type of waveform.
19	Result invalid. Waveform is not displayed.
20	Result invalid. Waveform is clipped high.
21	Result invalid. Waveform is clipped low.
22	Result invalid. Waveform is clipped high and low.
23	Result invalid. Data contains all holes.
24	Result invalid. No data on screen.
29	Result invalid. FFT peak not found.
30	Result invalid. Eye pattern not found.
31	Result invalid. No NRZ eye pattern found.

Table 16 Result States (continued)

Code	Description
32	Result invalid. The Extinction Ratio measurement will display a question mark ("?") result if: ▪ The dark calibration has not been performed at all. ▪ The vertical sensitivity, offset, or sample rate has changed since the dark calibration was run. ▪ The probe temperature has changed by > 2 °C. For more information on the extinction ratio measurement, see ":MEASure:ERATio" on page 928.
33	Result invalid. There is more than one source on creating the database.
35	Signal may be too small to evaluate.
36	Result invalid. Awaiting completion of averaging.
38	A clock signal waveform or an alternating ones and zeros data signal waveform is expected.
39	Result invalid. Need jitter package to make this measurement or must be in jitter mode to make this measurement.
40	Current measurement is not on screen.
41	Not enough points available to recover the clock.
42	The loop bandwidth of the PLL is too high to recover the clock.
43	RJDJ pattern not found in data.
45	Clock recovery mode is not permitted.
46	Too much jitter to make a RJDJ separation.
52	Signals have a different X-increment or sample rate.
53	Signals do not cross.
54	The signal has too many periodic characteristics for arbitrary analysis.
55	Ran out of memory computing measurement.
56	Lower threshold not on waveform, but noise is okay.
57	Upper threshold not on waveform, but noise is okay.
59	Too much noise to do noise separation.
60	Measurement is not valid for the specified signal type.
61	Measurement requires an open eye and there is no open eye. Applying equalization may be able to open the eye.
62	Measurement does not apply for this configuration.
63	Measurement will not work for this responsivity. See ":CHANnel<N>:PROBe:RESPonsivity" on page 498.

Table 16 Result States (continued)

Code	Description
64	The cross-correlation time range is too big. Lower the memory depth.
65	Invalid edge polarity.
66	Carrier frequency is not available.

See Also

- ":MEASure:SENDvalid" on page 1114
- ":MEASure:STATistics" on page 1125

History Legacy command (existed before version 3.10).

Version 6.20: There is now an AORDered option for returning the results according to the order in which measurements were added (ignoring any front panel graphical user interface (GUI) reordering) or a GORDered option for returning the results according to the order they appear on the display (even after front panel GUI reordering). Not specifying any option is the same as using the AORDered option.

:MEASure:RISetime

Command

```
:MEASure:RISetime [<source>[,<start_level>,<stop_level>]]
```

The :MEASure:RISetime command measures the rise time of the first displayed edge by measuring the time at the lower threshold of the rising edge, measuring the time at the upper threshold of the rising edge, then calculating the rise time with the following algorithm:

Rise time = time at upper threshold point - time at lower threshold point.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the RISetime command.

Four or more sample points on the rising edge of the waveform are required to make this measurement (one above the upper threshold, one below the lower threshold, and two between the thresholds).

With standard thresholds selected, the lower threshold is at the 10% point and the upper threshold is at the 90% point on the rising edge.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<start_level>, <stop_level> When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the <start_level> and <stop_level> parameters are integers that identify the edge to measure. For PAM-4, the levels may be from 0-3.

For PAM rise time measurements, "Measure All Edges" must be turned on (see :ANALyze:AEDGes).

Example This example measures the rise time of the channel 1 waveform.

```
myScope.WriteString ":MEASure:RISEtime CHANnel1"
```

Query

```
:MEASure:RISetime? [<source>[,<start_level>,<stop_level>]]
```

The :MEASure:RISetime? query returns the rise time of the specified source.

Returned Format

```
[:MEASure:RISetime] <value>[,<result_state>]<NL>
```

<value> Rise time in seconds.

A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement. Check the <result_state>. If the required four or more sample points are not present on the rising edge of the waveform, you can use a faster sampling rate or more interpolation points.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See Table 16 for a list of values and descriptions of the result state value.

Example This example places the current value of rise time in the numeric variable, varRise, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RISEtime? CHANnel1"
varRise = myScope.ReadNumber
Debug.Print FormatNumber(varRise, 0)
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

Version 5.50: With PAM signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.

Version 11.20: With PAM6 and PAM8 signal types, additional <start_level> and <stop_level> parameters are used to identify the edge to measure.

:MEASure:RJDJ:ALL?

Query

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:ALL?
```

The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements. These values are returned as comma separated triples of values using the following format:

Returned Format

```
[:MEASure:RJDJ:ALL<space>]
TJ(<format>),<result>,<state>,
RJ(<format>),<result>,<state>,
DJ(<format>),<result>,<state>,
PJ(<format>),<result>,<state>,
BUJ(<format>),<result>,<state>,
DDJ(<format>),<result>,<state>,
DCD,<result>,<state>,
ISI(<format>),<result>,<state>,
Transitions,<number_of_transitions>,<transitions_state>,
UTJ(<format>),<result>,<state>,
UDJ(<format>),<result>,<state>,
Scope RJ(<format>),<result>,<state>,
DDPWS,<result>,<state>,
ABUJ(<format>),<result>,<state><NL>
```

CAUTION Whether some of these values are included or not depends on the setting of :MEASure:RJDJ:METHod and :MEASure:RJDJ:REPort.

For example, when :MEASure:RJDJ:REPort or :MEASure:RJDJ:METHod is SPECtral, the BUJ and ABUJ values are not returned, and there are two PJ values (one "rms" and one "dd").

Also, new measurement results could be added in future versions.

The code that parses these results should be prepared for a dynamic list.

With PAM-4 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:ALL<space>]
TJ(<format>) 01,<result>,<state>,
TJ(<format>) 12,<result>,<state>,
TJ(<format>) 23,<result>,<state>,
RJ(<format>) 01,<result>,<state>,
RJ(<format>) 12,<result>,<state>,
RJ(<format>) 23,<result>,<state>,
DJ(<format>) 01,<result>,<state>,
DJ(<format>) 12,<result>,<state>,
```

```
DJ(<format>) 23,<result>,<state>,
Transitions 01,<number_of_transitions>,<transitions_state>,
Transitions 12,<number_of_transitions>,<transitions_state>,
Transitions 23,<number_of_transitions>,<transitions_state><NL>
```

With PAM-6 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:ALL<space>]
TJ(<format>) 01,<result>,<state>,
TJ(<format>) 12,<result>,<state>,
TJ(<format>) 23,<result>,<state>,
TJ(<format>) 34,<result>,<state>,
TJ(<format>) 45,<result>,<state>,
RJ(<format>) 01,<result>,<state>,
RJ(<format>) 12,<result>,<state>,
RJ(<format>) 23,<result>,<state>,
RJ(<format>) 34,<result>,<state>,
RJ(<format>) 45,<result>,<state>,
DJ(<format>) 01,<result>,<state>,
DJ(<format>) 12,<result>,<state>,
DJ(<format>) 23,<result>,<state>,
DJ(<format>) 34,<result>,<state>,
DJ(<format>) 45,<result>,<state>,
Transitions 01,<number_of_transitions>,<transitions_state>,
Transitions 12,<number_of_transitions>,<transitions_state>,
Transitions 23,<number_of_transitions>,<transitions_state>,
Transitions 34,<number_of_transitions>,<transitions_state>,
Transitions 45,<number_of_transitions>,<transitions_state><NL>
```

With PAM-8 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:ALL<space>]
TJ(<format>) 01,<result>,<state>,
TJ(<format>) 12,<result>,<state>,
TJ(<format>) 23,<result>,<state>,
TJ(<format>) 34,<result>,<state>,
TJ(<format>) 45,<result>,<state>,
TJ(<format>) 56,<result>,<state>,
TJ(<format>) 67,<result>,<state>,
RJ(<format>) 01,<result>,<state>,
RJ(<format>) 12,<result>,<state>,
RJ(<format>) 23,<result>,<state>,
RJ(<format>) 34,<result>,<state>,
RJ(<format>) 45,<result>,<state>,
RJ(<format>) 56,<result>,<state>,
RJ(<format>) 67,<result>,<state>,
DJ(<format>) 01,<result>,<state>,
DJ(<format>) 12,<result>,<state>,
DJ(<format>) 23,<result>,<state>,
DJ(<format>) 34,<result>,<state>,
DJ(<format>) 45,<result>,<state>,
DJ(<format>) 56,<result>,<state>,
DJ(<format>) 67,<result>,<state>,
```

```
Transitions 01,<number_of_transitions>,<transitions_state>,
Transitions 12,<number_of_transitions>,<transitions_state>,
Transitions 23,<number_of_transitions>,<transitions_state>,
Transitions 34,<number_of_transitions>,<transitions_state>,
Transitions 45,<number_of_transitions>,<transitions_state>,
Transitions 56,<number_of_transitions>,<transitions_state>,
Transitions 67,<number_of_transitions>,<transitions_state><NL>
```

Otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

<space> White space (ASCII 32) character.

<format> The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (dd) means the measurement uses a dual-Dirac delta model to derive the measurement. A format of (pp) means the measurement is a peak-to-peak measurement.

<result> The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<state> The measurement result state. See Table 16 for a list of values and descriptions of the result state value.

<number_of_transitions> The number of waveform transitions that have been measured.

Example This example places the jitter measures in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:ALL?"
strResults = myScope.ReadString
Debug.Print strResults
```

See Also

- ":MEASure:RJDJ:METHod" on page 1099
- ":MEASure:RJDJ:REPort" on page 1103
- ":MEASure:RJDJ:PAMThreshold" on page 1101

History Legacy command (existed before version 3.10).

Version 3.50: There are two possible additional measurement results, Scope RN(rms) and DDPWS.

Version 4.10: New results can be returned depending on the :MEASure:RJDJ:METHod and :MEASure:RJDJ:REPort settings.

Version 6.10: Jitter analysis is supported on PAM-4 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

Version 11.20: Jitter analysis is now supported on PAM-6 and PAM-8 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

Version 11.25: New jitter measurement results for Uncorrelated Total Jitter (UTJ(p-p)) and Uncorrelated Deterministic Jitter Delta-Delta (UDJδ-δ) have been added.

:MEASure:RJDJ:APLength?

Query

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:APLength?
```

When jitter or noise analysis is enabled and ":MEASure:RJDJ:PLENgth AUTO" option has been set to automatically detect the pattern length, the :MEASure:RJDJ:APLength? query returns the determined RjDj pattern length.

Returned Format

```
[:MEASure:RJDJ:APLength] <value><NL>
```

<value> The determined RjDj pattern length as a numeric data value.

When jitter or noise analysis is not enabled, this query returns an empty value and the -221,"Settings conflict" error message.

If the ":MEASure:RJDJ:PLENgth AUTO" option has not been set, or if there is no data, the value 9.99999E+37 is returned.

Example This example places the calculated pattern length in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:APLength?"
strResults = myScope.ReadString
Debug.Print strResults
```

See Also

- ":MEASure:RJDJ:PLENgth" on page 1102

History New in version 3.10.

:MEASure:RJDJ:BANDwidth

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:BANDwidth {NARRow | WIDE}
```

The :MEASure:RJDJ:BANDwidth command sets the type of filtering used to separate the data dependent jitter form the random jitter and the periodic jitter.

Example This example sets the RJ bandwidth to WIDE.

```
myScope.WriteString ":MEASure:RJDJ:BANDwidth WIDE"
```

Query

```
:MEASure:RJDJ:BANDwidth?
```

The :MEASure:RJDJ:BANDwidth? query returns the RJ bandwidth filter setting.

Returned Format

```
[:MEASure:RJDJ:BANDwidth] {NARRow | WIDE}<NL>
```

Example This example places the RJ filter setting the varFilter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:BANDwidth?"
varFilter = myScope.ReadNumber
Debug.Print FormatNumber(varFilter, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:BER

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:BER {E6 | E7 | E8 | E9 | E10 | E11 | E12 | E13 | E14
                  | E15 | E16 | E17 | E18 | J2 | J4 | J5 | J9}
```

The :MEASure:RJDJ:BER command sets the bit error rate for the Total Jitter (TJ) measurement. The E and J parameters have the following bit error rate meanings:

- E6 = 1E-6
- E7 = 1E-7
- E8 = 1E-8
- E9 = 1E-9
- E10 = 1E-10
- E11 = 1E-11
- E12 = 1E-12
- E13 = 1E-13
- E14 = 1E-14
- E15 = 1E-15
- E16 = 1E-16
- E17 = 1E-17
- E18 = 1E-18
- J2 = 2.5E-3
- J4 = 2.5E-5
- J5 = 2.5E-6
- J9 = 2.5E-10

Example This example sets the bit error rate to E16.

```
myScope.WriteString ":MEASure:RJDJ:BER E16"
```

Query

```
:MEASure:RJDJ:BER?
```

The :MEASure:RJDJ:BER? query returns the bit error rate setting.

Returned Format

```
[:MEASure:RJDJ:BER] {E6 | E7 | E8 | E9 | E10 | E11 | E12 | E13 | E14
                    | E15 | E16 | E17 | E18 | J2 | J4 | J5 | J9}<NL>
```

Example This example places the bit error rate in the varRate variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:BER?"
varRate = myScope.ReadNumber
Debug.Print FormatNumber(varRate, 0)
```

History Legacy command (existed before version 3.10).

Version 3.10: Added J2 and J9 jitter BER levels.

Version 5.75: Added J4 and J5 jitter BER levels.

:MEASure:RJDJ:CLOCk

Command

NOTE This command is available when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:CLOCk {ON | OFF}
```

When the :MEASure:RJDJ:CLOCk command is set to ON, it forces the pattern to be a clock and sets the jitter for edges not examined to zero (0).

To measure jitter on only rising (or falling) edges of a clock, you must also set :MEASure:CLOCk:METHod:EDGE to RISing or FALLing, and you must set :MEASure:RJDJ:EDGE to the same RISing or FALLing option.

Example This example turns on the RJDJ clock option.

```
myScope.WriteString ":MEASure:RJDJ:CLOCk ON"
```

Query

```
:MEASure:RJDJ:CLOCk?
```

The :MEASure:RJDJ:CLOCk? query returns the setting.

Returned Format

```
[:MEASure:RJDJ:CLOCk] {ON | OFF}<NL>
```

Example This example places the current RJDJ clock setting in the strSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:CLOCk?"
strSetting = myScope.ReadNumber
Debug.Print strSetting
```

See Also

- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":MEASure:RJDJ:EDGE" on page 1097

History New in version 4.30.

:MEASure:RJDJ:CREFerence

Command `:MEASure:RJDJ:CREFerence {0 | 1 | 2 | 3}`

The :MEASure:RJDJ:CREFerence command specifies the number of UI away from the data edge at which to measure jitter.

Some jitter measurements (for example, in DDR jitter tests) need to be made at several UI away from the data edge. If your measurements do not have these requirements, a value of zero (0) is normally used.

Query `:MEASure:RJDJ:CREFerence?`

The :MEASure:RJDJ:CREFerence? query returns the UI away from data edge setting.

Returned Format `<ui_from_edge><NL>`

`<ui_from_edge> ::= {0 | 1 | 2 | 3}`

History New in version 10.10.

:MEASure:RJDJ:EDGE

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:EDGE {RISing | FALLing | BOTH}
```

The :MEASure:RJDJ:EDGE command sets the edge used for the RJDJ measurements.

Example This example sets the RJDJ edge to use both edges.

```
myScope.WriteString ":MEASure:RJDJ:EDGE BOTH"
```

Query

```
:MEASure:RJDJ:EDGE?
```

The :MEASure:RJDJ:EDGE? query returns the edge being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:EDGE] {RIS | FALL | BOTH}<NL>
```

Example This example places the current edge being used for RJDJ measurements in the varEdge variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:EDGE?"
varEdge = myScope.ReadNumber
Debug.Print FormatNumber(varEdge, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:INTerpolate

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:INTerpolate {LINear | NONE}
```

The :MEASure:RJDJ:INTerpolate command sets the interpolation mode used for the RJDJ measurements.

Example This example sets the RJDJ interpolation to use both linear.

```
myScope.WriteString ":MEASure:RJDJ:INTerpolate LINear"
```

Query

```
:MEASure:RJDJ:INTerpolate?
```

The :MEASure:RJDJ:INTerpolate? query returns the interpolation mode being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:INTerpolate] {LINear | NONE}<NL>
```

Example This example places the current interpolation mode being used for RJDJ measurements in the interpolate variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:INTerpolate?"
varInterpolate = myScope.ReadNumber
Debug.Print FormatNumber(varInterpolate, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:METHod

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:METHod {SPECtral | BOTH}
```

The :MEASure:RJDJ:METHod command lets you select the method for random jitter (RJ) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.

When analyzing jitter with crosstalk or ground bounce effects present in your signal, select BOTH. When this option is selected, the deterministic jitter (DJ) that is uncorrelated to the data pattern, also known as bounded uncorrelated jitter (BUJ), is separated into periodic jitter (PJ) and aperiodic bounded uncorrelated jitter (ABUJ). ABUJ is caused by crosstalk and ground bounce effects.

When there are no crosstalk or ground bounce effects present in your signal, you can select the SPECtral method in order to run faster. When this option is selected, the deterministic jitter (DJ) that is uncorrelated to the data pattern is all reported as periodic jitter (PJ).

Example This example sets the RJDJ method to BOTH the spectral and tail fit analysis.

```
myScope.WriteString ":MEASure:RJDJ:METHod BOTH"
```

Query

```
:MEASure:RJDJ:METHod?
```

The :MEASure:RJDJ:METHod? query returns the selected RJDJ method.

Returned Format

```
[:MEASure:RJDJ:METHod] {SPEC | BOTH}<NL>
```

Example This example places the RJDJ method setting the strJitterMethod variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:METHod?"
strJitterMethod = myScope.ReadString
Debug.Print strJitterMethod
```

See Also

- ":MEASure:RJDJ:REPort" on page 1103

History New in version 4.10.

:MEASure:RJDJ:MODE

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:MODE {TIE | PERiod | NUI[,<ui>]}
```

The :MEASure:RJDJ:MODE command sets the RJDJ measurement mode. If NUI is selected then <ui> selects the number of unit intervals (for example: :MEASure:RJDJ:MODE NUI,5.

Example This example sets the RJDJ mode to TIE.

```
myScope.WriteString ":MEASure:RJDJ:MODE TIE"
```

Query

```
:MEASure:RJDJ:MODE?
```

The :MEASure:RJDJ:MODE? query returns the mode of the RJDJ measurements.

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:PAMThreshold

Command `:MEASure:RJDJ:PAMThreshold <level>`

<level> The :MEASure:RJDJ:PAMThreshold command specifies which PAM thresholds to measure for PAM signals. The <level> syntax is:

- For PAM-3: {T01 | T12 | ALL}
- For PAM-4: {T01 | T12 | T23 | ALL}
- For PAM-6: {T01 | T12 | T23 | T34 | T45 | ALL}
- For PAM-8: {T01 | T12 | T23 | T34 | T45 | T56 | T67 | ALL}

Query `:MEASure:RJDJ:PAMThreshold?`

The :MEASure:RJDJ:PAMThreshold? query returns the PAM threshold setting.

Returned Format `<level><NL>`

See Also

- ":DISPlay:JITTer:THReshold" on page 583
- ":MEASure:RJDJ:ALL?" on page 1087
- ":MEASure:RJDJ:TJRJDJ?" on page 1109

History New in version 6.10.

Version 11.20: Threshold levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:RJDJ:PLENgth

Command

NOTE **This command is available only when the Jitter Analysis Software license is installed.**

```
:MEASure:RJDJ:PLENgth {AUTO
    | {ARBitrary,<isi_filter_lead>,<isi_filter_lag>}
    | <number_of_bits>}
```

The :MEASure:RJDJ:PLENgth command sets the number of bits used pattern length for the RJDJ measurements.

<isi_filter_ lead> An integer number that is less than or equal to 0 that is the number of leading bits that are used to calculate the ISI filter.

<isi_filter_ lag} An integer number that is greater than or equal to 0 that is the number of trailing bits used to calculate the ISI filter.

<number_of_ bits> An integer number that is the length of pattern from 2 to 1024.

Example This example sets the RJDJ bits to 5.

```
myScope.WriteString ":MEASure:RJDJ:PLENgth 5"
```

Query

```
:MEASure:RJDJ:PLENgth?
```

The :MEASure:RJDJ:PLENgth? query returns the number of bits being used for the RJDJ measurements when Periodic pattern length is set. For Arbitrary pattern length, the ISI filter lead and filter lag numbers are returned.

Returned Format

```
[MEASure:RJDJ:PLENgth] {AUTO
    | ARBitrary,<isi_filter_lead>,<isi_filter_lag>
    | <number_of_bits>}<NL>
```

Example This example places the current number of bits for RJDJ measurements in the varBits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:PLENgth?"
varBits = myScope.ReadNumber
Debug.Print FormatNumber(varBits, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:REPort

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:REPort {SPECtral | TAILfit}
```

When the :MEASure:RJDJ:METHod BOTH command selects both the spectral and tail fit methods for random jitter analysis, the :MEASure:RJDJ:REPort command specifies which method is used for the reports in the jitter graphs/histograms and Jitter tab measurements.

Example This example specifies that the RJDJ report include measurements from both the spectral and tail fit analysis (including aperiodic bounded uncorrelated jitter ABUJ measurements).

```
myScope.WriteString ":MEASure:RJDJ:REPort TAILfit"
```

Query

```
:MEASure:RJDJ:REPort?
```

The :MEASure:RJDJ:REPort? query returns the report setting.

Returned Format

```
[:MEASure:RJDJ:REPort] {SPEC | TAIL}<NL>
```

Example This example places the report setting in the strReportSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:REPort?"
strReportSetting = myScope.ReadString
Debug.Print strReportSetting
```

See Also

- ":MEASure:RJDJ:METHod" on page 1099

History New in version 4.10.

:MEASure:RJDJ:RJ

Command

NOTE This command is available when the EZJIT Plus software is installed.

```
:MEASure:RJDJ:RJ {ON, <RJrms> | OFF}
```

The :MEASure:RJDJ:RJ command can specify a known amount of random jitter. When used, the remaining amount of the total jitter measured is reported as periodic jitter (PJ).

This command is used in situations when crosstalk aggressors influence the random jitter measured on a signal. If the random jitter on a signal is measured without the aggressor signal crosstalk, this known amount of random jitter can be specified when measuring the jitter again with the crosstalk aggressors.

- ON — Enables a specified amount of random jitter.
- <RJrms> — The known amount of random jitter.
- OFF — Disables the specification of known random jitter.

The amount of random jitter is shown in the jitter measurement results (see page 1087) as "RJ(rms specified)".

Example This example specifies 500 fs of random jitter.

```
myScope.WriteString ":MEAS:RJDJ:RJ ON, 500e-15"
```

Query

```
:MEASure:RJDJ:RJ?
```

The :MEASure:RJDJ:RJ? query returns the specified RJ settings.

Returned Format

```
[:MEASure:RJDJ:RJ] {ON, <RJrms> | OFF}<NL>
```

Example This example places the specified RJ settings in the strKnownRandomJitter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:RJ?"
strKnownRandomJitter = myScope.ReadString
Debug.Print strKnownRandomJitter
```

History New in version 3.50.

:MEASure:RJDJ:SCOPe:RJ

Command

NOTE This command is available when the EZJIT Plus software is installed.

```
:MEASure:RJDJ:SCOPe:RJ {ON, <RJrms> | AUTO | OFF}
```

The :MEASure:RJDJ:SCOPe:RJ command can specify the removal of the oscilloscope's calibrated random jitter from the reported RJ.

- ON — Enables the "manual" removal of a known oscilloscope random jitter from the reported RJ.
- <RJrms> — The known oscilloscope random jitter to remove from the reported RJ.
- AUTO — This option cannot be selected until the oscilloscope jitter calibration has been run (use the **Calibrate scope jitter** button in the front panel user interface). When selected, the calculated oscilloscope random jitter is removed from the reported RJ.

  The calculated oscilloscope random jitter is shown in the jitter measurement results (see page 1087) as "Scope RJ(rms)".
- OFF — Disables the removal of the oscilloscope's calibrated random jitter from the reported RJ.

Example This example specifies 300 fs of known oscilloscope random jitter.

```
myScope.WriteString ":MEASure:RJDJ:SCOPe:RJ ON, 300e-15"
```

Query

```
:MEASure:RJDJ:SCOPe:RJ?
```

The :MEASure:RJDJ:SCOPe:RJ? query returns the oscilloscope RJ settings.

Returned Format

```
[:MEASure:RJDJ:SCOPe:RJ] {ON, <RJrms> | OFF}<NL>
```

Example This example places the oscilloscope RJ settings in the strScopeRJSettings variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:SCOPe:RJ?"
strScopeRJSettings = myScope.ReadString
Debug.Print strScopeRJSettings
```

See Also

- ":MEASure:RJDJ:SCOPe:RJ:CALibrate" on page 1106

History New in version 3.50.

:MEASure:RJDJ:SCOPe:RJ:CALibrate

Command `:MEASure:RJDJ:SCOPe:RJ:CALibrate`

The :MEASure:RJDJ:SCOPe:RJ:CALibrate command calibrates the oscilloscope's jitter for jitter/noise analysis or for PAM-4 12-edge jitter measurements.

Sending this command is the same as clicking the **Calibrate scope jitter** button in the oscilloscope's front panel user interface "Remove, Specify Jitter, Noise" dialog box.

Before sending this command, you must:

- Disconnect or disable the input to the oscilloscope. The oscilloscope input must remain disconnected during the calibration.
- Enable jitter analysis, noise analysis, or PAM-4 12-edge jitter measurements.
- Specify one of the oscilloscope's analog input channels as the jitter or noise analysis source.
- Specify the **Custom: thresholds +/- hysteresis** thresholds mode.

See Also

- ":MEASure:RJDJ:SCOPe:RJ" on page 1105

History New in version 3.50.

Version 10.20: This command is also used to calibrates the oscilloscope's jitter for PAM-4 12-edge jitter measurements.

:MEASure:RJDJ:SOURce

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:SOURce <source>
```

The :MEASure:RJDJ:SOURce command sets the source for the RJDJ measurements.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the RJDJ source to the channel 1 waveform.

```
myScope.WriteString ":MEASure:RJDJ:SOURce CHANnel1"
```

Query

```
:MEASure:RJDJ:SOURce?
```

The :MEASure:RJDJ:SOURce? query returns the source being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:SOURce] <source><NL>
```

Example This example places the current source for RJDJ measurements in the varSource variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:SOURce?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:STATe

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:STATe {ON | OFF}
```

The :MEASure:RJDJ:STATe command enables or disables the RJDJ measurements.

Example This example sets the RJDJ state to on.

```
myScope.WriteString ":MEASure:RJDJ:STATe ON"
```

Query

```
:MEASure:RJDJ:STATe?
```

The :MEASure:RJDJ:STATe? query returns the state of the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:STATe] {1 | 0}<NL>
```

Example This example places the current state of the RJDJ measurements in the varState variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:STATe?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:RJDJ:TJRJDJ?

Query

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:TJRJDJ?
```

The :MEASure:RJDJ:TJRJDJ? query returns the Total Jitter (TJ), Random Jitter (RJ), and the Deterministic Jitter (DJ) measurements. These values are returned as comma separated triples of values using the following format:

Returned Format

```
[:MEASure:RJDJ:TJRJDJ] TJ(<tj_format>),<tj_result>,<tj_state>,
RJ(<rj_format>),<rj_result>,<rj_state>,
DJ(<dj_format>),<dj_result>,<dj_state><NL>
```

With PAM-4 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:TJRJDJ] TJ(<tj_format>) 01,<tj_result>,<tj_state>,
TJ(<tj_format>) 12,<tj_result>,<tj_state>,
TJ(<tj_format>) 23,<tj_result>,<tj_state>,
RJ(<rj_format>) 01,<rj_result>,<rj_state>,
RJ(<rj_format>) 12,<rj_result>,<rj_state>,
RJ(<rj_format>) 23,<rj_result>,<rj_state>,
DJ(<dj_format>) 01,<dj_result>,<dj_state>,
DJ(<dj_format>) 12,<dj_result>,<dj_state>,
DJ(<dj_format>) 23,<dj_result>,<dj_state><NL>
```

With PAM-6 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:TJRJDJ] TJ(<tj_format>) 01,<tj_result>,<tj_state>,
TJ(<tj_format>) 12,<tj_result>,<tj_state>,
TJ(<tj_format>) 23,<tj_result>,<tj_state>,
TJ(<tj_format>) 34,<tj_result>,<tj_state>,
TJ(<tj_format>) 45,<tj_result>,<tj_state>,
RJ(<rj_format>) 01,<rj_result>,<rj_state>,
RJ(<rj_format>) 12,<rj_result>,<rj_state>,
RJ(<rj_format>) 23,<rj_result>,<rj_state>,
RJ(<rj_format>) 34,<rj_result>,<rj_state>,
RJ(<rj_format>) 45,<rj_result>,<rj_state>,
DJ(<dj_format>) 01,<dj_result>,<dj_state>,
DJ(<dj_format>) 12,<dj_result>,<dj_state>,
DJ(<dj_format>) 23,<dj_result>,<dj_state>,
DJ(<dj_format>) 34,<dj_result>,<dj_state>,
DJ(<dj_format>) 45,<dj_result>,<dj_state><NL>
```

With PAM-8 signals, when the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels. For example:

```
[:MEASure:RJDJ:TJRJDJ] TJ(<tj_format>) 01,<tj_result>,<tj_state>,
TJ(<tj_format>) 12,<tj_result>,<tj_state>,
TJ(<tj_format>) 23,<tj_result>,<tj_state>,
TJ(<tj_format>) 34,<tj_result>,<tj_state>,
TJ(<tj_format>) 45,<tj_result>,<tj_state>,
TJ(<tj_format>) 56,<tj_result>,<tj_state>,
TJ(<tj_format>) 67,<tj_result>,<tj_state>,
RJ(<rj_format>) 01,<rj_result>,<rj_state>,
RJ(<rj_format>) 12,<rj_result>,<rj_state>,
RJ(<rj_format>) 23,<rj_result>,<rj_state>,
RJ(<rj_format>) 34,<rj_result>,<rj_state>,
RJ(<rj_format>) 45,<rj_result>,<rj_state>,
RJ(<rj_format>) 56,<rj_result>,<rj_state>,
RJ(<rj_format>) 67,<rj_result>,<rj_state>,
DJ(<dj_format>) 01,<dj_result>,<dj_state>,
DJ(<dj_format>) 12,<dj_result>,<dj_state>,
DJ(<dj_format>) 23,<dj_result>,<dj_state>,
DJ(<dj_format>) 34,<dj_result>,<dj_state>,
DJ(<dj_format>) 45,<dj_result>,<dj_state>,
DJ(<dj_format>) 56,<dj_result>,<dj_state>,
DJ(<dj_format>) 67,<dj_result>,<dj_state><NL>
```

Otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

<tj_format> <rj_format> <dj_format> The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (d-d) means the measurement uses from a dual-Dirac delta model used to derive the measurement. A format of (p-p) means the measurement is a peak-to-peak measurement.

<tj_result> <rj_result> <dj_result> The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<tj_state> <rj_state> <dj_state> The measurement result state. See Table 16 for a list of values and descriptions of the result state value.

Example This example places the Total Jitter (TJ), Random Jitter (RJ), and the Deterministic Jitter (DJ) measurements in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:TJRJDJ?"
strResult = myScope.ReadString
Debug.Print strResult
```

See Also

- ":MEASure:RJDJ:PAMThreshold" on page 1101

History Legacy command (existed before version 3.10).

Version 6.10: Jitter analysis is supported on PAM-4 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

Version 11.20: Jitter analysis is now supported on PAM-6 and PAM-8 signals. When the ":MEASure:RJDJ:PAMThreshold ALL" command setting has been made, the query results include values for multiple threshold levels; otherwise, the query results are for the specific threshold level specified in the :MEASure:RJDJ:PAMThreshold command.

:MEASure:RJDJ:UNITs

Command

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:RJDJ:UNITs {SECond | UNITinterval}
```

The :MEASure:RJDJ:UNITs command sets the unit of measure for RJDJ measurements to seconds or unit intervals.

Example This example sets the RJDJ units to unit interval.

```
myScope.WriteString ":MEASure:RJDJ:UNITs UNITinterval"
```

Query

```
:MEASure:RJDJ:UNITs?
```

The :MEASure:RJDJ:UNITs? query returns the units of measure being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:UNITs] {SECond | UNITinterval}<NL>
```

Example This example places the current units of measure for the RJDJ measurements in the varUnits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:UNITs?"
varUnits = myScope.ReadNumber
Debug.Print FormatNumber(varUnits, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:SCRatch

Command

```
:MEASure:{SCRatch | CLEar}
```

The :MEASure:SCRatch command clears the measurement results from the screen. This command performs the same function as :MEASure:CLEar.

Example This example clears the current measurement results from the screen.

```
myScope.WriteString ":MEASure:SCRatch"
```

History Legacy command (existed before version 3.10).

:MEASure:SENDvalid

Command

```
:MEASure:SENDvalid {{OFF|0} | {ON|1}}
```

The :MEASure:SENDvalid command enables the result state code to be returned with the :MEASure:RESults? query and all other measurement queries.

Example This example turns the send valid function on.

```
myScope.WriteString ":MEASure:SENDvalid ON"
```

Query

```
:MEASure:SENDvalid?
```

The :MEASure:SENDvalid? query returns the state of the send valid control.

Returned Format

```
{:MEASure:SENDvalid] {0 | 1}<NL>
```

Example This example places the current mode for SENDvalid in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASure:SENDvalid?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

History Legacy command (existed before version 3.10).

:MEASure:SER

Command `:MEASure:SER <source>`

When a pattern length and pattern can be determined (see the :ANALyze:SIGNal:PATTern:* commands), the :MEASure:SER command installs a cumulative symbol error rate (SER) measurement of the specified PAM waveform into the user interface's measurement Results pane.

With PAM signals, there can be multiple bits per symbol, and symbol error rate is related to bit error rate (BER). For example, with PAM-4 one symbol error can translate to one or two bit errors.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query `:MEASure:SER? <source>`

The :MEASure:SER? query returns the measured cumulative symbol error rate value.

Returned Format `[:MEASure:SER] <value><NL>`

`<value> ::= the cumulative SER value in NR3 format.`

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SERPeracq" on page 1116

History New in version 5.60.

:MEASure:SERPeracq

Command

```
:MEASure:SERPeracq <source>
```

When a pattern length and pattern can be determined (see the :ANALyze:SIGNal:PATTern:* commands), the :MEASure:SERPeracq command installs a symbol error rate (SER) per acquisition measurement of the specified PAM waveform into the user interface's measurement Results pane.

With PAM signals, there can be multiple bits per symbol, and symbol error rate is related to bit error rate (BER). For example, with PAM-4 one symbol error can translate to one or two bit errors.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:SERPeracq? <source>
```

The :MEASure:SERPeracq? query returns the measured symbol error rate per acquisition value.

Returned Format

```
[:MEASure:SERPeracq] <value><NL>

<value> ::= the SER per acquisition value in NR3 format.
```

See Also

- ":ANALyze:SIGNal:PATTern:CLEar" on page 395
- ":ANALyze:SIGNal:PATTern:LOAD" on page 397
- ":ANALyze:SIGNal:PATTern:PLENgth" on page 398
- ":ANALyze:SIGNal:PATTern:SMAP" on page 401
- ":MEASure:BER" on page 881
- ":MEASure:BERPeracq" on page 882
- ":MEASure:SER" on page 1115

History New in version 5.60.

:MEASure:SETuptime

Command

```
:MEASure:SETuptime [<data_source>,<data_source_dir>,
   <clock_source>,<clock_source_dir>]
```

The :MEASure:SETuptime command measures the setup time between the specified clock and data source.

This measurement requires all edges. When you add it, the "Measure All Edges" mode (see ":ANALyze:AEDGes" on page 332) is automatically set to ON. When the "Measure All Edges" mode is set to OFF, this measurement cannot be made, and there are no measurement results.

<data_source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | INPut | CORRected | ERRor | LFPR}

<clock_source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | INPut | CORRected | ERRor | LFPR}

The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.

The CLOCk source is available when the recovered clock is displayed.

The EQUalized<L> source is available when the Advanced Signal Integrity Software license is installed and the equalized waveform is displayed as a function.

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<R> An integer, 1-4.

<data_source_dir> {RISing | FALLing | BOTH}

Selects the direction of the data source edge. BOTH selects both edges to be measured.

<clock_source_dir> {RISing | FALLing}

Selects the direction of the clock source edge.

Example This example measures the setup time from the rising edge of channel 1 to the rising edge of channel 2.

```
myScope.WriteString ":MEASure:SETuptime CHAN1,RIS,CHAN2,RIS"
```

Query

```
:MEASure:SETuptime? [<data_source>,<data_source_dir>,
   <clock_source>,<clock_source_dir>]
```

The :MEASure:SETuptime query returns the measured setup time between the specified clock and data source.

The necessary waveform edges must be present on the display. Also, the "Measure All Edges" mode must be set (use the :ANALyze:AEDGes command or :MEASure:SETuptime command before the query).

The query will return 9.99999E+37 if the necessary edges are not displayed or if the "Measure All Edges" mode is not currently set.

Returned Format `{:MEASure:SETuptime] <value><NL>`

<value> Setup time in seconds.

Example This example places the current value of setup time in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:SETuptime? CHAN1,RIS,CHAN2,RIS"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

See Also
- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:SLEVel

Command

```
:MEASure:SLEVel [<source>[, <level>]]
```

The :MEASure:SLEVel command installs a Level Mean measurement on the SNDR Input PAM-4 waveform. For the specified PAM-4 level, this measurement returns the mean level of long-run-lengths within the IEEE or PCIE pattern. These are the same long-run-lengths used in the Sigma-n measurement (see :MEASure:NSIGma).

<source> INPut — This is the SNDR Input waveform (see :ANALyze:SNDR:INPut).

<level> An integer, 0-3 for the PAM-4 levels.

Query

```
:MEASure:SLEVel? [<source>[, <level>]]
```

The :MEASure:SLEVel? query returns the value of a Level Mean measurement on the SNDR Input PAM-4 waveform.

Returned Format

```
<value><NL>

<value> ::= a real number in NR3 format
```

See Also

- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789
- ":MEASure:NSIGma" on page 1004
- ":MEASure:SNDR" on page 1122

History New in version 11.30.

:MEASure:SLEWrate

Command

NOTE

This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:SLEWrate [<source>[,<direction>]]
```

The :MEASure:SLEWrate command measures the slew rate of the specified data source.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.

The CLOCk source is available when the recovered clock is displayed.

The EQUalized<L> source is available when the Advanced Signal Integrity Software license is installed and the equalized waveform is displayed as a function.

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<R> An integer, 1-4.

<direction> {RISing | FALLing | BOTH}

Specifies whether slew rate is measured on rising, falling, or either rising or falling edge(s). When <direction> is specified, the <source> parameter is required.

The BOTH option is valid only when the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332). In this case, the first edge from the left side of the display grid is used (whether the edge is rising or falling).

When the "Measure All Edges" mode is OFF, the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether slew rate is measured on rising or falling edges throughout the acquisition.

Example This example measures the slew rate of channel 1.

```
myScope.WriteString ":MEASure:SLEWrate CHANnel1,RISing"
```

Query

```
:MEASure:SLEWrate? [<source>[,<direction>]]
```

The :MEASure:SLEWrate? query returns the measured slew rate for the specified source.

Returned Format `{:MEASure:SLEWrate] <value><NL>`

<value> Slew rate in volts per second.

Example This example places the channel 1 value of slew rate in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:SLEWrate? CHANnel1,RISing"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:SNDR

Command

```
:MEASure:SNDR [<source>]
```

The :MEASure:SNDR command installs a Signal to Noise and Distortion Ratio (SNDR) measurement on a PAM-4 waveform.

Signal to Noise and Distortion Ratio (SNDR) is a measurement that shows how a waveform compares to its ideal. The SNDR measurement is arrived at by performing linear fit pulse response and linear fit error (matrix) math functions on the input waveform (see :ANALyze:SNDR:SOURce) and then by making calculations based on the results. Some technology standard documents publish SNDR measurement specifications.

Signal to Noise and Distortion Ratio (SNDR) is defined by the equation:

$$\text{SNDR (in dB)} = 10 \times \log_{10}(P_{max}^2 / (\sigma_e^2 + \sigma_n^2))$$

Where:

- P_{max} is the maximum value of p(k), which is the linear fit pulse response.
- σ_e is Sigma-e, the standard deviation of e(k), which is the linear fit error. The linear fit error is the difference between the input signal and the ideal signal; it represents essentially the distortion from a linear response.
- σ_n is Sigma-n, an averaged measurement of the RMS deviation from the mean voltage noise (σ) at all four PAM-4 levels. σ_n represents essentially the crosstalk and all other external noise.

You can a view the math waveforms created during the SNDR measurement process:

- The SNDR Input waveform (with the linear fit averaging and pts-per-UI parameters applied, see :ANALyze:SNDR:INPut, :ANALyze:SNDR:INPut:NAVerages, and :ANALyze:SNDR:INPut:PPUI).
- The linear fit Pulse Corrected waveform (see :ANALyze:SNDR:CORRected).
- The Error waveform, e(k) (see :ANALyze:SNDR:ERRor)
- The Linear Fit Pulse Response, p(k) (see :ANALyze:SNDR:LFPR)

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R> | FUNCtion<F> | EQUalized<L>}

Query

```
:MEASure:SNDR? [<source>]
```

The :MEASure:SNDR? query returns the value of a Signal to Noise and Distortion Ratio (SNDR) measurement on a PAM-4 waveform.

Returned Format

```
[:MEASure:SNDR] <value>[,<result_state>]<NL>
```

<value> A dB value real number in NR3 format.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":ANALyze:SNDR:CORRected" on page 357
- ":ANALyze:SNDR:ERRor" on page 364
- ":ANALyze:SNDR:LFPR" on page 368
- ":ANALyze:SNDR:INPut" on page 375
- ":ANALyze:SNDR:INPut:NAVerages" on page 379
- ":ANALyze:SNDR:INPut:PPUI" on page 380
- ":ANALyze:SNDR:LFPDelay" on page 384
- ":ANALyze:SNDR:LFPLength" on page 385
- ":ANALyze:SNDR:SOURce" on page 386
- ":LANE<N>:EQUalizer:DFE:TAP:LFPDelay" on page 788
- ":LANE<N>:EQUalizer:DFE:TAP:LFPLength" on page 789
- ":MEASure:NSIGma" on page 1004
- ":MEASure:SLEVel" on page 1119

History New in version 11.30.

:MEASure:SOURce

Command

```
:MEASure:SOURce {<source>[,<source>]}
```

The :MEASure:SOURce command selects the source for measurements. You can specify one or two sources with this command. All measurements except :MEASure:HOLDtime, :MEASure:SETUPtime, and :MEASure:DELTatime are made on the first specified source. The delta time measurement uses two sources if two are specified.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example selects channel 1 as the source for measurements.

```
myScope.WriteString ":MEASure:SOURce CHANnel1"
```

Query

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selection.

Returned Format

```
[:MEASure:SOURce] <source>[,<source>]<NL>
```

Example This example places the currently specified sources in the string variable, strSource, then prints the contents of the variable to the computer's screen.

```
Dim strSource As String   ' Dimension variable.
myScope.WriteString ":MEASure:SOURce?"
strSource = myScope.ReadString
Debug.Print strSource
```

History Legacy command (existed before version 3.10).

:MEASure:STATistics

Command

```
:MEASure:STATistics {{ON | 1} | CURRent | MAXimum | MEAN | MINimum
    | STDDev | COUNt}
```

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Example This example turns all the statistics function on.

```
myScope.WriteString ":MEASure:STATistics ON"
```

Query

```
:MEASure:STATistics?
```

The :MEASure:STATistics? query returns the current statistics mode.

Returned Format

```
[:MEASure:STATistics] {ON | CURR | MAX | MEAN | MIN | STDD | COUN}<NL>
```

Example This example places the current mode for statistics in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASure:STATistics?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also

- ":MEASure:RESults?" on page 1081 for information on the result returned and how it is affected by the STATistics command.
- ":MEASure:SENDvalid" on page 1114

History Legacy command (existed before version 3.10).

Version 10.10: The COUNt option has been added to allow the :MEASure:RESults? query to return the measurement count value.

:MEASure:TEDGe

Command `:MEASure:TEDGe <meas_thres_txt>,[<slope>]<occurrence>[,<source>]`

The :MEASure:TEDGe command measures the time interval between the trigger event and the specified edge (threshold level, slope, and transition). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TEDGe command.

<meas_thres _txt> UPPer, MIDDle, or LOWer to identify the threshold.

<slope> { - (minus) for falling | + (plus) for rising | <none> (the slope is optional; if no slope is specified, + (plus) is assumed) }

<occurrence> An integer value representing the edge of the occurrence. The desired edge must be present on the display. Edges are counted with 1 being the first edge from the left on the display, and a maximum value of 65534.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query `:MEASure:TEDGe? <meas_thres_txt>,[<slope>]<occurrence> [,<source>]`

The :MEASure:TEDGe? query returns the time interval between the trigger event and the specified edge (threshold level, slope, and transition).

Returned Format `[:MEASure:TEDGe] <time>[,<result_state>]<NL>`

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time interval between the trigger event and the 90% threshold on the second rising edge of the source waveform to the numeric variable, varTime. The contents of the variable are then printed to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TEDGe? UPPer,+2,CHANnel1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:THResholds:ABSolute

Command

```
:MEASure:THResholds:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<R> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_volts> <middle_volts> <lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:ABSolute CHANnel2,0.9,0.5,0.1"
```

Query

```
:MEASure:THResholds:ABSolute? <source>
```

The :MEASure:THResholds:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:ABSolute] <upper_volts>,<middle_volts>,<lower_volts
><NL>
```

Example This example returns the upper level, middle level, and lower level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:THResholds:DISPlay

Command

```
:MEASure:THResholds:DISPlay <source>,{{0 | OFF} | {1 | ON}}
```

When the source is a PAM-4 signal type (see :ANALyze:SIGNal:TYPE), the :MEASure:THResholds:DISPlay command turns on or off the display of the PAM measurement thresholds. This is the remote command equivalent of the graphical user interface's **Display Thresholds** check box in the Signal Type Setup dialog box.

<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | EQUalized<L>}

Query

```
:MEASure:THResholds:DISPlay? <source>
```

The :MEASure:THResholds:DISPlay? query returns thresholds display setting.

Returned Format

```
<setting><NL>

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

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

History New in version 6.10.

:MEASure:THResholds:GENAUTO

Command `:MEASure:THResholds:GENAUTO <source>`

The :MEASure:THResholds:GENAUTO command automatically sets the general "Custom: thresholds +/- hysteresis" when thresholds apply to individual waveforms. This command is the same as pressing the **Auto set thresholds** button in the graphical user interface.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

See Also

- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:HYSTeresis" on page 1133

History New in version 6.00.

:MEASure:THResholds:GENeral:ABSolute

Command

```
:MEASure:THResholds:GENeral:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:GENeral:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<R> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_volts> <middle_volts> <lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:ABSolute CHANnel2,0.9,0
.5,0.1"
```

Query

```
:MEASure:THResholds:GENeral:ABSolute? <source>
```

The :MEASure:THResholds:GENeral:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:GENeral:ABSolute] <upper_volts>,<middle_volts>,<low
er_volts><NL>
```

Example This example returns the upper level, middle level, and lower level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE **Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 1128
- ":MEASure:THResholds:RFALl:ABSolute" on page 1150
- ":MEASure:THResholds:SERial:ABSolute" on page 1162

History New in version 3.10.

:MEASure:THResholds:GENeral:HYSTeresis

Command

```
:MEASure:THResholds:GENeral:HYSTeresis <source>,<range>,<level>
```

The :MEASure:THResholds:GENeral:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE **These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.**

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:HYSTeresis CHANnel2,0.9
,0.1"
```

Query

```
:MEASure:THResholds:GENeral:HYSTeresis? <source>
```

The :MEASure:THResholds:GENeral:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:GENeral:HYSTeresis]<range>,<level><NL>
```

Example This example returns the range and level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:HYSTeresis? CHANnel1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:GENAUTO" on page 1130
- ":MEASure:THResholds:HYSTeresis" on page 1146
- ":MEASure:THResholds:SERial:HYSTeresis" on page 1164

History New in version 3.10.

:MEASure:THResholds:GENeral:METHod

Command

```
:MEASure:THResholds:GENeral:METHod <source>,{ABSolute | PERCent
    | HYSTeresis | PAMCustom | PAMAutomatic}
```

The :MEASure:THResholds:GENeral:METHod command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

NOTE **These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.**

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), you can choose between these methods for setting the general measurement thresholds:

- PAMCustom — Then, use the :MEASure:THResholds:GENeral:PAMCustom command to set the PAM threshold levels for general measurements to the values you specify.
- PAMAutomatic — Then, use the :MEASure:THResholds:GENeral:PAMAutomatic command to specify whether the PAM threshold levels for general measurements are determined automatically or using the PAM-4 levels you specify.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:METHod CHANnel1,HYSTere
sis"
```

Query

```
:MEASure:THResholds:GENeral:METHod? <source>
```

The :MEASure:THResholds:GENeral:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:GENeral:METHod <source>,] {ABS | PERC | HYST
    | PAMC | PAMA}
```

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:METHod?"
```

```
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 1148
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:SERial:METHod" on page 1166
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139

History New in version 3.10.

Version 5.50: When the signal type is PAM-4, you can choose between PAMCustom and PAMautomatic methods for setting the general measurement thresholds.

:MEASure:THResholds:GENeral:PAMCustom

Command

```
:MEASure:THResholds:GENeral:PAMCustom
   <source>,<NN_threshold>,...[,<hysteresis>]
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE) and :MEASure:THResholds:GENeral:METHod is set to PAMCustom, the :MEASure:THResholds:GENeral:PAMCustom command sets the PAM threshold levels for general measurements to the values you specify.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<NN_threshold>,... Values for the PAM thresholds in NR3 format:

- For PAM-3: <01_threshold>, <12_threshold>
- For PAM-4: <01_threshold>, <12_threshold>, <23_threshold>
- For PAM-6: <01_threshold>, <12_threshold>, <23_threshold>, <34_threshold>, <45_threshold>
- For PAM-8: <01_threshold>, <12_threshold>, <23_threshold>, <34_threshold>, <45_threshold>, <56_threshold>, <67_threshold>

<hysteresis> Threshold hysteresis value in NR3 format.

Query

```
:MEASure:THResholds:GENeral:PAMCustom? <source>
```

The :MEASure:THResholds:GENeral:PAMCustom? query returns the currently set PAM custom threshold levels for general measurements.

Returned Format

```
[:MEASure:THResholds:GENeral:PAMCustom]
  <source>,<NN_threshold>,...,<hysteresis><NL>
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085

- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 6.10: Added an optional <hysteresis> value at the end of the command parameters.

Version 10.10: Modified to work with the PAM-3 signal type.

Version 11.20: Threshold levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:THResholds:GENeral:PAMAutomatic

Command

```
:MEASure:THResholds:GENeral:PAMAutomatic
   <source>,{AUTomatic | <N_level>,...}
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE) and :MEASure:THResholds:GENeral:METHod is set to PAMAutomatic, the :MEASure:THResholds:GENeral:PAMAutomatic command specifies whether the PAM threshold levels for general measurements are determined automatically or using the PAM levels you specify.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<N_level>,... Voltage values for the PAM levels in NR3 format:

- For PAM-3: <0_level>, <1_level>, <2_level>
- For PAM-4: <0_level>, <1_level>, <2_level>, <3_level>
- For PAM-6: <0_level>, <1_level>, <2_level>, <3_level>, <4_level>, <5_level>
- For PAM-8: <0_level>, <1_level>, <2_level>, <3_level>, <4_level>, <5_level>, <6_level>, <7_level>

Query

```
:MEASure:THResholds:GENeral:PAMAutomatic? <source>
```

The :MEASure:THResholds:GENeral:PAMAutomatic? query returns the values used for automatically setting the PAM general threshold levels.

Returned Format

```
[:MEASure:THResholds:GENeral:PAMAutomatic]
   <source>,{AUTomatic | <N_level>,...}<NL>
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129

- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:THResholds:GENeral:PERCent

Command

```
:MEASure:THResholds:GENeral:PERCent <source>,<upper_pct>,<middle_pct>,<l
ower_pct>
```

The :MEASure:THResholds:GENeral:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANNel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_pct> <middle_pct> <lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:PERCent CHANnel2,100,50
,0"
```

Query

```
:MEASure:THResholds:GENeral:PERCent? <source>
```

The :MEASure:THResholds:GENeral:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:GENeral:PERCent] <upper_pct>,<middle_pcts>,<lower_p
ct><NL>
```

Example This example returns the upper level, middle level, and lower level percentages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 1149
- ":MEASure:THResholds:RFALl:PERCent" on page 1156
- ":MEASure:THResholds:SERial:PERCent" on page 1167

History New in version 3.10.

:MEASure:THResholds:GENeral:TOPBase:ABSolute

Command

```
:MEASure:THResholds:GENeral:TOPBase:ABSolute <source>,<top_volts>,<base_volts>
```

The :MEASure:THResholds:GENeral:TOPBase:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

NOTE

These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANNel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<top_volts> <base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:ABSolute CHANnel2,0.9,0.1"
```

Query

```
:MEASure:THResholds:GENeral:TOPBase:ABSolute? <source>
```

The :MEASure:THResholds:GENeral:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:GENeral:TOPBase:ABSolute] <top_volts>,<base_volts><NL>
```

Example This example returns the top level and base level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:ABSolute? CHANnel1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 1172
- ":MEASure:THResholds:RFALl:TOPBase:ABSolute" on page 1158
- ":MEASure:THResholds:SERial:TOPBase:ABSolute" on page 1169

History New in version 3.10.

:MEASure:THResholds:GENeral:TOPBase:METHod

Command

```
:MEASure:THResholds:GENeral:TOPBase:METHod <source>,{ABSolute | HISTONLY
   | MINmax | STANdard}
```

The :MEASure:THResholds:GENeral:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:METHod CHANnel1
,HISTONLY"
```

Query

```
:MEASure:THResholds:GENeral:TOPBase:METHod? <source>
```

The :MEASure:THResholds:GENeral:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:GENeral:TOPBase:METHod] {ABSolute | HISTONLY | MINm
ax | STANdard}
```

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:METHod CHANnel1
"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 1173
- ":MEASure:THResholds:RFALl:TOPBase:METHod" on page 1160
- ":MEASure:THResholds:SERial:TOPBase:METHod" on page 1171

History New in version 3.10.

:MEASure:THResholds:HYSTeresis

Command

```
:MEASure:THResholds:HYSTeresis <source>,<range>,<level>
```

The :MEASure:THResholds:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE This command does not affect Rise/Fall measurement thresholds.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:HYSTeresis CHANnel2,0.9,0.1"
```

Query

```
:MEASure:THResholds:HYSTeresis? <source>
```

The :MEASure:THResholds:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:HYSTeresis]<range>,<level><NL>
```

Example This example returns the range and level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:HYSTeresis? CHANnel1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:THResholds:METHod

Command

```
:MEASure:THResholds:METHod <source>,{ABSolute | PERCent | HYSTeresis}
```

The :MEASure:THResholds:METHod command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

NOTE

This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:SERial:METHod" on page 1166

<source>

{ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example

This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:METHod CHANnel1,HYSTeresis"
```

Query

```
:MEASure:THResholds:METHod? <source>
```

The :MEASure:THResholds:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:METHod <source>,] {ABSolute | PERCent | HYSTeresis}
```

Example

This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

History

Legacy command (existed before version 3.10).

:MEASure:THResholds:PERCent

Command

```
:MEASure:THResholds:PERCent <source>,<upper_pct>,<middle_pct>,<lower_pct
>
```

The :MEASure:THResholds:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

<source> {ALL | CHANNel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_pct> <middle_pct> <lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:PERCent CHANnel2,100,50,0"
```

Query

```
:MEASure:THResholds:PERCent? <source>
```

The :MEASure:THResholds:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:PERCent] <upper_pct>,<middle_pcts>,<lower_pct><NL>
```

Example This example returns the upper level, middle level, and lower level percentages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:THResholds:RFALl:ABSolute

Command

```
:MEASure:THResholds:RFALl:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:RFALl:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

NOTE

These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<R> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_volts> <middle_volts> <lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:ABSolute CHANnel2,0.9,0.5
,0.1"
```

Query

```
:MEASure:THResholds:RFALl:ABSolute? <source>
```

The :MEASure:THResholds:RFALl:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:RFALl:ABSolute] <upper_volts>,<middle_volts>,<lower
_volts><NL>
```

Example This example returns the upper level, middle level, and lower level voltages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 1128
- ":MEASure:THResholds:GENeral:ABSolute" on page 1131
- ":MEASure:THResholds:SERial:ABSolute" on page 1162

History New in version 3.10.

:MEASure:THResholds:RFALl:METHod

Command

```
:MEASure:THResholds:RFALl:METHod <source>,{ABSolute | PERCent | T1090
    | T2080}
```

The :MEASure:THResholds:RFALl:METHod command determines the way that the top and base of a waveform are calculated for rise/fall measurements:

- ABSolute — Absolute thresholds specified by the :MEASure:THResholds:RFALl:ABSolute command are used.
- PERCent — Percent thresholds specified by the :MEASure:THResholds:RFALl:PERCent command are used.
- T1090, T2080 — Lets you choose T1090 (10% and 90% of levels) or T2080 (20% and 80% of levels) when setting the rise/fall measurement thresholds.

  While the T1090 and T2080 selections are available for UNSPecified and NRZ signal types (see :ANALyze:SIGNal:TYPE), they are the only selections available for PAM signal types. With PAM signal types, use the :MEASure:THResholds:RFALl:PAMAutomatic command to specify whether the PAM threshold levels for rise/fall measurements are determined automatically or using PAM-4 levels you specify.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:METHod CHANnel1,HYSTeresi
s"
```

Query

```
:MEASure:THResholds:RFALl:METHod? <source>
```

The :MEASure:THResholds:RFALl:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:RFALl:METHod <source>,] {ABS | PERC | T1090
    | T2080}
```

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 1148
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:SERial:METHod" on page 1166
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:THResholds:RFALl:ABSolute" on page 1150
- ":MEASure:THResholds:RFALl:PERCent" on page 1156
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

History

New in version 3.10.

Version 5.50: When the signal type is PAM-4, you can choose between T1090 (10% and 90% of levels) and T2080 (20% and 80% of levels) when setting the rise/fall measurement thresholds.

Version 11.10: The T1090 (10% and 90% of levels) and T2080 (20% and 80% of levels) settings for rise/fall measurement thresholds now work for NRZ and UNSPecified signal types as well as PAM signal types.

:MEASure:THResholds:RFALl:PAMAutomatic

Command

```
:MEASure:THResholds:RFALl:PAMAutomatic
   <source>,{AUTomatic | <N_level>,...}
```

When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the :MEASure:THResholds:RFALl:PAMAutomatic command specifies whether the PAM threshold levels for rise/fall measurements are determined automatically or using the PAM levels you specify.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | XT<X>}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<N_level>,... Voltage values for the PAM levels in NR3 format:

- For PAM-3: <0_level>, <1_level>, <2_level>
- For PAM-4: <0_level>, <1_level>, <2_level>, <3_level>
- For PAM-6: <0_level>, <1_level>, <2_level>, <3_level>, <4_level>, <5_level>
- For PAM-8: <0_level>, <1_level>, <2_level>, <3_level>, <4_level>, <5_level>, <6_level>, <7_level>

Query

```
:MEASure:THResholds:RFALl:PAMAutomatic? <source>
```

The :MEASure:THResholds:RFALl:PAMAutomatic? query returns the values used for automatically setting the PAM rise/fall threshold levels.

Returned Format

```
[:MEASure:THResholds:RFALl:PAMAutomatic]
   <source>,{AUTomatic | <N_level>,...}<NL>
```

See Also

- ":ANALyze:CLOCk:METHod:SKEW" on page 349
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:DISPlay" on page 1129
- ":MEASure:THResholds:GENeral:METHod" on page 1135

- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Version 11.20: Levels for PAM-6 and PAM-8 signal types are now supported.

:MEASure:THResholds:RFALl:PERCent

Command

```
:MEASure:THResholds:RFALl:PERCent <source>,<upper_pct>,<middle_pct>,<low
er_pct>
```

The :MEASure:THResholds:RFALl:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_pct> <middle_pct> <lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:PERCent CHANnel2,100,50,0
"
```

Query

```
:MEASure:THResholds:RFALl:PERCent? <source>
```

The :MEASure:THResholds:RFALl:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:RFALl:PERCent] <upper_pct>,<middle_pcts>,<lower_pct
><NL>
```

Example This example returns the upper level, middle level, and lower level percentages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE **Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 1149
- ":MEASure:THResholds:GENeral:PERCent" on page 1141
- ":MEASure:THResholds:SERial:PERCent" on page 1167

History New in version 3.10.

:MEASure:THResholds:RFALl:TOPBase:ABSolute

Command

```
:MEASure:THResholds:RFALl:TOPBase:ABSolute <source>,<top_volts>,<base_vo
lts>
```

The :MEASure:THResholds:RFALl:TOPBase:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

NOTE

These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<top_volts> <base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:ABSolute CHANnel2
,0.9,0.1"
```

Query

```
:MEASure:THResholds:RFALl:TOPBase:ABSolute? <source>
```

The :MEASure:THResholds:RFALl:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:RFALl:TOPBase:ABSolute] <top_volts>,<base_volts><NL
>
```

Example This example returns the top level and base level voltages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:ABSolute? CHANnel
1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 1172
- ":MEASure:THResholds:GENeral:TOPBase:ABSolute" on page 1143
- ":MEASure:THResholds:SERial:TOPBase:ABSolute" on page 1169

History New in version 3.10.

:MEASure:THResholds:RFALl:TOPBase:METHod

Command

```
:MEASure:THResholds:RFALl:TOPBase:METHod <source>,{ABSolute | HISTONLY
    | MINmax | STANdard}
```

The :MEASure:THResholds:RFALl:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:METHod CHANnel1,H
ISTONLY"
```

Query

```
:MEASure:THResholds:RFALl:TOPBase:METHod? <source>
```

The :MEASure:THResholds:RFALl:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:RFALl:TOPBase:METHod] {ABSolute | HISTONLY | MINmax
 | STANdard}
```

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 1173
- ":MEASure:THResholds:GENeral:TOPBase:METHod" on page 1145
- ":MEASure:THResholds:SERial:TOPBase:METHod" on page 1171

History New in version 3.10.

:MEASure:THResholds:SERauto

Command

```
:MEASure:THResholds:SERauto <source>
```

For protocol decodes that do not use clock recovery, the :MEASure:THResholds:SERauto command automatically sets the general "Custom: thresholds (low, mid, up)" or "Custom: thresholds +/- hysteresis" when thresholds apply to individual waveforms. This command is the same as pressing the **Auto set thresholds** button in the graphical user interface.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

See Also

- ":MEASure:THResholds:SERial:METHod" on page 1166
- ":MEASure:THResholds:SERial:HYSTeresis" on page 1164

History New in version 11.10.

:MEASure:THResholds:SERial:ABSolute

Command

```
:MEASure:THResholds:SERial:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:SERial:ABSolute command sets the upper level, middle level, and lower level voltages that are used for protocol decode.

NOTE **These serial threshold settings are used for protocol decode.**

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<R> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_volts> <middle_volts> <lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:ABSolute CHANnel2,0.9,0.
5,0.1"
```

Query

```
:MEASure:THResholds:SERial:ABSolute? <source>
```

The :MEASure:THResholds:SERial:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:SERial:ABSolute] <upper_volts>,<middle_volts>,<lowe
r_volts><NL>
```

Example This example returns the upper level, middle level, and lower level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE **Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 1128
- ":MEASure:THResholds:GENeral:ABSolute" on page 1131
- ":MEASure:THResholds:RFALl:ABSolute" on page 1150

History New in version 3.10.

:MEASure:THResholds:SERial:HYSTeresis

Command

```
:MEASure:THResholds:SERial:HYSTeresis <source>,<range>,<level>
```

The :MEASure:THResholds:SERial:HYSTeresis command sets the range and level voltages that are used for protocol decode. The range is added to the level to determine the upper level voltage. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE

These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:HYSTeresis CHANnel2,0.9,
0.1"
```

Query

```
:MEASure:THResholds:SERial:HYSTeresis? <source>
```

The :MEASure:THResholds:SERial:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:SERial:HYSTeresis]<range>,<level><NL>
```

Example This example returns the range and level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:HYSTeresis? CHANnel1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:HYSTeresis" on page 1146
- ":MEASure:THResholds:GENeral:HYSTeresis" on page 1133

History New in version 3.10.

:MEASure:THResholds:SERial:METHod

Command

```
:MEASure:THResholds:SERial:METHod <source>,{ABSolute | PERCent | HYSTere
sis}
```

The :MEASure:THResholds:SERial:METHod command determines the way that the top and base of a waveform are calculated for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:METHod CHANnel1,HYSTeres
is"
```

Query

```
:MEASure:THResholds:SERial:METHod? <source>
```

The :MEASure:THResholds:SERial:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:SERial:METHod <source>,] {ABSolute | PERCent | HYST
eresis}
```

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 1148
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:RFALl:METHod" on page 1152

History New in version 3.10.

:MEASure:THResholds:SERial:PERCent

Command

```
:MEASure:THResholds:SERial:PERCent <source>,<upper_pct>,<middle_pct>,<lo
wer_pct>
```

The :MEASure:THResholds:SERial:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used for protocol decode.

NOTE

These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<upper_pct> <middle_pct> <lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:PERCent CHANnel2,100,50,
0"
```

Query

```
:MEASure:THResholds:SERial:PERCent? <source>
```

The :MEASure:THResholds:SERial:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:SERial:PERCent] <upper_pct>,<middle_pcts>,<lower_pc
t><NL>
```

Example This example returns the upper level, middle level, and lower level percentages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 1149
- ":MEASure:THResholds:GENeral:PERCent" on page 1141
- ":MEASure:THResholds:RFALl:PERCent" on page 1156

History New in version 3.10.

:MEASure:THResholds:SERial:TOPBase:ABSolute

Command

```
:MEASure:THResholds:SERial:TOPBase:ABSolute <source>,<top_volts>,<base_v
olts>
```

The :MEASure:THResholds:SERial:TOPBase:ABSolute command sets the top level and base level voltages that are used for protocol decode.

NOTE

These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<top_volts> <base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:ABSolute CHANnel
2,0.9,0.1"
```

Query

```
:MEASure:THResholds:SERial:TOPBase:ABSolute? <source>
```

The :MEASure:THResholds:SERial:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:SERial:TOPBase:ABSolute] <top_volts>,<base_volts><N
L>
```

Example This example returns the top level and base level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:ABSolute? CHANne
l1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 1172
- ":MEASure:THResholds:GENeral:TOPBase:ABSolute" on page 1143
- ":MEASure:THResholds:RFALl:TOPBase:ABSolute" on page 1158

History New in version 3.10.

:MEASure:THResholds:SERial:TOPBase:METHod

Command

```
:MEASure:THResholds:SERial:TOPBase:METHod <source>,{ABSolute | HISTONLY
    | MINmax | STANdard}
```

The :MEASure:THResholds:SERial:TOPBase:METHod command determines the way that the top and base of a waveform are derived for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:METHod CHANnel1,
HISTONLY"
```

Query

```
:MEASure:THResholds:SERial:TOPBase:METHod? <source>
```

The :MEASure:THResholds:SERial:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:SERial:TOPBase:METHod] {ABSolute | HISTONLY | MINma
x | STANdard}
```

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 1173
- ":MEASure:THResholds:GENeral:TOPBase:METHod" on page 1145
- ":MEASure:THResholds:RFALl:TOPBase:METHod" on page 1160

History New in version 3.10.

:MEASure:THResholds:TOPBase:ABSolute

Command

```
:MEASure:THResholds:TOPBase:ABSolute <source>,<top_volts>,<base_volts>
```

The :MEASure:THResholds:TOPBase:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<top_volts> <base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:ABSolute CHANnel2,0.9,0
.1"
```

Query

```
:MEASure:THResholds:TOPBase:ABSolute? <source>
```

The :MEASure:THResholds:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:TOPBase:ABSolute] <top_volts>,<base_volts><NL>
```

Example This example returns the top level and base level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:ABSolute? CHANnel1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

History Legacy command (existed before version 3.10).

:MEASure:THResholds:TOPBase:METHod

Command

```
:MEASure:THResholds:TOPBase:METHod <source>,{ABSolute | HISTONLY
    | MINmax | STANdard}
```

The :MEASure:THResholds:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:METHod CHANnel1,HISTONL
Y"
```

Query

```
:MEASure:THResholds:TOPBase:METHod? <source>
```

The :MEASure:THResholds:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:TOPBase:METHod] {ABSolute | HISTONLY | MINmax | STA
Ndard}
```

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:TIEClock2

Command

NOTE **Turn Off Headers**

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:TIEClock2 <source>,{SECond | UNITinterval},
   <direction>,{AUTO | CUSTOM,<frequency>
   | {VARiable,<frequency>,<bandwidth>} | CLOCk}
```

The :MEASure:TIEClock2 command measures time interval error on a clock. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval. If AUTO is selected, the oscilloscope selects the ideal constant clock frequency. If CUSTOM is selected, you can enter your own ideal clock frequency. If VARiable is selected, a first order PLL clock recovery is used at the give clock frequency and loop bandwidth. If CLOCk is given, clock recovery is specified with the :MEASure:CLOCk:METHod command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing | BOTH}

Specifies direction of clock edge. BOTH selects the first edge from the left-hand side of the waveform viewing area.

<frequency> A real number for the ideal clock frequency for clock recovery.

<bandwidth> A real number for the loop bandwidth of the PLL clock recovery method.

Example This example measures the clock time interval error on the rising edge of channel 1, ideal clock frequency set to automatic, units set to seconds.

```
myScope.WriteString ":MEASure:TIEClock2 CHANnel1,SECond,RISing,AUTO"
```

Query

```
:MEASure:TIEClock2? <source>,{SECond | UNITinterval},
   <direction>,{AUTO | CUSTOM,<frequency>
   | {VARiable,<frequency>,<bandwidth>} | CLOCk}
```

The :MEASure:TIEClock2? query returns the current value of the clock time interval error.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:TIEClock2] <value>[,<result_state>]<NL>
```

<value> The clock time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of the clock time interval error in the variable strValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:TIEClock2? CHANnel1,SECond,FALLing,CUSTOM,
2.5E9"
strValue = myScope.ReadString
Debug.Print strValue
```

See Also

- ":ANALyze:AEDGes" on page 332
- ":MEASure:CTCJitter" on page 907
- ":MEASure:NCJitter" on page 987
- ":MEASure:CTCPwidth" on page 911
- ":MEASure:CTCNwidth" on page 909
- ":MEASure:CTCDutycycle" on page 905

History Legacy command (existed before version 3.10).

:MEASure:TIEData2

Command

```
:MEASure:TIEData2 <source>,{SECond | UNITinterval}[,<threshold>]
```

The :MEASure:TIEData2 command measures data time interval error. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<threshold> When the signal type is one of the PAM types (see :ANALyze:SIGNal:TYPE), the <threshold> parameter is an integer that specifies which PAM threshold to measure.

- For PAM-3, the <threshold> may be from 0-1.
- For PAM-4, the <threshold> may be from 0-2.
- For PAM-6, the <threshold> may be from 0-4.
- For PAM-8, the <threshold> may be from 0-6.

Example This example measures the data time interval error on channel 1, ideal data rate set to automatic, units set to seconds.

```
myScope.WriteString ":MEASure:TIEData2 CHANnel1,SECond"
```

Query

```
:MEASure:TIEData2? <source>,{SECond | UNITinterval}[,<threshold>]
```

The :MEASure:TIEData2? query returns the current value of the data time interval error.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:TIEData2] <value>[,<result_state>]<NL>
```

<value> The data time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of the data time interval error in the variable strValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:TIEData2? CHANnel1,SECond"
```

```
strValue = myScope.ReadString
Debug.Print strValue
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":ANALyze:AEDGes" on page 332

History New in version 5.50. This command replaces the now deprecated command ":MEASure:TIEData" on page 1886.

Version 11.20: The <threshold> parameter now supports PAM-6 and PAM-8 eyes.

:MEASure:TIEFilter:DAMPing

Command

```
:MEASure:TIEFilter:DAMPing <damping_factor>
```

The :MEASure:TIEFilter:DAMPing command specifies the damping factory for a second order low-pass TIE filter.

Use the :MEASure:TIEFilter:TYPE command to set the low-pass filter type.

Use the :MEASure:TIEFilter:SHAPe command to set the Second Order TIE filter shape.

<damping_factor> The damping factor in NR3 format.

Query

```
:MEASure:TIEFilter:DAMPing?
```

The :MEASure:TIEFilter:DAMPing? query returns the damping factor setting.

Returned Format

```
<damping_factor><NL>
```

See Also

- ":MEASure:TIEFilter:TYPE" on page 1184
- ":MEASure:TIEFilter:SHAPe" on page 1179

History New in version 10.25.

:MEASure:TIEFilter:SHAPe

Command

```
:MEASure:TIEFilter:SHAPe {RECTangular | DB20 | DB40 | DB60 | FIRSt
    | SECond}
```

The :MEASure:TIEFilter:SHAPe command specifies the shape of the TIE filter edge(s):

- RECTangular — The TIE filter is a brickwall filter.
- DB20 — The TIE filter edge(s) roll off at 20 dB per decade.
- DB40 — The TIE filter edge(s) roll off at 40 dB per decade.
- DB60 — The TIE filter edge(s) roll off at 40 dB per decade.
- FIRSt — First Order TIE filter. This is similar to the 20 dB per decade roll off, but the response is more curved.
- SECond — Second Order TIE filter. This is similar to the 40 dB per decade roll off, but the response is more curved.

When MEASure:TIEFilter:TYPE is BANDpass, the valid shapes are RECTangular, DB20, DB40, or DB60.

Example This example specifies that the TIE filter edge(s) roll off at 40 dB per decade.

```
myScope.WriteString ":MEASure:TIEFilter:SHAPe DB40"
```

Query

```
:MEASure:TIEFilter:SHAPe?
```

The :MEASure:TIEFilter:SHAPe? query returns the specified shape of the TIE filter edge(s).

Returned Format

```
[:MEASure:TIEFilter:SHAPe] {RECTangular | DB20 | DB40 | DB60 | FIRS
    | SEC}<NL>
```

Example This example places the specified shape of the TIE filter edge(s) in the string variable, strShape, then prints the contents of the variable to the computer's screen.

```
Dim strShape As String   ' Dimension variable.
myScope.WriteString ":MEASure:TIEFilter:SHAPe?"
strShape = myScope.ReadString
Debug.Print strShape
```

See Also

- ":MEASure:TIEFilter:TYPE" on page 1184
- ":MEASure:TIEFilter:DAMPing" on page 1178
- ":MEASure:TIEFilter:STATe" on page 1182
- ":MEASure:TIEFilter:STARt" on page 1181
- ":MEASure:TIEFilter:STOP" on page 1183

History New in version 4.10.

Version 10.25: Added the FIRSt and SECond options for the new First Order and Second Order TIE filter shapes.

Version 11.10: Added the DB60 option for the new 60dB/Decade TIE filter shape.

:MEASure:TIEFilter:STARt

Command `:MEASure:TIEFilter:STARt <start_frequency>`

The :MEASure:TIEFilter:STARt command sets the starting frequency for the TIE filter.

<start_ frequency> A real number.

Query `:MEASure:TIEFilter:STARt?`

The :MEASure:TIEFilter:STARt? query returns the current value of the starting frequency of the TIE filter.

Returned Format `[:MEASure:TIEFilter:STARt] <value><NL>`

<value> The start frequency for the TIE filter.

Example This example returns the current value of the starting frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STARt?"
varStart = myScope.ReadNumber
Debug.Print FormatNumber(varStart, 0)
```

See Also

- ":MEASure:TIEFilter:STATe" on page 1182
- ":MEASure:TIEFilter:TYPE" on page 1184
- ":MEASure:TIEFilter:SHAPe" on page 1179
- ":MEASure:TIEFilter:STOP" on page 1183

History Legacy command (existed before version 3.10).

:MEASure:TIEFilter:STATe

Command

```
:MEASure:TIEFilter:STATe {{ON | 1} | {OFF | 0}}
```

The :MEASure:TIEFilter:STATe command enables the TIE filter for TIE data measurements.

Query

```
:MEASure:TIEFilter:STATe?
```

The :MEASure:TIEFilter:STATe? query returns the current state of the TIE data filter.

Returned Format

```
[:MEASure:TIEFilter:STATe] {0 | 1}<NL>
```

Example

This example returns the current state of the TIE data filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STATe?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

See Also

- ":MEASure:TIEFilter:TYPE" on page 1184
- ":MEASure:TIEFilter:SHAPe" on page 1179
- ":MEASure:TIEFilter:STARt" on page 1181
- ":MEASure:TIEFilter:STOP" on page 1183

History

Legacy command (existed before version 3.10).

:MEASure:TIEFilter:STOP

Command `:MEASure:TIEFilter:STOP <stop_frequency>`

The :MEASure:TIEFilter:STOP command sets the stopping frequency for the TIE filter.

<stop_ frequency> A real number.

Query `:MEASure:TIEFilter:STOP?`

The :MEASure:TIEFilter:STOP? query returns the current value of the stopping frequency of the TIE filter.

Returned Format [:MEASure:TIEFilter:STOP] <value><NL>

<value> The stop frequency for the TIE filter.

Example This example returns the current value of the stopping frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STOP?"
varStop = myScope.ReadNumber
Debug.Print FormatNumber(varStop, 0)
```

See Also

- ":MEASure:TIEFilter:STATe" on page 1182
- ":MEASure:TIEFilter:TYPE" on page 1184
- ":MEASure:TIEFilter:SHAPe" on page 1179
- ":MEASure:TIEFilter:STARt" on page 1181

History Legacy command (existed before version 3.10).

:MEASure:TIEFilter:TYPE

Command

```
:MEASure:TIEFilter:TYPE {BANDpass | LOWPass | HIGHpass}
```

The :MEASure:TIEFilter:TYPE command sets the type of TIE filter to be used.

Example This example sets the TIE filter to highpass.

```
myScope.WriteString ":MEASure:TIEFilter:TYPE HIGHpass"
```

Query

```
:MEASure:TIEFilter:TYPE?
```

The :MEASure:TIEFilter:TYPE? query returns the current type of TIE filter being used.

Returned Format

```
[:MEASure:TIEFilter:TYPE] {BANDpass | LOWPass | HIGHpass}<NL>
```

Example This example places the current mode for TIEFilter:TYPE in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASure:TIEFilter:TYPE?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also

- ":MEASure:TIEFilter:STATe" on page 1182
- ":MEASure:TIEFilter:SHAPe" on page 1179
- ":MEASure:TIEFilter:STARt" on page 1181
- ":MEASure:TIEFilter:STOP" on page 1183

History Legacy command (existed before version 3.10).

:MEASure:TMAX

Command

```
:MEASure:TMAX [<source>]
```

The :MEASure:TMAX command measures the first time at which the maximum voltage of the source waveform occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMAX command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:TMAX? [<source>]
```

The :MEASure:TMAX? query returns the time at which the first maximum voltage occurred.

Returned Format

```
[:MEASure:TMAX] <time>[,<result_state>]<NL>
```

<time> Time at which the first maximum voltage occurred or frequency where the maximum FFT amplitude occurred.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first maximum voltage occurred to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TMAX? CHANnel1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:TMIN

Command

```
:MEASure:TMIN [<source>]
```

The :MEASure:TMIN command measures the time at which the first minimum voltage occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMIN command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:TMIN? [<source>]
```

The :MEASure:TMIN? query returns the time at which the first minimum voltage occurred or the frequency where the minimum FFT amplitude occurred.

Returned Format

```
[:MEASure:TMIN] <time>[,<result_state>]<NL>
```

<time> Time at which the first minimum voltage occurred.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first minimum voltage occurred to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:TMIN? CHANnel1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:TVOLt

Command

```
:MEASure:TVOLt <voltage>,[<slope>]<occurrence> [,<source>]
```

The :MEASure:TVOLt command measures the time interval between the trigger event and the defined voltage level and transition. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TVOLt command.

The TEDGe command can be used to get the time of edges.

When the "Measure All Edges" mode is on (see ":ANALyze:AEDGes" on page 332), the first edge from the beginning of the acquisition is used.

When the "Measure All Edges" mode is off, the first edge from the left side of the display grid is used.

<voltage> Voltage level at which time will be measured.

<slope> The direction of the waveform change when the specified voltage is crossed - rising (+) or falling (-). If no +/- sign is present, + is assumed.

<occurrence> The number of the crossing to be reported (if one, the first crossing is reported; if two, the second crossing is reported, etc.). The desired crossing must be present on the display. Occurrences are counted with 1 being the first occurrence from the left of the display, and a maximum value of 65534.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:TVOLt? <voltage>,[<slope>]<occurrence> [,<source>]
```

The :MEASure:TVOLt? query returns the time interval between the trigger event and the specified voltage level and transition.

Returned Format

```
[:MEASure:TVOLt] <time>[,<result_state>]<NL>
```

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time interval between the trigger event and the transition through -0.250 Volts on the third rising occurrence of the source waveform to the numeric variable, varTime. The contents of the variable are then printed to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:TVOLt? -0.250,+3,CHANnel1"
```

```
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:UITouijitter

Command

```
:MEASure:UITouijitter <source>, <N>
```

The :MEASure:UITouijitter command measures the difference between two consecutive N-UI measurements. The measurement then moves over one unit interval and makes another measurement. When N=1, this is analogous to cycle-cycle jitter, but measures unit intervals instead of periods. When N>1, this is analogous to N-Cycle jitter but measures unit intervals instead of periods.

<source> The source on which the measurement is made.

For more information on <source> parameters, see "Measurement Sources" on page 877.

<N> An integer greater than or equal to 1.

Example This example measures the UI-UI jitter for 3 consecutive unit intervals on channel 1.

```
myScope.WriteString ":MEASure:UITouijitter CHAN1, 3"
```

Query

```
:MEASure:UITouijitter?
```

The :MEASure:UITouijitter? query returns the measured UI-UI jitter.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:UNDershoot

Command

```
:MEASure:UNDershoot [<source>[,<direction>]]
```

The :MEASure:UNDershoot command installs an Undershoot measurement. This measurement is visible in the front-panel user interface Measurements window in the Results pane.

Undershoot is a waveform aberration that is less than (<) the Top voltage within a positive pulse region or greater than (>) the Base voltage within a negative pulse region.

NOTE If more than one such waveform aberration exists, the one with the largest magnitude is the undershoot unless otherwise specified.

NOTE Undershoot, unlike Overshoot and Preshoot is measurement in respect of pulse polarity and not rising or falling edge only. Consequently, a minimum of two (2) edges must be in the viewing window for the measurement to occur.

NOTE By definition, undershoot is measured only after the waveform reaches a level greater than or equal to (≥) the Top voltage for positive pulse undershoot measurements or less than equal to (≤) the Base voltage for negative pulse undershoot measurements.

Negative Pulse Undershoot equation:

$$\text{Negative Pulse Undershoot (\%)} = \begin{cases} \dfrac{localMaximum - Base}{Top - Base} & localMaximum > Base \\ 0 & localMaximum \le Base \end{cases}$$

Where localMaximum is achieved between falling edge mesial (lower) and rising edge mesial (lower) points on the waveform. After the waveform has reached a level less than or equal to (≤) the Base voltage.

Positive Pulse Undershoot equation:

$$\text{Positive Pulse Undershoot (\%)} = \begin{cases} \dfrac{Top - localMinimum}{Top - Base} & localMinimum < Top \\ 0 & localMinimum \geq Top \end{cases}$$

Where localMinimum is achieved between rising edge distal (upper) and falling edge distal (upper) points on the waveform. After the waveform has reached a level greater than or equal to (≥) the Top voltage.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether positive pulse undershoot or negative pulse undershoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether positive pulse undershoot or negative pulse undershoot is measured throughout the acquisition.

Query

```
:MEASure:UNDershoot? [<source>[,<direction]]
```

The :MEASure:UNDershoot? query returns the value of the Undershoot measurement.

Returned Format

```
[:MEASure:UNDershoot] <value>[,<result_state>]<NL>
```

<value> Ratio of undershoot to amplitude, in percent.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:VUNDershoot" on page 1210
- ":MEASure:OVERshoot" on page 1012
- ":MEASure:PREShoot" on page 1074

History New in version 11.30.

:MEASure:UNITinterval

Command

NOTE

This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:UNITinterval <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval command measures the unit interval value of the selected source. Use the :MEASure:DATarate command/query to measure the data rate of the source

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<data_rate> A real number representing the data rate.

Example This example measures the unit interval of channel 1.

```
myScope.WriteString ":MEASure:UNITinterval CHANnel1"
```

Query

```
:MEASure:UNITinterval? <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval? query returns the measured unit interval.

NOTE

This measurement requires the **Measure All Edges** setting to be enabled. You can do this by:

- Installing the measurement on the display (using the command syntax), which automatically enables the **Measure All Edges** setting
- Sending the ":ANALyze:AEDGes ON" command.

Returned Format

```
[:MEASure:UNITinterval] <value>[,<result_state>]<NL>
```

<value> Unit interval of the source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current unit interval of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:UNITinterval? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":ANALyze:AEDGes" on page 332

History Legacy command (existed before version 3.10).

:MEASure:VAMPlitude

Command

```
:MEASure:VAMPlitude [<source>]
```

The :MEASure:VAMPlitude command calculates the difference between the top and base voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAMPlitude command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example calculates the difference between the top and base voltage of the specified source.

```
myScope.WriteString ":MEASure:VAMPlitude CHANnel1"
```

Query

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query returns the calculated difference between the top and base voltage of the specified source.

Returned Format

```
[:MEASure:VAMPlitude] <value>[,<result_state>]<NL>
```

<value> Calculated difference between the top and base voltage.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current Vamplitude value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:VAMPlitude? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VAVerage

Command
```
:MEASure:VAVerage {CYCLe | DISPlay}[,<source>]
```

The :MEASure:VAVerage command calculates the average voltage over the displayed waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

CYCLe The CYCLe parameter instructs the average measurement to measure the average voltage across the first period on the display.

DISPlay The DISPlay parameter instructs the average measurement to measure all the data on the display.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example calculates the average voltage over the displayed waveform.

```
myScope.WriteString ":MEASure:VAVerage DISPlay,CHANnel1"
```

Query
```
:MEASure:VAVerage? {CYCLe | DISPlay}[,<source>]
```

The :MEASure:VAVerage? query returns the calculated average voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

Returned Format
```
[:MEASure:VAVerage] <value>[,<result_state>]<NL>
```

<value> The calculated average voltage.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current average voltage in the numeric variable, varAverage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VAVerage? DISPlay,CHANnel1 CHANnel1"
varAverage = myScope.ReadNumber
Debug.Print FormatNumber(varAverage, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VBASe

Command `:MEASure:VBASe [<source>]`

The :MEASure:VBASe command measures the statistical base of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VBASe command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the voltage at the base of the waveform.

```
myScope.WriteString ":MEASure:VBASe CHANnel1"
```

Query `:MEASure:VBASe? [<source>]`

The :MEASure:VBASe? query returns the measured voltage value at the base of the specified source.

Returned Format `[:MEASure:VBASe] <value>[,<result_state>]<NL>`

<value> Voltage at the base of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the current voltage at the base of the waveform to the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VBASe? CHANnel1"
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VLOWer

Command

```
:MEASure:VLOWer [<source>]
```

The :MEASure:VLOWer command measures the voltage value at the lower threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VLOWer command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:VLOWer?
```

The :MEASure:VLOWer? query returns the measured lower threshold of the selected source.

Returned Format

```
[:MEASure:VLOWer] <value>[,<result_state>]<NL>
```

<value> Voltage value at the lower threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured voltage at the lower threshold of the waveform to the numeric variable, varVlower, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VLOWer? CHANnel1"
varVlower = myScope.ReadNumber
Debug.Print FormatNumber(varVlower, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VMAX

Command

```
:MEASure:VMAX [<source>]
```

The :MEASure:VMAX command measures the absolute maximum voltage present on the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMAX command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the absolute maximum voltage on the waveform.

```
myScope.WriteString ":MEASure:VMAX CHANnel1"
```

Query

```
:MEASure:VMAX? [<source>]
```

The :MEASure:VMAX? query returns the measured absolute maximum voltage or maximum FFT amplitude present on the selected source waveform.

Returned Format

```
[:MEASure:VMAX] <value>[,<result_state>]<NL>
```

<value> Absolute maximum voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute maximum voltage on the waveform to the numeric variable, varMaximum, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VMAX? CHANnel1"
varMaximum = myScope.ReadNumber
Debug.Print FormatNumber(varMaximum, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VMIDdle

Command

```
:MEASure:VMIDdle [<source>]
```

The :MEASure:VMIDdle command measures the voltage level at the middle threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMIDdle command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASure:VMIDdle? [<source>]
```

The :MEASure:VMIDdle? query returns the voltage value at the middle threshold of the waveform.

Returned Format

```
[MEASure:VMIDdle] <value>[,<result_state>]<NL>
```

<value> The middle voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured middle voltage on the waveform to the numeric variable, varMiddle, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VMIDdle? CHANnel1"
varMiddle = myScope.ReadNumber
Debug.Print FormatNumber(varMiddle, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VMIN

Command

```
:MEASure:VMIN [<source>]
```

The :MEASure:VMIN command measures the absolute minimum voltage present on the selected source waveform. Sources are specified with :MEASure:SOURce or with the optional parameter following the :MEASure:VMIN command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the absolute minimum voltage on the waveform.

```
myScope.WriteString ":MEASure:VMIN CHANnel1"
```

Query

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query returns the measured absolute minimum voltage or minimum FFT amplitude present on the selected source waveform.

Returned Format

```
[:MEASure:VMIN] <value>[,<result_state>]<NL>
```

<value> Absolute minimum voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute minimum voltage on the waveform to the numeric variable, varMinimum, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VMIN? CHANnel1"
varMinimum = myScope.ReadNumber
Debug.Print FormatNumber(varMinimum, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VOVershoot

Command

```
:MEASure:VOVershoot [<source>[,<direction>]]
```

The :MEASure:VOVershoot command is similar to the overshoot measurement, but instead of returning the ratio of overshoot voltage to amplitude as a percent, it returns the local voltage of the overshoot.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether rising edge overshoot or falling edge overshoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether rising edge overshoot or falling edge overshoot is measured throughout the acquisition.

Example This example measures the local voltage of the overshoot.

```
myScope.WriteString ":MEASure:VOVershoot CHAN1"
```

Query

```
:MEASure:VOVershoot? [<source>[,<direction>]]
```

The :MEASure:VOVershoot? query returns the local voltage of the overshoot.

Returned Format

```
[:MEASure:VOVershoot] <value>[,<result_state>]<NL>
```

<value> The local voltage of the overshoot.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of overshoot in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VOVershoot? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VPP

Command
```
:MEASure:VPP [<source>]
```

The :MEASure:VPP command measures the maximum and minimum voltages on the selected source, then calculates the peak-to-peak voltage as the difference between the two voltages. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VPP command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the peak-to-peak voltage or FFT amplitude range of the previously selected source.

```
myScope.WriteString ":MEASure:VPP CHANnel1"
```

Query
```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query returns the specified source peak-to-peak voltage.

Returned Format
```
[:MEASure:VPP] <value>[,<result_state>]<NL>
```

<value> Peak-to-peak voltage of the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current peak-to-peak voltage in the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:VPP? CHANnel1"
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VPReshoot

Command

```
:MEASure:VPReshoot <source>[,<direction>]
```

The :MEASure:VPReshoot command is similar to the preshoot measurement, but instead of returning the ratio of preshoot voltage to amplitude as a percent, it returns the local voltage of the preshoot.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether rising edge preshoot or falling edge preshoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether rising edge preshoot or falling edge preshoot is measured throughout the acquisition.

Example This example measures the local voltage of the preshoot.

```
myScope.WriteString ":MEASure:VPReshoot CHAN1"
```

Query

```
:MEASure:VPReshoot? <source>[,<direction>]
```

History Legacy command (existed before version 3.10).

:MEASure:VRMS

Command

```
:MEASure:VRMS {CYCLe | DISPlay},{AC | DC} [,<source> [,{VOLT | DBM}]]
```

The :MEASure:VRMS command measures the RMS voltage of the selected waveform by subtracting the average value of the waveform from each data point on the display. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VRMS command.

CYCLe The CYCLe parameter instructs the RMS measurement to measure the RMS voltage across the first period of the display.

DISPlay The DISPlay parameter instructs the RMS measurement to measure all the data on the display. Generally, RMS voltage is measured across one waveform or cycle, however, measuring multiple cycles may be accomplished with the DISPlay option. The DISPlay parameter is also useful when measuring noise.

AC The AC parameter is used to measure the RMS voltage subtracting the DC component.

DC The DC parameter is used to measure RMS voltage including the DC component.

The AC RMS, DC RMS, and VAVG parameters are related as in this formula:

$$DCVRMS^2 = ACVRMS^2 + VAVG^2$$

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

VOLT | DBM Specifies the units of the measurement as either volts or decibels.

Example This example measures the RMS voltage of the previously selected waveform.

```
myScope.WriteString ":MEASure:VRMS CYCLe,AC,CHANnel1"
```

Query

```
:MEASure:VRMS? {CYCLe | DISPlay},{AC | DC} [,<source> [,{VOLT | DBM}]]
```

The :MEASure:VRMS? query returns the RMS voltage of the specified source.

Returned Format

```
[:MEASure:VRMS] <value>[,<result_state>]<NL>
```

<value> RMS voltage of the selected waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current AC RMS voltage over one period of the waveform in the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VRMS? CYCLe,AC,CHANnel1"
```

```
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

History Legacy command (existed before version 3.10).

Version 3.10: Added the VOLT and DBM parameters for specifying the measurement units.

:MEASure:VTIMe

Command

```
:MEASure:VTIMe <time>[,<source>]
```

The :MEASure:VTIMe command measures the voltage at the specified time. The time is referenced to the trigger event and must be on the screen. When an FFT function is the specified source, the amplitude at the specified frequency is measured. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTIMe command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<time> A real number for time from trigger in seconds, or frequency in Hertz for an FFT (when a function is set to FFT or a waveform memory contains an FFT).

Query

```
:MEASure:VTIMe? <time>[,<source>]
```

The :MEASure:VTIMe? query returns the measured voltage or amplitude.

Returned Format

```
[:MEASure:VTIMe] <value>[,<result_state>]<NL>
```

<value> Voltage at the specified time. When the source is an FFT function, the returned value is the vertical value at the horizontal setting passed in the VTIMe <time> parameter. The time parameter is in Hertz when an FFT function is the source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the voltage at 500 ms in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VTIMe? 500E-3,CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VTOP

Command

```
:MEASure:VTOP [<source>]
```

The :MEASure:VTOP command measures the statistical top of the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTOP command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the voltage at the top of the waveform.

```
myScope.WriteString ":MEASure:VTOP CHANnel1"
```

Query

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the measured voltage at the top of the specified source.

Returned Format

```
[:MEASure:VTOP] <value>[,<result_state>]<NL>
```

<value> Voltage at the top of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the top of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VTOP? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:VUNDershoot

Command

```
:MEASure:VUNDershoot [<source>[,<direction>]]
```

The :MEASure:VUNDershoot command installs a V undershoot measurement. This measurement is visible in the front-panel user interface Measurements window in the Results pane.

The Vundershoot measurement is similar to the measurement installed with :MEASure:UNDershoot, but instead of returning the ratio of undershoot voltage to amplitude as a percent, it returns the local Minimum or local Maximum voltage of the undershoot.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

<direction> {RISing | FALLing}

Specifies whether positive pulse V undershoot or negative pulse V undershoot is measured. When <direction> is specified, the <source> parameter is required.

When the "Measure All Edges" mode is OFF (see ":ANALyze:AEDGes" on page 332), the RISing and FALLing options specify whether the first rising or falling edge from the left side of the display grid is used.

When the "Measure All Edges" mode is ON, the RISing and FALLing options specify whether positive pulse V undershoot or negative pulse V undershoot is measured throughout the acquisition.

Query

```
:MEASure:VUNDershoot? [<source>[,<direction]]
```

The :MEASure:VUNDershoot? query returns the value of the V undershoot measurement.

Returned Format

```
[:MEASure:VUNDershoot] <value>[,<result_state>]<NL>
```

<value> The local Minimum V undershoot value for positive pulses or the local Maximum V undershoot value for negative pulses.

<result_state> If :MEASure:SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

See Also

- ":MEASure:UNDershoot" on page 1190
- ":MEASure:VOVershoot" on page 1203
- ":MEASure:VPReshoot" on page 1205

History New in version 11.30.

:MEASure:VUPPer

Command

```
:MEASure:VUPPer [<source>]
```

The :MEASure:VUPPer command measures the voltage value at the upper threshold of the waveform. Sources are specified with the MEASure:SOURce command or with the optional parameter following the :MEASure:VUPPer command.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Example This example measures the voltage at the upper threshold of the waveform.

```
myScope.WriteString ":MEASure:VUPPer CHANnel1"
```

Query

```
:MEASure:VUPPer? [<source>]
```

The :MEASure:VUPPer? query returns the measured upper threshold value of the selected source.

Returned Format

```
[:MEASure:VUPPer] <value>[,<result_state>]<NL>
```

<value> Voltage at the upper threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the upper threshold of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:VUPPer? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MEASure:WINDow

Command
```
:MEASure:WINDow {ZOOM | CGRade | {MAIN | ALL}}, {MEAS<N>}
```

The :MEASure:WINDow command specifies whether measurements are made in the ZOOM window (measurement gating), the CGRade (color grade) view, or over the entire acquisition (MAIN or ALL). The MAIN and ALL parameters are equivalent.

Not all measurements can be applied to the color grade view.

If MEAS<N> is omitted, the command attempts to apply the selected window to all active measurements.

<N> Can be an integer from 1 – 40.

Example This example gates Measurement 1 to the zoom window.

```
myScope.WriteString ":MEASure:WINDow ZOOM, MEAS1"
```

Query :MEASure:WINDow? {MEAS<N>}

This query returns whether the measurement is being performed on the zoomed portion of the waveform (ZOOM), the color grade view of the waveform (CGR) or the entire acquisition (MAIN or ALL).

If MEAS<N> is omitted on the query, it returns the window of the first measurement.

History Legacy command (existed before version 3.10).

Version 3.10: The short form of the command was changed from :MEAS:WIN to :MEAS:WIND.

Version 5.00: Added the CGRade (color grade) view as a measurement window option.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASure:XCORtie

Command

```
:MEASure:XCORtie <source1>, <source2>, <edge_dir>, <xcor_time_range>
```

The :MEASure:XCORtie command adds a cross-correlated TIE measurement to the oscilloscope's front-panel display.

This measurement uses the same *two-channel cross-correlation technique* (see "Two-Channel Cross-Correlation Lowers the Oscilloscope Noise Floor" in the oscilloscope's online help) that is used to lower the oscilloscope's phase noise measurement floor.

<source1>, <source2> {CHANnel<N>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<edge_dir> {RISing | FALLing | BOTH}

<xcor_time_range> {AUT | <manual_range>}

<manual_range> The time range in seconds in NR3 format.

Query

```
:MEASure:XCORtie? <source1>, <source2>, <edge_dir>, <xcor_time_range>
```

The :MEASure:XCORtie? query returns the measured cross-correlated TIE value.

Returned Format

```
<measured_value><NL>

<measured_value> ::= cross-correlated TIE value in seconds in NR3 format
```

See Also

- ":MEASure:TIEClock2" on page 1174

History New in version 10.10.

:MEASure:XCPeriod

Command

```
:MEASure:XCPeriod <source1>, <source2>, <edge_dir>, <xcor_time_range>
```

The :MEASure:XCPeriod command adds a cross-correlated Period measurement to the oscilloscope's front-panel display.

This measurement uses the same *two-channel cross-correlation technique* (see "Two-Channel Cross-Correlation Lowers the Oscilloscope Noise Floor" in the oscilloscope's online help) that is used to lower the oscilloscope's phase noise measurement floor.

<source1>, <source2> {CHANnel<N>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<edge_dir> {RISing | FALLing | BOTH}

<xcor_time_range> {AUT | <manual_range>}

<manual_range> The time range in seconds in NR3 format.

Query

```
:MEASure:XCPeriod? <source1>, <source2>, <edge_dir>, <xcor_time_range>
```

The :MEASure:XCPeriod? query returns the measured cross-correlated Period value.

Returned Format

```
<measured_value><NL>

<measured_value> ::= cross-correlated period value in seconds in NR3 for
mat
```

See Also

- ":MEASure:PERiod" on page 1051

History New in version 11.20.

:MEASure:ZTMAX

Command `:MEASure:ZTMAX {MEASurement<N>}`

When "Measure All Edges" is enabled and the measurement supports "Zoom To Max", the :MEASure:ZTMAX command adjusts the horizontal scale and position to zoom in on the maximum measured value.

Check the front panel user interface to see if a measurement supports "Zoom To Max" by right-clicking the measurement results. Typically, measurements that involve a time period support "Zoom To Max".

This command is the same as :MEASurement<N>:ZTMAX.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

See Also

- ":MEASure:ZTMIN" on page 1216
- ":MEASurement<N>:ZTMAX" on page 1221

History New in version 10.20.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASure:ZTMIN

Command `:MEASure:ZTMIN {MEASurement<N>}`

When "Measure All Edges" is enabled and the measurement supports "Zoom To Min", the :MEASure:ZTMIN command adjusts the horizontal scale and position to zoom in on the minimum measured value.

Check the front panel user interface to see if a measurement supports "Zoom To Min" by right-clicking the measurement results. Typically, measurements that involve a time period support "Zoom To Min".

This command is the same as :MEASurement<N>:ZTMIN.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

See Also

- ":MEASure:ZTMAX" on page 1215
- ":MEASurement<N>:ZTMIN" on page 1222

History New in version 10.20.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:CLEar

Command

```
:MEASurement<N>:CLEar
```

The :MEASurement<N>:CLEar command clears a single measurement.

<N> An integer, 1-40.

NOTE When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.

See Also

- ":MEASure:CLEar" on page 903

History New in version 10.20.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:NAME

Command

```
:MEASurement<N>:NAME <name>
```

The :MEASurement<N>:NAME command sets the name of the specified measurement to whatever string is given to <name>. This lets you give specific names to measurements displayed on the oscilloscope's screen.

<N> An integer, 1-40. This number represents the position of the measurement on screen in the Measurements results window.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

<name> A quoted string.

Query

```
:MEASurement<N>:NAME?
```

The :MEASurement<N>:NAME? query returns the name of the corresponding measurement.

History New in version 4.50.

Version 10.10: Now supports up to 20 measurements.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:POSition

Command

```
:MEASurement<N>:POSition <new_position>
```

The :MEASurement<N>:POSition command lets you reorder measurements within the Measurements window in the Results pane.

<new_position> An integer from 1-40.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

See Also

- ":MEASurement<N>:CLEar" on page 1217

History New in version 11.10.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:SOURce

Command

```
:MEASurement<N>:SOURce <source>[,<source>]
```

The :MEASurement<N>:SOURce command changes the source of an existing measurement in the Measurements tab of the user interface.

<N> An integer, 1-40. This number represents the position of the measurement on screen in the Measurements tab.

NOTE When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | DIGital<M> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see "Measurement Sources" on page 877.

Query

```
:MEASurement<N>:SOURce?
```

The :MEASurement<N>:SOURce? query returns the source(s) of the selected measurement.

Returned Format

```
[:MEASurement<N>:SOURce] <source>[,<source>]<NL>
```

Example This example places the currently specified measurement 1 source(s) in the string variable, strSource, then prints the contents of the variable to the computer's screen.

```
Dim strSource As String   ' Dimension variable.
myScope.WriteString ":MEASurement1:SOURce?"
strSource = myScope.ReadString
Debug.Print strSource
```

See Also

- ":MEASurement<N>:NAME" on page 1218

History New in version 4.50.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:ZTMAX

Command `:MEASurement<N>:ZTMAX`

When "Measure All Edges" is enabled and the measurement supports "Zoom To Max", the :MEASurement<N>:ZTMAX command adjusts the horizontal scale and position to zoom in on the maximum measured value.

Check the front panel user interface to see if a measurement supports "Zoom To Max" by right-clicking the measurement results. Typically, measurements that involve a time period support "Zoom To Max".

This command is the same as :MEASure:ZTMAX.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

See Also

- ":MEASurement<N>:ZTMIN" on page 1222
- ":MEASure:ZTMAX" on page 1215

History New in version 10.20.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

:MEASurement<N>:ZTMIN

Command `:MEASurement<N>:ZTMIN`

When "Measure All Edges" is enabled and the measurement supports "Zoom To Min", the :MEASurement<N>:ZTMIN command adjusts the horizontal scale and position to zoom in on the minimum measured value.

Check the front panel user interface to see if a measurement supports "Zoom To Min" by right-clicking the measurement results. Typically, measurements that involve a time period support "Zoom To Min".

This command is the same as :MEASure:ZTMIN.

<N> An integer, 1-40.

NOTE **When <N> is 10-40, the long form of the mnemonic, MEASurement<N>, is too long. In this case, you must use the short form, MEAS<N>.**

See Also

- ":MEASurement<N>:ZTMAX" on page 1221
- ":MEASure:ZTMIN" on page 1216

History New in version 10.20.

Version 11.25: The maximum number of measurements has increased from 20 to 40.

35 :MEASure Power Commands

These :MEASure commands are available when the Power Analysis application is enabled.

:MEASure:ANGLe

Command

```
:MEASure:ANGLe [<source>]
```

```
<source> ::= {POWer{1|2} | CHANnel<N> | WMEMory<R>}
```

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

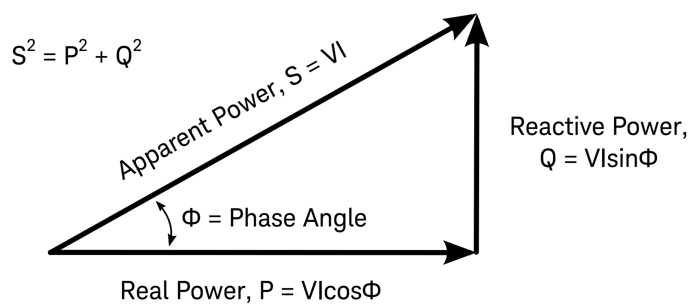
```
<R> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:ANGLe command installs a power phase angle measurement on screen.

The <source> parameter should be a power waveform. The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) The source can also be specified by the :MEASure:SOURce command.

Phase angle between current and voltage is a measure of power quality. In the *power triangle* (the right triangle where apparent_power^2 = real_power^2 + reactive_power^2), phase angle is the angle between the apparent power and the real power, indicating the amount of reactive power. Small phase angles equate to less reactive power.

Power Triangle



Query

```
:MEASure:ANGLe? [<source>]
```

The :MEASure:ANGLe query returns the measured power phase angle in degrees.

Returned Format

```
<return_value><NL>
```

```
<return_value> ::= the power phase angle in degrees in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History New in version 11.00.

:MEASure:APParent

Command

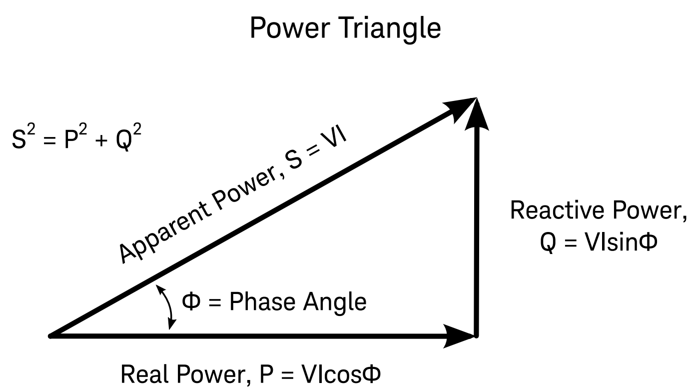
```
:MEASure:APParent [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<N>}

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

The :MEASure:APParent command installs an apparent power measurement on screen.

The <source1> parameter is the channel probing voltage and the <source2> parameter is the channel probing current. These sources can also be specified by the :MEASure:SOURce command.

Apparent power is a measure of power quality. It is the portion of AC line power flow due to stored energy which returns to the source in each cycle.

IRMS * VRMS



Query

```
:MEASure:APParent? [<source1>][,<source2>]
```

The :MEASure:APParent query returns the measured apparent power.

Returned Format

```
<return_value><NL>

<return_value> ::= the apparent power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History New in version 11.00.

:MEASure:CPLoss

Command

```
:MEASure:CPLoss [<source1>][,<source2>]

<source1> ::= {POWer{1|2} | CHANnel<N> | FUNCtion<F> | WMEMory<R>}

<source2> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:CPLoss command installs a power loss per cycle measurement on screen.

The <source1> parameter is typically a math multiply waveform or other waveform that represents power (voltage * current). The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) This source can also be specified by the :MEASure:SOURce command.

Power loss per cycle is Pn = (Vdsn * Idn) * (Time range of zoom window) * (Counter measurement of the voltage of the switching signal), where n is each sample.

This measurement operates when in zoom mode and the counter measurement is installed on the <source2> voltage of the switching signal.

Query

```
:MEASure:CPLoss? [<source1>][,<source2>]
```

The :MEASure:CPLoss query returns the switching loss per cycle in watts.

Returned Format

```
<return_value><NL>

<return_value> ::= the switching loss per cycle value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:SWITch:APPLy" on page 1392

History

New in version 11.00.

:MEASure:CRESt

Command

```
:MEASure:CRESt [<source>]

<source> ::= {CHANnel<N> | FUNCtion<F>}

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

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

The :MEASure:CRESt command installs a crest factor measurement on screen.

The <source> parameter is the channel probing current or voltage. This source can also be specified by the :MEASure:SOURce command.

Crest factor is a measure of power quality. It is the ratio between the instantaneous peak AC line current (or voltage) required by the load and the RMS current (or voltage). For example: Ipeak / IRMS or Vpeak / VRMS.

Query

```
:MEASure:CRESt? [<source>]
```

The :MEASure:CRESt query returns the measured crest factor.

Returned Format

```
<return_value><NL>

<return_value> ::= the crest factor value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History New in version 11.00.

:MEASure:EFFiciency

Command

```
:MEASure:EFFiciency
```

The :MEASure:EFFiciency command installs an efficiency (output power / input power) measurement on screen.

Before sending this command or query, you must specify the channels probing the input voltage, input current, output voltage, and output current (using the :POWer:SIGNals:SOURce:VOLTage<i> and :POWer:SIGNals:SOURce:CURRent<i> commands) and you must perform the automated signals setup (using the :POWer:SIGNals:AUTosetup EFFiciency command).

Query

```
:MEASure:EFFiciency?
```

The :MEASure:EFFiciency query returns the measured efficiency as a percent value.

Returned Format

```
<return_value><NL>

<return_value> ::= percent value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:EFFiciency:APPLy" on page 1331

History

New in version 11.00.

:MEASure:ELOSs

Command

```
:MEASure:ELOSs [<source>]

<source> ::= {POWer{1|2} | CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:ELOSs command installs an energy loss measurement on screen.

The <source> parameter is typically a math multiply waveform or other waveform that represents power (voltage * current). The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) This source can also be specified by the :MEASure:SOURce command.

Energy loss = Σ (Vds_n * Id_n) * sample size, where n is each sample.

Query

```
:MEASure:ELOSs? [<source>]
```

The :MEASure:ELOSs query returns the switching loss in joules.

Returned Format

```
<return_value><NL>

<return_value> ::= the energy loss value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:SWITch:APPLy" on page 1392

History New in version 11.00.

:MEASure:FACTor

Command

```
:MEASure:FACTor [<source>]
```

```
<source> ::= {POWer{1|2} | CHANnel<N> | WMEMory<R>}
```

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

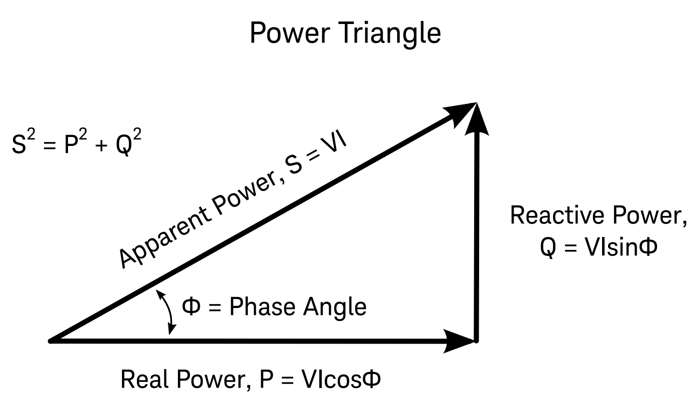
```
<R> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:FACTor command installs a power factor measurement on screen.

The <source> parameter should be a power waveform. The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) The source can also be specified by the :MEASure:SOURce command.

Power factor is a measure of power quality. It is the ratio of the actual AC line power to the apparent power:

Real Power / Apparent Power



Query

```
:MEASure:FACTor? [<source>]
```

The :MEASure:FACTor query returns the measured power factor.

Returned Format

```
<return_value><NL>
```

```
<return_value> ::= the power factor value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History New in version 11.00.

:MEASure:IPOWer

Command
```
:MEASure:IPOWer
```

The :MEASure:IPOWer command installs an input power measurement on screen.

Before sending this command or query, you must specify the channels probing the input voltage, input current, output voltage, and output current (using the :POWer:SIGNals:SOURce:VOLTage<i> and :POWer:SIGNals:SOURce:CURRent<i> commands) and you must perform the automated signals setup (using the :POWer:SIGNals:AUTosetup EFFiciency command).

Query
```
:MEASure:IPOWer?
```

The :MEASure:IPOWer query returns the measured input power.

Returned Format
```
<return_value><NL>
```

```
<return_value> ::= the input power value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:EFFiciency:APPLy" on page 1331

History New in version 11.00.

:MEASure:OFFTime

Command
```
:MEASure:OFFTime [<source1>][,<source2>]
```
```
<source1>, <source2> ::= {CHANnel<N>}
```
```
<N> ::= 1 to (# analog channels) in NR1 format
```
The :MEASure:OFFTime command installs an "off time" measurement on screen.

Turn off time measures the difference of time between when the input AC Voltage last falls to 10% of its maximum amplitude to the time when the output DC Voltage last falls to 10% of its maximum amplitude.

The <source1> parameter is the AC Voltage and the <source2> parameter is the DC Voltage. These sources can also be specified by the :MEASure:SOURce command.

Query
```
:MEASure:OFFTime? [<source1>][,<source2>]
```
The :MEASure:OFFTime query returns the measured turn off time.

Returned Format
```
<return_value><NL>
```
```
<return_value> ::= the time in seconds in NR3 format
```
See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:ONOFf:TEST" on page 1354
- ":POWer:ONOFf:APPLy" on page 1352

History New in version 11.00.

:MEASure:ONTime

Command

```
:MEASure:ONTime [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<N>}

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

The :MEASure:ONTime command installs an "on time" measurement on screen.

Turn on time measures the difference of time between when the input AC Voltage first rises to 10% of its maximum amplitude to the time when the output DC Voltage rises to 90% of its maximum amplitude.

The <source1> parameter is the AC Voltage and the <source2> parameter is the DC Voltage. These sources can also be specified by the :MEASure:SOURce command.

Query

```
:MEASure:ONTime? [<source1>][,<source2>]
```

The :MEASure:ONTime query returns the measured turn off time.

Returned Format

```
<return_value><NL>

<return_value> ::= the time in seconds in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:ONOFf:TEST" on page 1354
- ":POWer:ONOFf:APPLy" on page 1352

History New in version 11.00.

:MEASure:OUTPower

Command

```
:MEASure:OUTPower
```

The :MEASure:OUTPower command installs an output power measurement on screen.

Before sending this command or query, you must specify the channels probing the input voltage, input current, output voltage, and output current (using the :POWer:SIGNals:SOURce:VOLTage<i> and :POWer:SIGNals:SOURce:CURRent<i> commands) and you must perform the automated signals setup (using the :POWer:SIGNals:AUTosetup EFFiciency command).

Query

```
:MEASure:OUTPower?
```

The :MEASure:OUTPower query returns the measured output power.

Returned Format

```
<return_value><NL>

<return_value> ::= the output power value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:EFFiciency:APPLy" on page 1331

History New in version 11.00.

:MEASure:PCURrent

Command

```
:MEASure:PCURrent [<source>]

<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:PCURrent command installs a peak current measurement on screen.

The <source> parameter is the channel probing the current. This source can also be specified by the :MEASure:SOURce command.

This command measures the peak current when the power supply first turned on.

Query

```
:MEASure:PCURrent? [<source>]
```

The :MEASure:PCURrent query returns the measured peak current.

Returned Format

```
<return_value><NL>

<return_value> ::= the peak current value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:INRush:APPLy" on page 1346

History New in version 11.00.

:MEASure:PLOSs

Command

```
:MEASure:PLOSs [<source>]

<source> ::= {POWer{1|2} | CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:PLOSs command installs a power loss measurement on screen.

The <source> parameter is typically a math multiply waveform or other waveform that represents power (voltage * current). The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) This source can also be specified by the :MEASure:SOURce command.

Power loss is $P_n = Vds_n * Id_n$, where n is each sample.

Query

```
:MEASure:PLOSs? [<source>]
```

The :MEASure:PLOSs query returns the switching loss in watts.

Returned Format

```
<return_value><NL>

<return_value> ::= the power loss value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:SWITch:APPLy" on page 1392

History New in version 11.00.

:MEASure:RDSon

Command

```
:MEASure:RDSon [<source1>][,<source2>]
```

```
<source1>, <source2> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}
```

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

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

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

The :MEASure:RDSon command installs a power Rds(on) measurement on screen.

Rds(on) is the ON resistance between the drain and source of MOSFET. The Rds(on) characteristic is also published in the switching device data sheet.

Query

```
:MEASure:RDSon? [<source1>][,<source2>]
```

The :MEASure:RDSon? query returns the measured Rds(on) value.

Returned Format

```
<return_value><NL>
```

```
<return_value> ::= the Rds(on) value in NR3 format
```

See Also

- ":MEASure:VCESat" on page 1243

History New in version 11.00.

:MEASure:REACtive

Command

```
:MEASure:REACtive [<source>]

<source> ::= {POWer{1|2} | CHANnel<N> | WMEMory<R>}

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

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

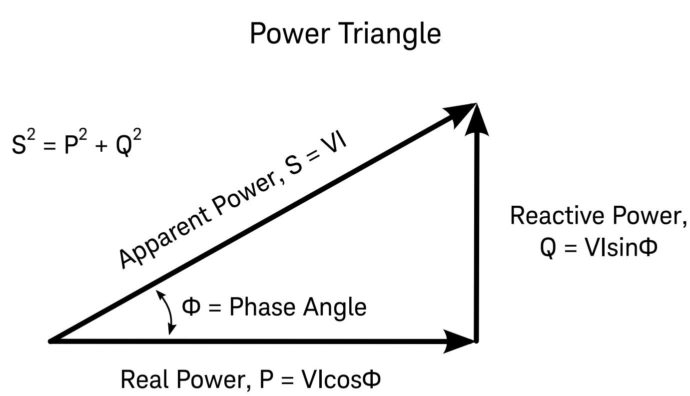
The :MEASure:REACtive command installs a reactive power measurement on screen.

The <source> parameter should be a power waveform. The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) The source can also be specified by the :MEASure:SOURce command.

Reactive power is a measure of power quality. It is the difference between apparent power and real power due to reactance. Using the *power triangle* (the right triangle where apparent_power^2 = real_power^2 + reactive_power^2):

$$\text{Reactive Power} = \sqrt{\text{Apparent Power}^2 - \text{Real Power}^2}$$

Reactive power is measured in VAR (Volts-Amps-Reactive).



Query

```
:MEASure:REACtive? [<source>]
```

The :MEASure:REACtive query returns the measured reactive power.

Returned Format

```
<return_value><NL>

<return_value> ::= the reactive power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History New in version 11.00.

:MEASure:REAL

Command

```
:MEASure:REAL [<source>]

<source> ::= {POWer{1|2} | CHANnel<N> | WMEMory<R>}

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

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

The :MEASure:REAL command installs a real power measurement on screen.

The <source> parameter is typically a math multiply waveform or other waveform that represents power (voltage * current). The POWer1 waveform is the input power generated by the Power Analysis feature by multiplying the specified input voltage and current channels. (The POWer2 waveform is the output power generated by the Power Analysis feature by multiplying the specified output voltage and current channels.) This source can also be specified by the :MEASure:SOURce command.

Real power is a measure of power quality. It is the portion of power flow that, averaged over a complete cycle of the AC waveform, results in net transfer of energy in one direction.

$$\text{Real Power} = \frac{1}{N}\sum_{n=0}^{N-1} V_n I_n$$

Where N refers to a instantaneous (Nth) sample, not the cycle.

Query

```
:MEASure:REAL? [<source>]
```

The :MEASure:REAL query returns the measured real power.

Returned Format

```
<return_value><NL>

<return_value> ::= the real power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:QUALity:APPLy" on page 1368

History

New in version 11.00.

:MEASure:RIPPle

Command

```
:MEASure:RIPPle [<source>]
```

```
<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}
```

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

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

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

The :MEASure:RIPPle command installs an output ripple measurement on screen.

The <source> parameter is the channel probing the output voltage. This source can also be specified by the :MEASure:SOURce command.

Output ripple is: Vmax - Vmin.

Query

```
:MEASure:RIPPle? [<source>]
```

The :MEASure:RIPPle query returns the measured output ripple.

Returned Format

```
<return_value><NL>
```

```
<return_value> ::= the output ripple value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:RIPPle:APPLy" on page 1369

History New in version 11.00.

:MEASure:TRESponse

Command

```
:MEASure:TRESponse [<source>]

<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:TRESponse command installs a transient response time measurement on screen.

The <source> parameter is the channel probing the output voltage. This source can also be specified by the :MEASure:SOURce command.

Transient response time = t2 – t1, where:

- t1 = The first time a voltage waveform exits the settling band.
- t2 = The last time it enters into the settling band.
- Settling band = +/-overshoot % of the steady state output voltage.

Query

```
:MEASure:TRESponse? [<source>]
```

The :MEASure:TRESponse query returns the measured transient response time.

Returned Format

```
<return_value><NL>

<return_value> ::= time in seconds for the overshoot to settle back
                   into the band in NR3 format
```

See Also

- ":MEASure:SOURce" on page 1124
- ":POWer:TRANsient:APPLy" on page 1398

History New in version 11.00.

:MEASure:VCESat

Command

```
:MEASure:VCESat [<source>]

<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

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

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

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

The :MEASure:VCESat command installs a power Vce(sat) measurement on screen.

Vce(sat) is the saturation voltage between the collector and emitter of a BJT. The Vce(sat) characteristic is also published in the switching device data sheet.

Query

```
:MEASure:VCESat? [<source>]
```

The :MEASure:VCESat? query returns the measured Vce(sat) value.

Returned Format

```
<return_value><NL>

<return_value> ::= the VCE(sat) value in NR3 format
```

See Also

- ":MEASure:RDSon" on page 1237

History New in version 11.00.

36 :MTESt (Mask Test) Commands

The MTESt subsystem commands and queries control the mask test features. Mask Testing automatically compares measurement results 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.

:MTESt:CPOLygons

Command
```
:MTESt:CPOLygons {{0 | OFF} | {1 | ON}}
```

When hardware-accelerated mask testing is not being used (see :MTESt:HWMask:ACTive?), the :MTESt:CPOLygons command specifies whether polygons should be counted in mask test results.

Query
```
:MTESt:CPOLygons?
```

The :MTESt:CPOLygons? query returns whether polygons will be counted in mask test results.

Returned Format
```
<setting><NL>

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

See Also

- ":MTESt:HWMask:ACTive?" on page 1260
- ":MTESt:STATistics" on page 1267
- ":MTESt:UHWMask" on page 1269

History New in version 11.20.

:MTESt:FENable

Command
```
:MTESt:FENable {{0 | OFF} | {1 | ON}}
```

The :MTESt:FENable command enables or disables the "Stop on Failure" option. It is the same as the :MTESt:RUMode:SOFailure command.

When enabled, when a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Query
```
:MTESt:FENable?
```

The :MTESt:FENable? query returns the "Stop on Failure" option setting.

Returned Format
```
<setting><NL>

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

See Also

- ":MTESt:RUMode:SOFailure" on page 1264
- ":MTESt:RUMode" on page 1262

History New in version 11.00.

:MTESt:FOLDing (Clock Recovery software only)

Command

```
:MTESt:FOLDing {{ON | 1} | {OFF | 0}} [,<source>]
```

The :MTESt:FOLDing command enables (ON) or disables (OFF) the display of the real-time eye.

Color grade must be enabled before enabling the real-time eye.

Refer to the :MEASure:CLOCk commands for clock recovery.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected}

If <source> is omitted:

- The real-time eye is enabled for all sources which currently have a color grade view on.
- When enabling real-time eye, the main waveform view is turned off.
- When disabling real-time eye, the main waveform view is turned on.

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example enables the display of the real-time eye.

```
myScope.WriteString ":MTESt:FOLDing ON"
```

Query

```
:MTESt:FOLDing? [<source>]
```

The :MTESt:FOLDing? query returns the current state of clock recovery folding.

If <source> is omitted, the query returns ON (1) if any source has real-time eye enabled.

Returned Format

```
[:MTESt:FOLDing] {1 | 0} <NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258
- ":ANALyze:CLOCk" on page 333
- ":ANALyze:CLOCk:METHod" on page 334

- ":ANALyze:CLOCk:METHod:ALIGn" on page 338
- ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339
- ":ANALyze:CLOCk:METHod:EDGE" on page 340
- ":ANALyze:CLOCk:METHod:JTF" on page 343
- ":ANALyze:CLOCk:METHod:OJTF" on page 345

History Legacy command (existed before version 3.10).

Version 5.00: Added the optional <source> parameter for specifying the waveform on which to enable/disable the real-time eye.

Version 5.50: When the <source> parameter is not provided, enabling the real-time eye will turn off the main waveform view, and disabling the real-time eye will turn on the main waveform view.

:MTESt:FOLDing:BITS

Command

```
:MTESt:FOLDing:BITS <source>,{BOTH | DEEMphasis | TRANsition
                    | PATTern, "<pattern>", <cursor>}
```

The :MTESt:FOLDing:BITS command determines the type of data bits used to create the eye pattern. The transition bits are greater in amplitude than the deemphasis bits. The PCI Express standard requires that compliance mask testing be done for both bit types.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, "Waveform Sources," starting on page 2011.

<pattern> An eight character string of level values:

- For typical NRZ (non-return-to-zero) signals, the levels are "1", "0", or "X" (for example, "101XX010").
- For PAM-3 (three-level) signals, the levels are "2", "1", "0", or "X" (for example, "01200X02").
- For PAM-4 (four-level) signals, the levels are "3", "2", "1", "0", or "X" (for example, "01230X03").
- For PAM-6 (six-level) signals, the levels are "5", "4", "3", "2", "1", "0", or "X" (for example, "01235X04").
- For PAM-8 (eight-level) signals, the levels are "7", "6", "5", "4", "3", "2", "1", "0", or "X" (for example, "01357X04").

<cursor> A value from 0 to 7 representing which bit is bit 0 from the LSB.

Example This example sets bit type to transition bits on the CHANnel1 real-time eye.

```
myScope.WriteString ":MTESt:FOLDing:BITS CHANnel1,TRANsition"
```

Query

```
:MTESt:FOLDing:BITS? <source>
```

The :MTESt:FOLDing:BITS? query returns the current setting of the real time eye bits.

Returned Format

```
[:MTESt:FOLDing:BITS] {BOTH | DEEMphasis | TRANsition
                      | PATT,<pattern>,<cursor>} <NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:BITS? CHANnel1"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:POSition" on page 1255

- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258

History Legacy command (existed before version 3.10).

Version 4.00: Added a PATTern option for specifying bit pattern qualification for the real-time eye display.

Version 5.00: Added the required <source> parameter to specify the waveform on which to set the real-time eye bit qualification.

Version 5.50: The <pattern> string can contain characters "2" and "3" when specified for PAM-4 signals.

Version 11.20: The <pattern> string can contain characters "2", "3", "4", or "5" when specified for PAM-6 signals or characters "2", "3", "4", "5", "6", or "7" when specified for PAM-8 signals.

:MTESt:FOLDing:COUNt:UI?

Query

```
:MTESt:FOLDing:COUNt:UI? [<source>]
```

The :MTESt:FOLDing:COUNt:UI? query returns the number of unit intervals in the real time eye.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Returned Format

```
[:MTESt:FOLDing:COUNt:UI] <UI_count><NL>
```

The UI count returned is a floating-point value.

Example

```
myScope.WriteString ":MTESt:FOLDing:COUNt:UI? CHANnel1"
varUiCount = myScope.ReadNumber
Debug.Print FormatNumber(varUiCount, 0)
```

See Also

- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258

History

New in version 5.50. This query replaces part of the now deprecated query ":MTESt:FOLDing:COUNt?" on page 1856.

Version 5.52: The <source> parameter is now optional.

:MTESt:FOLDing:COUNt:WAVeforms?

Query

```
:MTESt:FOLDing:COUNt:WAVeforms? [<source>]
```

The :MTESt:FOLDing:COUNt:WAVeforms? query returns the number of waveforms in the real time eye.

<source>

{CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Returned Format

```
[:MTESt:FOLDing:COUNt:WAVeforms] <Wfm_count><NL>
```

The Wfm count returned is an integer.

Example

```
myScope.WriteString ":MTESt:FOLDing:COUNt:WAVeforms? CHANnel1"
strWfmCount = myScope.ReadString
Debug.Print strWfmCount
```

See Also

- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258

History

New in version 5.50. This query replaces part of the now deprecated query ":MTESt:FOLDing:COUNt?" on page 1856.

Version 5.52: The <source> parameter is now optional.

:MTESt:FOLDing:POSition

Command

```
:MTESt:FOLDing:POSition <UI_position> [,<source>]
```

The :MTESt:FOLDing:POSition command sets the real-time eye horizontal center position in unit intervals.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, this command sets the position for all sources.

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the real-time eye horizontal center position to -0.300 UI.

```
myScope.WriteString ":MTESt:FOLDing:POSition -0.300"
```

Query

```
:MTESt:FOLDing:POSition? [<source>]
```

The :MTESt:FOLDing:POSition? query returns the real-time eye horizontal center position.

If <source> is omitted, the query returns the position for the first real-time eye.

Returned Format

```
[:MTESt:FOLDing:POSition] <UI_position> <NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:POSition?"
strUiPosition = myScope.ReadString
Debug.Print strUiPosition
```

See Also

- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258

History New in version 5.00.

:MTESt:FOLDing:SCALe

Command

```
:MTESt:FOLDing:SCALe <UI_scale> [,<source>]
```

The :MTESt:FOLDing:SCALe command sets the real-time eye horizontal scale, that is, the number of unit intervals (UIs) shown on screen.

<source> {CHANNel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, this command sets the number of unit intervals for all sources.

For more information on <source> parameters, see Chapter 50, "Waveform Sources," starting on page 2011.

Example This example sets the real-time eye horizontal scale to 2.0 UI.

```
myScope.WriteString ":MTESt:FOLDing:SCALe 2.0"
```

Query

```
:MTESt:FOLDing:SCALe? [<source>]
```

The :MTESt:FOLDing:SCALe? query returns the current real-time eye horizontal scale.

If <source> is omitted, the query returns the number of unit intervals for the first real-time eye.

Returned Format

```
[:MTESt:FOLDing:SCALe] <UI_scale><NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:SCALe?"
strUiScale = myScope.ReadString
Debug.Print strUiScale
```

See Also

- ":MTESt:FOLDing:TSCale" on page 1258
- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing:TPOSition" on page 1257

History New in version 5.00.

:MTESt:FOLDing:TPOSition

Command

```
:MTESt:FOLDing:TPOSition <position> [,<source>]
```

The :MTESt:FOLDing:TPOSition command sets the real-time eye horizontal center position in time.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, this command sets the position for all sources.

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the real-time eye horizontal center position to -0.300 ns.

```
myScope.WriteString ":MTESt:FOLDing:TPOSition -0.300E-09"
```

Query

```
:MTESt:FOLDing:TPOSition? [<source>]
```

The :MTESt:FOLDing:TPOSition? query returns the real-time eye horizontal center position.

If <source> is omitted, the query returns the position for the first real-time eye.

Returned Format

```
[:MTESt:FOLDing:TPOSition] <position> <NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:TPOSition?"
strTimePosition = myScope.ReadString
Debug.Print strTimePosition
```

See Also

- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:TSCale" on page 1258
- ":MTESt:FOLDing:SCALe" on page 1256

History New in version 5.10.

:MTESt:FOLDing:TSCale

Command

```
:MTESt:FOLDing:TSCale <scale> [,<source>]
```

The :MTESt:FOLDing:TSCale command sets the real-time eye horizontal scale per division in time.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L> | XT<X> | INPut | CORRected | ERRor | LFPR}

If <source> is omitted, this command sets the number of unit intervals for all sources.

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the real-time eye horizontal scale to 2.0 microseconds.

```
myScope.WriteString ":MTESt:FOLDing:TSCale 2.0E-06"
```

Query

```
:MTESt:FOLDing:TSCale? [<source>]
```

The :MTESt:FOLDing:TSCale? query returns the current real-time eye horizontal scale.

If <source> is omitted, the query returns the number of unit intervals for the first real-time eye.

Returned Format

```
[:MTESt:FOLDing:TSCale] <scale><NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:TSCale?"
strTimeScale = myScope.ReadString
Debug.Print strTimeScale
```

See Also

- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:COUNt:UI?" on page 1253
- ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254
- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:POSition" on page 1255

History New in version 5.10.

:MTESt:HAMPlitude

Command

```
:MTESt:HAMPlitude <upper_limit>
```

The :MTESt:HAMPlitude command sets the maximum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the upper limit used during mask testing.

<upper_limit> A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example sets the maximum pulse amplitude to 3.6 volts.

```
myScope.WriteString ":MTESt:HAMPlitude 3.6"
```

Query

```
:MTESt:HAMPlitude?
```

The :MTESt:HAMPlitude? query returns the current value of the maximum pulse amplitude.

Returned Format

```
[MTESt:HAMPlitude] <upper_limit><NL>
```

<upper_limit> A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example returns the current upper pulse limit and prints it to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MTESt:HAMPlitude?"
varULimit = myScope.ReadNumber
Debug.Print FormatNumber(varULimit, 0)
```

History Legacy command (existed before version 3.10).

:MTESt:HWMask:ACTive?

Query `:MTESt:HWMask:ACTive?`

When using hardware-accelerated mask testing is enabled (see :MTESt:UHWMask), the :MTESt:HWMask:ACTive? query returns whether hardware-acceleration is actually being used.

Certain oscilloscope states and settings (for example, **Fast Plot** being off or a protocol decode being enabled) will prevent hardware-accelerated mask testing from being used even when it is enabled. For more information, see the online help's section on "Hardware-Accelerated Mask Testing".

Returned Format `<setting><NL>`

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

See Also

- ":MTESt:CPOLygons" on page 1247
- ":MTESt:STATistics" on page 1267
- ":MTESt:UHWMask" on page 1269

History New in version 11.20.

:MTESt:LAMPlitude

Command

```
:MTESt:LAMPlitude <lower_limit>
```

The :MTESt:LAMPlitude command sets the minimum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the lower limit used during mask testing.

<lower_limit> A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example sets the minimum pulse amplitude to 2.4 volts.

```
myScope.WriteString ":MTESt:LAMPlitude 2.4"
```

Query

```
:MTESt:LAMPlitude?
```

The :MTESt LAMPlitude? query returns the current value of the minimum pulse amplitude.

Returned Format

```
[:MTESt:LAMPlitude] <lower_limit><NL>
```

<lower_limit> A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example returns the current lower pulse limit and prints it to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF  ! Response headers off.
myScope.WriteString ":MTESt:LAMPlitude?""
varULimit = myScope.ReadNumber
Debug.Print FormatNumber(varULimit, 0)
```

History Legacy command (existed before version 3.10).

:MTESt:RUMode

Command

```
:MTESt:RUMode {FORever | TIME, <time> | {WAVeforms, <number_of_waveforms
>}}
```

The :MTESt:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

If WAVeforms is selected, a second parameter is required indicating the number of failures that can occur or the number of samples or waveforms that are to be acquired.

FORever FORever runs the Mask Test until the test is turned off. This is used when you want a measurement to run continually and not to stop after a fixed number of failures. For example, you may want the Mask Test to run overnight and not be limited by a number of failures.

TIME TIME sets the amount of time in minutes that a mask test will run before it terminates.

<time> A real number: 0.1 to 1440.0

WAVeforms WAVeforms sets the maximum number of waveforms that are required before the mask test terminates.

<number_of_ waveforms> An integer: 1 to 1,000,000,000.

Example This example sets the mask test subsystem run until mode to continue testing until 500,000 waveforms have been gathered.

```
myScope.WriteString ":MTESt:RUMode WAVeforms,500E3"
```

Query

```
:MTESt:RUMode?
```

The query returns the currently selected termination condition and value.

Returned Format

```
[:MTESt:RUMode] {FORever | TIME,<time> | {WAVeforms, <number_of_waveform
s>}}<NL>
```

Example This example gets the current setting of the mask test run until mode from the oscilloscope and prints it on the computer screen.

```
Dim strMTEST_Runmode As String
myScope.WriteString ":MTESt:RUMode?"
strMTEST_Runmode = myScope.ReadString
Debug.Print strMTEST_Runmode
```

History Legacy command (existed before version 3.10).

:MTESt:RUMode:MOFailure

Command

```
:MTESt:RUMode:MOFailure {{ON | 1} | {OFF | 0}}
```

The :MTESt:RUMode:MOFailure command enables or disables the Perform Multipurpose On Failure run until option. When a mask test is run and a mask violation is detected, the configured Multipurpose action is performed.

Example

This example enables the Perform Multipurpose On Failure run until option.

```
myScope.WriteString ":MTESt:RUMode:MOFailure ON"
```

Query

```
:MTESt:RUMode:MOFailure?
```

The :MTESt:RUMode:MOFailure? query returns the current state of the Perform Multipurpose on Failure control.

Returned Format

```
[:MTESt:RUMode:MOFailure] {1 | 0}<NL>
```

See Also

- ":MTESt:FENable" on page 1248
- ":MTESt:RUMode" on page 1262

History

New in version 11.00.

:MTESt:RUMode:SOFailure

Command

```
:MTESt:RUMode:SOFailure {{ON | 1} | {OFF | 0}}
```

The :MTESt:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Example

This example enables the Stop On Failure run until criteria.

```
myScope.WriteString ":MTESt:RUMode:SOFailure ON"
```

Query

```
:MTESt:RUMode:SOFailure?
```

The :MTESt:RUMode:SOFailure? query returns the current state of the Stop on Failure control.

Returned Format

```
[:MTESt:RUMode:SOFailure] {1 | 0}<NL>
```

See Also

- ":MTESt:FENable" on page 1248
- ":MTESt:RUMode" on page 1262

History

Legacy command (existed before version 3.10).

:MTESt:RUNNing?

Query `:MTESt:RUNNing?`

The :MTESt:RUNNing? query returns whether the mask test is running.

Returned Format `<setting><NL>`

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

See Also

- ":MTESt:STARt" on page 1266
- ":MTESt:STOP" on page 1268

History New in version 11.00.

:MTESt:STARt

Command

```
:MTESt:STARt
```

The :MTESt:STARt command starts the mask test. The :MTESt:STARt command also starts the oscilloscope acquisition system.

Example

This example starts the mask test and acquisition system.

```
myScope.WriteString ":MTESt:STARt"
```

See Also

- ":MTESt:STOP" on page 1268
- ":MTESt:RUNNing?" on page 1265

History

Legacy command (existed before version 3.10).

:MTESt:STATistics

Command `:MTESt:STATistics <count-type>`

The :MTESt:STATistics command specifies the type(s) of counts that should be included in the mask testing pass/fail statistics results.

<count-type> {UI | WAVeforms | BOTH}

- UI — Unit interval counts are included in the mask test results.
- WAVeforms — Waveform counts are included in the mask test results.
- BOTH — Both waveform and UI counts are included in the mask test results.

  This option is not available when hardware-accelerated mask testing is being used (see :MTESt:HWMask:ACTive?).

Query `:MTESt:STATistics?`

The :MTESt:STATistics? query returns the type(s) of counts that will be included in the mask testing pass/fail statistics results.

Returned Format `<count-type><NL>`

`<count-type> ::= {UI | WAV | BOTH}`

See Also

- ":MTESt:CPOLygons" on page 1247
- ":MTESt:HWMask:ACTive?" on page 1260
- ":MTESt:UHWMask" on page 1269

History New in version 11.20.

:MTESt:STOP

Command

```
:MTESt:STOP
```

The :MTESt:STOP command stops the mask test. The :MTESt:STOP command does not stop the acquisition system.

Example

This example stops the mask test.

```
myScope.WriteString ":MTESt:STOP"
```

See Also

- ":MTESt:STARt" on page 1266
- ":MTESt:RUNNing?" on page 1265

History

Legacy command (existed before version 3.10).

:MTESt:UHWMask

Command `:MTESt:UHWMask {{0 | OFF} | {1 | ON}}`

The :MTESt:UHWMask command enables or disables using hardware-accelerated mask testing.

Query `:MTESt:UHWMask?`

The :MTESt:UHWMask? query returns whether using hardware-accelerated mask testing is enabled or disabled.

Returned Format `<setting><NL>`

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

See Also

- ":MTESt:CPOLygons" on page 1247
- ":MTESt:HWMask:ACTive?" on page 1260
- ":MTESt:STATistics" on page 1267

History New in version 11.20.

:MTESt<N>:AMASk:CREate

Command `:MTESt<N>:AMASk:CREate`

The :MTESt<N>:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the AMASk:XDELta, AMASk:YDELta, and 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<N>:SOURce command selects the channel and should be set before using this command.

<N> An integer, 1-8.

Example This example creates an automask using the current XDELta and YDELta units settings.

```
myScope.WriteString ":MTESt1:AMASk:CREate"
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AMASk:SAVE

Command

```
:MTESt<N>:AMASk:{SAVE | STORe} "<filename>"
```

NOTE The :MTESt<N>::AMASk:STORe command is equivalent to the :MTESt<N>:AMASk:SAVE command.

The :MTESt<N>:AMASk:SAVE command saves the automask generated mask to a file. If an automask has not been generated, an error occurs.

<N> An integer, 1-8.

<filename> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name. The default save path is C:\Users\Public\Documents\Infiniium\masks. The filename must have a .msk or .MSK extension or the command will fail.

Example This example saves the automask generated mask to a file named "FILE1.MSK".

```
myScope.WriteString ":MTESt1:AMASk:SAVE ""FILE1.MSK"""
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AMASk:SOURce

Command

```
:MTESt<N>:AMASk:SOURce <source>
```

When a mask is enabled (:MTESt<N>:ENABle ON), the :MTESt<N>:AMASk:SOURce command selects the source for the interpretation of the AMASk:XDELta and AMASk:YDELta parameters when 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: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.

<N> An integer, 1-8.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | CLOCk | EQUalized<L> | MTRend | MSPectrum | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example sets the automask source to Channel 1.

```
myScope.WriteString ":MTESt1:AMASk:SOURce CHANnel1"
```

Query

```
:MTESt<N>:AMASk:SOURce?
```

The :MTESt<N>:AMASk:SOURce? query returns the currently set source.

Returned Format

```
[:MTESt<N>:AMASk:SOURce] {CHAN<N> | DIFF<D> | COMM<C> | WMEM<R>
    | FUNC<F> | CLOC | EQU<L> | MTR | MSP | XT<X> | INP | CORR | ERRor
    | LFPR}<NL>
```

Example This example gets the source setting for automask and prints the result on the computer display.

```
Dim strAmask_source As String
myScope.WriteString ":MTESt1:AMASk:SOURce?"
strAmask_source = myScope.ReadString
Debug.Print strAmask_source
```

See Also

- ":MTESt<N>:ENABle" on page 1284

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AMASk:UNITs

Command

```
:MTESt<N>:AMASk:UNITs {CURRent | DIVisions}
```

The :MTESt<N>:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by AMASk:XDELta and AMASk:YDELta commands.

<N> An integer, 1-8.

CURRent When set to CURRent, the mask test subsystem uses the units as set by the :CHANnel:UNITs command, usually time for ΔX and voltage for ΔY.

DIVisions When set to 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.

Example This example sets the measurement units for automasking to the current :CHANnel:UNITs setting.

```
myScope.WriteString ":MTESt1:AMASk:UNITs CURRent"
```

Query

```
:MTESt<N>:AMASk:UNITs?
```

The AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Returned Format

```
[:MTESt<N>:AMASk:UNITs] {CURRent | DIVision}<NL>
```

Example This example gets the automask units setting, then prints the setting on the screen of the computer.

```
Dim strAutomask_units As String
myScope.WriteString ":MTESt1:AMASk:UNITs?"
strAutomask_units = myScope.ReadString
Debug.Print strAutomask_units
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AMASk:XDELta

Command

```
:MTESt<N>:AMASk:XDELta <xdelta_value>
```

The :MTESt<N>: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.

<N> An integer, 1-8.

<xdelta_value> A value for the horizontal tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for AMASk:UNITs is DIVisions, the same xdelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example This example sets the units to divisions and sets the ΔX tolerance to one-eighth of a division.

```
myScope.WriteString ":MTESt1:AMASk:UNITs DIVisions"
myScope.WriteString ":MTESt1:AMASk:XDELta 125E-3"
```

Query

```
:MTESt<N>:AMASk:XDELta?
```

The 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 AMASk:UNITs query.

Returned Format

```
[:MTESt<N>:AMASk:XDELta] <xdelta_value><NL>
```

Example This example gets the measurement system units and ΔX settings for automasking from the oscilloscope and prints the results on the computer screen.

```
Dim strAutomask_units As String
Dim strAutomask_xdelta As String
myScope.WriteString ":MTESt1:AMASk:UNITs?"
strAutomask_units = myScope.ReadString
myScope.WriteString ":MTESt1:AMASk:XDELta?"
strAutomask_xdelta = myScope.ReadString
Debug.Print strAutomask_units
Debug.Print strAutomask_xdelta
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AMASk:YDELta

Command

```
:MTESt<N>:AMASk:YDELta <ydelta_value>
```

The :MTESt<N>: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.

This command requires that mask testing be enabled, otherwise a settings conflict error message is displayed. See :MTESt<N>:ENABle for information on enabling mask testing.

<N> An integer, 1-8.

<ydelta_value> A value for the vertical tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for AMASk:UNITs is DIVisions, the same ydelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example This example sets the units to current and sets the ΔY tolerance to 30 mV, assuming that the current setting specifies volts in the vertical direction.

```
myScope.WriteString ":MTESt1:AMASk:UNITs CURRent"
myScope.WriteString ":MTESt1:AMASk:YDELta 30E-3"
```

Query

```
:MTESt<N>:AMASk:YDELta?
```

The 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 AMASk:UNITs query.

Returned Format

```
[:MTESt<N>:AMASk:YDELta] <ydelta_value><NL>
```

Example This example gets the measurement system units and ΔY settings for automasking from the oscilloscope and prints the results on the computer screen.

```
Dim strAutomask_units As String
Dim strAutomask_ydelta As String
myScope.WriteString ":MTESt1:AMASk:UNITs?"
strAutomask_units = myScope.ReadString
myScope.WriteString ":MTESt1:AMASk:YDELta?"
strAutomask_ydelta = myScope.ReadString
Debug.Print strAutomask_units
Debug.Print strAutomask_ydelta
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:FAILures?

Query

```
:MTESt<N>:COUNt:FAILures? REGion<number>
```

The MTESt:COUNt:FAILures? query returns the number of failures that occurred within a particular mask region.

The value 9.999E37 is returned if mask testing is not enabled or if you specify a region number that is unused.

<N> An integer, 1-8.

<number> An integer, 1 through 8, designating the region for which you want to determine the failure count.

Returned Format

```
[:MTESt<N>:COUNt:FAILures] REGion<number><number_of_failures> <NL>
```

<number_of_failures> The number of failures that have occurred for the designated region.

Example This example determines the current failure count for region 3 and prints it on the computer screen.

```
Dim strMask_failures As String
myScope.WriteString ":MTESt1:COUNt:FAILures? REGion3"
strMask_failures = myScope.ReadString
Debug.Print strMask_failures
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:FUI?

Query

```
:MTESt<N>:COUNt:FUI?
```

The MTESt:COUNt:FUI? query returns the number of unit interval failures that have occurred.

<N> An integer, 1-8.

Returned Format

```
[:MTESt<N>:COUNt:FUI?] <unit_interval_failures> <NL>
```

<unit_interval_failures> The number of unit interval failures.

Example This example determines the current number of unit interval failures and prints it to the computer screen.

```
Dim strFailures As String
myScope.WriteString ":MTESt1:COUNt:FUI?"
strFailures = myScope.ReadString
Debug.Print strFailures
```

See Also

- ":MTESt<N>:COUNt:UI?" on page 1281
- ":MTESt<N>:COUNt:SUI?" on page 1280
- ":MTESt<N>:COUNt:WAVeforms?" on page 1282
- ":MTESt<N>:COUNt:FWAVeforms?" on page 1278

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:FWAVeforms?

Query `:MTESt<N>:COUNt:FWAVeforms?`

The :MTESt<N>: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, so if you wish to determine failures by region number, use the COUNt:FAILures? query.

This count may not always be available. It is available only when the following conditions are true:

- Mask testing was turned on before the histogram or color grade persistence, and
- No mask changes have occurred, including scaling changes, editing, or new masks.

The value 9.999E37 is returned if mask testing is not enabled, or if you have modified the mask.

<N> An integer, 1-8.

Returned Format `[:MTESt<N>:COUNt:FWAVeforms] <number_of_failed_waveforms><NL>`

<number_of_failed_waveforms> The total number of failed waveforms for the current test run.

Example This example determines the number of failed waveforms and prints the result on the computer screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTESt1:COUNt:FWAVeforms?"
strMask_fwaveforms = myScope.ReadString
Debug.Print strMask_fwaveforms
```

See Also

- ":MTESt<N>:COUNt:UI?" on page 1281
- ":MTESt<N>:COUNt:FUI?" on page 1277
- ":MTESt<N>:COUNt:SUI?" on page 1280
- ":MTESt<N>:COUNt:WAVeforms?" on page 1282

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:MARGin:FAILures?

Query `:MTESt<N>:COUNt:MARGin:FAILures? REGion<number>`

The :MTESt<N>:COUNt:MARGin:FAILures? query returns the margin failure count for a specified region.

<N> An integer, 1-8.

<number> An integer, 1-8.

Returned Format `<failures><NL>`

`<failures> ::= number of failures in NR3 format.`

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:SUI?

Query

```
:MTESt<N>:COUNt:SUI?
```

The :MTESt<N>:COUNt:SUI? query returns the total number of samples that have been mask tested in the UI bit time.

This count is valid only when mask testing a real-time eye.

<N> An integer, 1-8.

Returned Format

```
[:MTESt<N>:COUNt:SUI?] <samples_tested><NL>
```

<samples_tested> The total number of samples that have been mask tested in the UI bit time.

See Also

- ":MTESt<N>:COUNt:UI?" on page 1281
- ":MTESt<N>:COUNt:FUI?" on page 1277
- ":MTESt<N>:COUNt:WAVeforms?" on page 1282
- ":MTESt<N>:COUNt:FWAVeforms?" on page 1278

History New in version 6.00.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:UI?

Query

```
:MTESt<N>:COUNt:UI?
```

The MTESt:COUNt:UI? query returns the number of unit intervals that have been mask tested.

<N> An integer, 1-8.

Returned Format

```
[:MTESt<N>:COUNt:UI?] <unit_intervals_tested> <NL>
```

<unit_intervals_tested> The number of unit intervals tested.

Example This example determines the current number of unit intervals tested and prints it to the computer screen.

```
Dim strUnit_intervals As String
myScope.WriteString ":MTESt1:COUNt:UI?"
strUnit_intervals = myScope.ReadString
Debug.Print strUnit_intervals
```

See Also

- ":MTESt<N>:COUNt:FUI?" on page 1277
- ":MTESt<N>:COUNt:SUI?" on page 1280
- ":MTESt<N>:COUNt:WAVeforms?" on page 1282
- ":MTESt<N>:COUNt:FWAVeforms?" on page 1278

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:COUNt:WAVeforms?

Query `:MTESt<N>:COUNt:WAVeforms?`

The :MTESt<N>:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run. The value 9.999E37 is returned if mask testing is not enabled.

<N> An integer, 1-8.

Returned Format `[:MTESt<N>:COUNt:WAVeforms] <number_of_waveforms><NL>`

<number_of_ waveforms> The total number of waveforms for the current test run.

Example This example determines the number of waveforms acquired in the current test run and prints the result on the computer screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTESt1:COUNt:WAVeforms?"
varMask_waveforms = myScope.ReadNumber
Debug.Print FormatNumber(varMask_waveforms, 0)
```

See Also

- ":MTESt<N>:COUNt:UI?" on page 1281
- ":MTESt<N>:COUNt:FUI?" on page 1277
- ":MTESt<N>:COUNt:SUI?" on page 1280
- ":MTESt<N>:COUNt:FWAVeforms?" on page 1278

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:DELete

Command

```
:MTESt<N>:DELete
```

The :MTESt<N>:DELete command clears the currently loaded mask.

<N> An integer, 1-8.

Example This example clears the currently loaded mask.

```
myScope.WriteString ":MTESt1:DELete"
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:ENABle

Command

```
:MTESt<N>:ENABle {{ON | 1} | {OFF | 0}}
```

The :MTESt<N>:ENABle command enables or disables the mask test features.

<N> An integer, 1-8.

ON Enables the mask test features.

OFF Disables the mask test features.

Example This example enables the mask test features.

```
myScope.WriteString ":MTESt1:ENABle ON"
```

Query

```
:MTESt<N>:ENABle?
```

The :MTESt<N>:ENABle? query returns the current state of mask test features.

Returned Format

```
[MTESt:ENABle] {1 | 0}<NL>
```

Example This example places the current value of the mask test state in the numeric variable varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTESt1:ENABle?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:INVert

Command

```
:MTESt<N>:INVert {{ON | 1} | {OFF | 0}}
```

The :MTESt<N>:INVert command inverts the mask for testing negative-going pulses. The trigger level and mask offset are also adjusted. Not all masks support negative-going pulse testing, and for these masks, the command is ignored.

<N> An integer, 1-8.

Example This example inverts the mask for testing negative-going pulses.

```
myScope.WriteString ":MTESt1:INVert ON"
```

Query

```
:MTESt<N>:INVert?
```

The :MTESt<N>:INVert? query returns the current inversion setting.

Returned Format

```
[:MTESt<N>:INVert] {1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:LOAD

Command

```
:MTESt<N>:LOAD "<filename>"
```

The :MTESt<N>:LOAD command loads the specified mask file. The default path for mask files is C:\Users\Public\Documents\Infiniium\masks. To use a different path, specify the complete path and file name.

<N> An integer, 1-8.

<filename> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

Example This example loads the mask file named "140md_itu_1.msk".

```
myScope.WriteString _
":MTESt1:LOAD ""C:\Users\Public\Documents\Infiniium\masks\
140md_itu_1.msk"""
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:AUTO:HITS

Command

```
:MTESt<N>:MARGin:AUTO:HITS <hit_count>
```

```
<hit_count> ::= an integer.
```

When the automatic margin method is selected, and the hit ratio method is selected, the :MTESt<N>:MARGin:AUTO:HITS command specifies the hit count.

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:AUTO:HITS?
```

The :MTESt<N>:MARGin:AUTO:HITS? query returns the hit count setting.

Returned Format

```
<hit_count><NL>
```

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:AUTO:HRATio

Command

```
:MTESt<N>:MARGin:AUTO:HRATio <hit_ratio>
```

```
<hit_ratio> ::= a floating-point number from 0.1 to 1E-12.
```

When the automatic margin method is selected, and the hit ratio method is selected, the :MTESt<N>:MARGin:AUTO:HRATio command specifies the hit ratio.

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:AUTO:HRATio?
```

The :MTESt<N>:MARGin:AUTO:HRATio? query returns the hit ratio setting.

Returned Format

```
<hit_ratio><NL>
```

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:AUTO:METHod

Command

```
:MTESt<N>:MARGin:AUTO:METHod <method>
```

```
<method> ::= {HITS | HRATio}
```

When the automatic margin method is selected, the :MTESt<N>:MARGin:AUTO:METHod command selects between the hit count and hit ratio and automatic margin methods:

- HITS — With the hit count automatic margin method, you specify the hit count with :MTESt<N>:MARGin:AUTO:HITS.
- HRATio — With the hit ratio automatic margin method, you specify the hit ratio with :MTESt<N>:MARGin:AUTO:HRATio.

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:AUTO:METHod?
```

The :MTESt<N>:MARGin:AUTO:METHod? query returns the automatic margin method setting.

Returned Format

```
<method><NL>
```

```
<method> ::= {HITS | HRATio}
```

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:METHod

Command

```
:MTESt<N>:MARGin:METHod <method>
```

```
<method> ::= {MANual | AUTO}
```

The :MTESt<N>:MARGin:METHod command selects between the manual and automatic margin methods:

- MANual — With the manual margin method, you specify a margin percent using :MTESt:MARGin:PERCent.
- AUTO — With the automatic margin method, you specify the auto margin method using :MTESt<N>:MARGin:AUTO:METHod.

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:METHod?
```

The :MTESt<N>:MARGin:METHod? query returns the margin type setting.

Returned Format

```
<method><NL>
```

```
<method> ::= {MANual | AUTO}
```

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:PERCent

Command

```
:MTESt<N>:MARGin:PERCent <percent>
```

```
<percent> ::= an integer from -100 to 100.
```

When the manual margin method is selected, the :MTESt<N>:MARGin:PERCent command specifies the margin percent.

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:PERCent?
```

The :MTESt<N>:MARGin:PERCent? query returns the margin percent setting.

Returned Format

```
<percent><NL>
```

See Also

- ":MTESt<N>:MARGin:STATe" on page 1292
- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:MARGin:STATe

Command

```
:MTESt<N>:MARGin:STATe {{0 | OFF} | {1 | ON}}
```

The :MTESt<N>:MARGin:STATe command enables or disables mask margin testing.

To enable mask margin testing, there must be a real-time eye and you must load a mask file that has a margin definition (*.mskx).

<N> An integer, 1-8.

Query

```
:MTESt<N>:MARGin:STATe?
```

The :MTESt<N>:MARGin:STATe? query returns the mask margin testing state.

Returned Format

```
<setting><NL>

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

See Also

- ":MTESt<N>:MARGin:METHod" on page 1290
- ":MTESt<N>:MARGin:PERCent" on page 1291
- ":MTESt<N>:MARGin:AUTO:METHod" on page 1289
- ":MTESt<N>:MARGin:AUTO:HITS" on page 1287
- ":MTESt<N>:MARGin:AUTO:HRATio" on page 1288
- ":MTESt<N>:COUNt:MARGin:FAILures?" on page 1279

History New in version 5.70.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:NREGions?

Query

```
:MTESt<N>:NREGions?
```

The :MTESt<N>:NREGions? query returns the number of regions that define the mask.

<N> An integer, 1-8.

Returned Format

```
[:MTESt<N>:NREGions] <regions><NL>
```

<regions> `An integer from 0 to 8.`

Example This example returns the number of mask regions.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTESt1:NREGions?"
varRegions = myScope.ReadNumber
Debug.Print FormatNumber(varRegions, 0)
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:BIND

Command

```
:MTESt<N>:SCALe:BIND {{ON | 1} | {OFF | 0}}
```

The :MTESt<N>:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control. 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 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 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 disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

<N> An integer, 1-8.

Example This example enables the Bind 1 & 0 Levels control.

```
myScope.WriteString ":MTESt1:SCALe:BIND ON"
```

Query

```
:MTESt<N>:SCALe:BIND?
```

The :MTESt<N>:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Returned Format

```
[:MTESt<N>:SCALe:BIND?] {1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:DRAW

Command

```
:MTESt<N>:SCALe:DRAW {{0 | OFF} | {1 | ON}}
```

The :MTESt<N>:SCALe:DRAW command specifies whether the mask bounding region is displayed.

When displayed, the mask bounding region lets you move and perhaps scale the gray mask regions relative to waveforms.

<N> An integer, 1-8.

Query

```
:MTESt<N>:SCALe:DRAW?
```

The :MTESt<N>:SCALe:DRAW? query returns whether the mask bounding region is displayed.

Returned Format

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

History New in version 10.10.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:X1

Command

```
:MTESt<N>:SCALe:X1 <x1_value>
```

The :MTESt<N>:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the 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:

$$X = (X \times \Delta X) + X1$$

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.

<N> An integer, 1-8.

<x1_value> A time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Example This example sets the X1 coordinate at 150 ms.

```
myScope.WriteString ":MTESt1:SCALe:X1 150E-3"
```

Query

```
:MTESt<N>:SCALe:X1?
```

The :MTESt<N>:SCALe:X1? query returns the current X1 coordinate setting.

Returned Format

```
[:MTESt<N>:SCALe:X1] <x1_value><NL>
```

Example This example gets the current setting of the X1 coordinate from the oscilloscope and prints it on the computer screen.

```
Dim strScale_x1 As String
myScope.WriteString ":MTESt1:SCALe:X1?"
strScale_x1 = myScope.ReadString
Debug.Print strScale_x1
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:XDELta

Command

```
:MTESt<N>:SCALe:XDELta <xdelta_value>
```

The :MTESt<N>: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:

$$X = (X \times \Delta X) + X1$$

<N> An integer, 1-8.

<xdelta_value> A time value specifying the distance of the X2 marker with respect to the X1 marker.

Example Assume that the period of the waveform you wish to test is 1 ms. Then the following example will set ΔX to 1 ms, ensuring that the waveform's period is between the X1 and X2 markers.

```
myScope.WriteString ":MTESt1:SCALe:XDELta 1E-6:
```

Query

```
:MTESt<N>:SCALe:XDELta?
```

The :MTESt<N>:SCALe:XDELta? query returns the current value of ΔX.

Returned Format

```
[:MTESt<N>:SCALe:XDELta] <xdelta_value><NL>
```

Example This example gets the value of ΔX from the oscilloscope and prints it on the computer screen.

```
Dim strScale_xdelta As String
myScope.WriteString ":MTESt1:SCALe:XDELta?""
strScale_xdelta = myScope.ReadString
Debug.Print strScale_xdelta
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:Y1

Command
```
:MTESt<N>:SCALe:Y1 <y_value>
```

The :MTESt<N>: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:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

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.

<N> An integer, 1-8.

<y1_value> A voltage value specifying the point at which Y=0.

Example This example sets the Y1 marker to -150 mV.

```
myScope.WriteString ":MTESt1:SCALe:Y1 -150E-3"
```

Query
```
:MTESt<N>:SCALe:Y1?
```

The SCALe:Y1? query returns the current setting of the Y1 marker.

Returned Format
```
[:MTESt<N>:SCALe:Y1] <y1_value><NL>
```

Example This example gets the setting of the Y1 marker from the oscilloscope and prints it on the computer screen.

```
Dim strScale_y1 As String
myScope.WriteString ":MTESt1:SCALe:Y1?"
strScale_y1 = myScope.ReadString
Debug.Print strScale_y1
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SCALe:Y2

Command

```
:MTESt<N>:SCALe:Y2 <y2_value>
```

The :MTESt<N>: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:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

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.

<N> An integer, 1-8.

<y2_value> A voltage value specifying the location of the Y2 marker.

Example This example sets the Y2 marker to 2.5 V.

```
myScope.WriteString ":MTESt1:SCALe:Y2 2.5"
```

Query

```
:MTESt<N>:SCALe:Y2?
```

The SCALe:Y2? query returns the current setting of the Y2 marker.

Returned Format

```
[:MTESt<N>:SCALe:Y2] <y2_value><NL>
```

Example This example gets the setting of the Y2 marker from the oscilloscope and prints it on the computer screen.

```
Dim strScale_y2 As String
myScope.WriteString ":MTESt1:SCALe:Y2?"
strScale_y2 = myScope.ReadString
Debug.Print strScale_y2
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:SOURce

Command

```
:MTESt<N>:SOURce {<source> | NONE}
```

The :MTESt<N>:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

Setting a mask source to NONE also has the effect of disabling that mask.

<N> An integer, 1-8.

<source> {CHANnel<N> | FUNCtion<F> | EQUalized<L> | WMEMory<R> | XT<X> | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example selects channel 1 as the mask test source.

```
myScope.WriteString ":MTESt1:SOURce CHANnel1"
```

Query

```
:MTESt<N>:SOURce?
```

The :MTESt<N>:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Returned Format

```
[:MTESt<N>:SOURce] {CHAN<N> | FUNC<F> | EQU<L> | WMEM<R> | XT<X>
   | INP | CORR | ERR | LFPR | NONE}<NL>
```

Example This example gets the mask test source setting and prints the result on the computer display.

```
Dim strAmask_source As String
myScope.WriteString ":MTESt1:SOURce?"
strAmask_source = myScope.ReadString
Debug.Print strAmask_source
```

See Also

- ":MTESt<N>:ENABle" on page 1284

History Legacy command (existed before version 3.10).

Version 6.00: Waveform memories can now be used as a source for mask testing.

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

Version 11.20: NONE can now be selected as the mask test source.

:MTESt<N>:TITLe?

Query

```
:MTESt<N>:TITLe?
```

The :MTESt<N>:TITLe? query returns the mask title which is a string of up to 23 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

<N> An integer, 1-8.

Returned Format

```
[:MTESt<N>:TITLe] <mask_title><NL>
```

<mask_title> A string of up to 23 ASCII characters which is the mask title.

Example This example places the mask title in the string variable and prints the contents to the computer's screen.

```
Dim strTitle As String
myScope.WriteString ":MTESt1:TITLe?"
strTitle = myScope.ReadString
Debug.Print strTitle
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

37 :POD<N> Commands

NOTE

The POD commands only apply to the MSO oscilloscopes.

:POD<N>:DISPlay

NOTE The POD commands only apply to the MSO oscilloscopes.

Command

```
:POD<N>[:DISPlay] {ON | OFF | 1 | 0}
```

The :POD<N>:DISPlay command enables or disables the view for the selected digital channel pod. Pod 1 has the digital channels 0 through 7, and pod 2 has the digital channels 8 through 15.

Displaying a pod automatically enables digital channels. See ENABle command in the root subsystem.

<N> An integer, 1-2.

Example This example turns on the display of bit 5 for the digital channels.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":POD2:DISPlay ON"
```

Query

```
:POD<N>[:DISPlay]?
```

The :POD<N>:DISPlay? query returns the value of the display setting for the pod.

Returned Format

```
[:POD<N>:DISPlay] {1 | 0}<NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 520
- ":ENABle DIGital" on page 267

History Legacy command (existed before version 3.10).

:POD<N>:NIBBle<N>:THReshold

Command

```
:POD<N>:NIBBle<N>:THReshold {CMOS50 | CMOS33 | CMOS25 | ECL | PECL
   | TTL | DIFFerential | <value>}
```

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:

- CMOS50 = 2.5 V
- CMOS33 = 1.65 V
- CMOS25 = 1.25 V
- ECL = -1.3 V
- PECL = 3.7 V
- TTL = 1.4 V
- DIFFerential = 0 V

<N> An integer, 1-2.

<value> A real number representing the voltage value which distinguishes a 1 logic level from a 0 logic level. Waveform voltages greater than the threshold are 1 logic levels while waveform voltages less than the threshold are 0 logic levels.

On MXR/EXR-Series oscilloscopes, the range of the threshold voltage is from -8 volts to 8 volts.

Example 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"
```

Query `:POD<N>:NIBBle<N>:THReshold?`

The :POD<N>:NIBBle<N>:THReshold? query returns the threshold value for the specified pod and nibble.

Returned Format

```
[:POD<N>:NIBBle<N>:THReshold] {CMOS50 | CMOS33 | CMOS25 | ECL | PECL
   | TTL | DIFF | <value>}<NL>
```

See Also

- ":DIGital<N>:THReshold" on page 524
- ":POD<N>:THReshold" on page 1308

History New in version 11.00.

:POD<N>:PSKew

Command

```
:POD<N>:PSKew <skew_value>
```

The :POD<N>:PSKew command lets you adjust the digital channels with respect to the analog channels when there is a time delay between the analog and digital channels. This can occur when different length cables are used.

<N> An integer, 1-2.

<skew_value> A real number for the skew value, in the range -1 ms to +1 ms.

Example This example sets the probe skew for all digital channels to 10 µs.

```
myScope.WriteString ":POD1:PSKew 10E-6"
```

Query

```
:POD<N>:PSKew?
```

The :POD<N>:PSKew? query returns the current probe skew setting for all digital channels.

Returned Format

```
[:POD<N>:PSKew] <skew_value><NL>
```

History Legacy command (existed before version 3.10).

:POD<N>:THReshold

Command

NOTE The POD commands only apply to the MSO oscilloscopes.

```
:POD<N>:THReshold {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFFerential | <value>}
```

The :POD<N>:THReshold command sets the logic threshold value for a pod. Pod 1 has the digital channels 0 through 7, and pod 2 has the digital channels 8 through 15. This command is equivalent to the :DIGital<N>:THReshold command.

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:

- CMOS50 = 2.5 V
- CMOS33 = 1.65 V
- CMOS25 = 1.25 V
- ECL = -1.3 V
- PECL = 3.7 V
- TTL = 1.4 V
- DIFFerential = 0 V

<N> An integer, 1-2.

<value> A real number representing the voltage value which distinguishes a 1 logic level from a 0 logic level. Waveform voltages greater than the threshold are 1 logic levels while waveform voltages less than the threshold are 0 logic levels.

On 9000 Series, 9000H Series, and S-Series mixed-signal oscilloscopes, the range of the threshold voltage is from -8 volts to 8 volts.

On 90000 X-Series and V-Series mixed-signal oscilloscopes, the range of the threshold voltage is from -3.75 volts to 3.75 volts.

Example This example sets the threshold to 1.8 volts for pod 2 (digital channels D15 through D8).

```
myScope.WriteString ":POD2:THReshold 1.8"
```

Query

```
:POD<N>:THReshold?
```

The :POD<N>:THReshold? query returns the threshold value for the specified pod.

Returned Format

```
[:POD<N>:THReshold] {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFF | <value>}<NL>
```

See Also

- ":DIGital<N>:THReshold" on page 524

History Legacy command (existed before version 3.10).

Version 4.50: Added the DIFFerential parameter for specifying the threshold voltage.

38 :POWer Commands

These :POWer commands are for the Power Analysis application.

:POWer:ANALysis

Command

```
:POWer:ANALysis <analysis>

<analysis> ::= {HARMonics | EFFiciency | INRush | RIPPle | MODulation
    | QUALity | PSRR | CLResponse | SLEW | RDSVce | SWITch | TRANsient
    | ONOFf | DESKew | SOArea}
```

The :POWer:ANALysis command selects the type of power analysis.

Query

```
:POWer:ANALysis?
```

The :POWer:ANALysis? query returns the selected type of power analysis.

Returned Format

```
<analysis><NL>

<analysis> ::= {HARM | EFF | INR | RIPP | MOD | QUAL | PSRR | CLR
    | SLEW | RDSV | SWIT | TRAN | ONOF | DESK | SOA}
```

See Also

- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:EFFiciency:APPLy" on page 1331
- ":POWer:RIPPle:APPLy" on page 1369
- ":POWer:MODulation:APPLy" on page 1349
- ":POWer:QUALity:APPLy" on page 1368
- ":POWer:SLEW:APPLy" on page 1390
- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:CYCLes:QUALity" on page 1372
- ":POWer:SIGNals:DURation:EFFiciency" on page 1373
- ":POWer:SIGNals:DURation:MODulation" on page 1374
- ":POWer:SIGNals:DURation:RIPPle" on page 1377
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:CLResponse:APPLy

Command `:POWer:CLResponse:APPLy`

The :POWer:CLResponse:APPLy command performs the control loop response (Bode) analysis to help you determine the margin of a control loop.

A Bode plot measurement plots gain and/or phase as a function of frequency.

This control loop response analysis requires an input sine wave (from the oscilloscope's waveform generator, Vi) be swept from a low to a high frequency while measuring Vi and Vo RMS voltages at each step frequency, using two channels of the oscilloscope.

For a gain plot, gain (A, in dB units) at each step frequency is computed as 20Log(Vo/Vi) and plotted using a math function waveform.

For a phase plot, the phase difference between the channels is measured at each step frequency. Phase measurements and plots are only possible if the input and output waveforms exceed 1 division peak-to-peak (>1 mVpp).

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

- "*ESR? — Event Status Register" on page 226
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:SINGle" on page 1322
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:DATA?

Query

```
:POWer:CLResponse:DATA? [SWEep | SINGle]
```

The :POWer:CLResponse:DATA? query returns data from the Control Loop Response (Bode) power analysis.

The comma-separated value format is suitable for spreadsheet analysis.

The SWEep or SINGle option specifies whether to get the data from a sweep or single-frequency analysis (see :POWer:CLResponse:FREQuency:MODE). If this option is not specified, the data from the sweep analysis is returned by default.

Returned Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA:GMARgin?" on page 1317
- ":POWer:CLResponse:DATA:GMARgin:FREQuency?" on page 1318
- ":POWer:CLResponse:DATA:PMARgin?" on page 1319
- ":POWer:CLResponse:DATA:PMARgin:FREQuency?" on page 1320
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:SINGle" on page 1322
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:DATA:GMARgin?

Query `:POWer:CLResponse:DATA:GMARgin?`

After the Control Loop Response (Bode) power analysis has been performed (see :POWer:CLResponse:APPLy), the :POWer:CLResponse:DATA:GMARgin? query returns the gain margin in dB.

Returned Format `<gain_margin><NL>`

`<gain_margin> ::= gain margin in dB in NR3 format.`

The query returns +9.9E+37 if the value cannot be calculated from the last sweep (that is, if there are no crossover points).

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:DATA:GMARgin:FREQuency?" on page 1318
- ":POWer:CLResponse:DATA:PMARgin?" on page 1319
- ":POWer:CLResponse:DATA:PMARgin:FREQuency?" on page 1320

History New in version 11.00.

:POWer:CLResponse:DATA:GMARgin:FREQuency?

Query `:POWer:CLResponse:DATA:GMARgin:FREQuency?`

After the Control Loop Response (Bode) power analysis has been performed (see :POWer:CLResponse:APPLy), the :POWer:CLResponse:DATA:GMARgin:FREQuency? query returns the 0° phase crossover frequency in Hz.

Returned Format `<frequency><NL>`

`<frequency> ::= 0 degrees phase crossover frequency in Hz in NR3 format`

The query returns +9.9E+37 if the value cannot be calculated from the last sweep (that is, if there are no crossover points).

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:DATA:GMARgin?" on page 1317
- ":POWer:CLResponse:DATA:PMARgin?" on page 1319
- ":POWer:CLResponse:DATA:PMARgin:FREQuency?" on page 1320

History New in version 11.00.

:POWer:CLResponse:DATA:PMARgin?

Query `:POWer:CLResponse:DATA:PMARgin?`

After the Control Loop Response (Bode) power analysis has been performed (see :POWer:CLResponse:APPLy), the :POWer:CLResponse:DATA:PMARgin? query returns the phase margin in degrees.

Returned Format `<phase_margin><NL>`

`<phase_margin> ::= phase margin in degrees in NR3 format.`

The query returns +9.9E+37 if the value cannot be calculated from the last sweep (that is, if there are no crossover points).

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:DATA:GMARgin?" on page 1317
- ":POWer:CLResponse:DATA:GMARgin:FREQuency?" on page 1318
- ":POWer:CLResponse:DATA:PMARgin:FREQuency?" on page 1320

History New in version 11.00.

:POWer:CLResponse:DATA:PMARgin:FREQuency?

Query `:POWer:CLResponse:DATA:PMARgin:FREQuency?`

After the Control Loop Response (Bode) power analysis has been performed (see :POWer:CLResponse:APPLy), the :POWer:CLResponse:DATA:PMARgin:FREQuency? query returns the 0 dB gain crossover frequency in Hz.

Returned Format `<frequency><NL>`

`<frequency> ::= 0dB gain crossover frequency in Hz in NR3 format.`

The query returns +9.9E+37 if the value cannot be calculated from the last sweep (that is, if there are no crossover points).

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:DATA:GMARgin?" on page 1317
- ":POWer:CLResponse:DATA:GMARgin:FREQuency?" on page 1318
- ":POWer:CLResponse:DATA:PMARgin?" on page 1319

History New in version 11.00.

:POWer:CLResponse:FREQuency:MODE

Command

```
:POWer:CLResponse:FREQuency:MODE <mode>

<mode> ::= {SWEep | SINGle}
```

The :POWer:CLResponse:FREQuency:MODE command specifies whether the analysis should be performed by sweeping through a range of frequencies (SWEep) or at a single frequency (SINGle).

The SINGle mode is useful for evaluating amplitudes at a single frequency, for example, near the expected 0 dB cross-over frequency. After running the test at a single frequency, you can manually adjust (increase) the waveform generator's amplitude until you begin to observe distortion in the waveforms on the oscilloscope's display. You can then use that amplitude at all frequencies in SWEep mode, or you can evaluate amplitudes at other frequencies in order to determine an optimized amplitude profile (see :POWer:CLResponse:WGEN:VOLTage:PROFile).

Query

```
:POWer:CLResponse:FREQuency:MODE?
```

The :POWer:CLResponse:FREQuency:MODE? query returns the frequency mode setting.

Returned Format

```
<mode><NL>

<mode> ::= {SWE | SING}
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:SINGle" on page 1322
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:FREQuency:SINGle

Command

```
:POWer:CLResponse:FREQuency:SINGle <value>[suffix]
```

```
<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 2000000}
```

```
[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:CLResponse:FREQuency:SINGle command sets the single frequency value. The control loop 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

```
:POWer:CLResponse:FREQuency:SINGle?
```

The :POWer:CLResponse:FREQuency:SINGle? query returns the single frequency setting.

Returned Format

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

```
<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 2000000}
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:FREQuency:STARt

Command

```
:POWer:CLResponse:FREQuency:STARt <value>[suffix]

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:CLResponse:FREQuency:STARt command sets the frequency sweep start value. The control loop 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

```
:POWer:CLResponse:FREQuency:STARt?
```

The :POWer:CLResponse:FREQuency:STARt? query returns the frequency sweep start setting.

Returned Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:FREQuency:STOP

Command

```
:POWer:CLResponse:FREQuency:STOP <value>[suffix]
```

```
<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}
```

```
[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:CLResponse:FREQuency:STOP command sets the frequency sweep stop value. The control loop 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

```
:POWer:CLResponse:FREQuency:STOP?
```

The :POWer:CLResponse:FREQuency:STOP? query returns the frequency sweep stop setting.

Returned Format

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

```
<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:SOURce:INPut

Command

```
:POWer:CLResponse:SOURce:INPut <source>

<source> ::= CHANnel<n>

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

The :POWer:CLResponse:SOURce:INPut command selects the oscilloscope channel that is probing the power supply input.

Query

```
:POWer:CLResponse:SOURce:INPut?
```

The :POWer:CLResponse:SOURce:INPut? query returns the channel selection.

Returned Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:SOURce:OUTPut

Command

```
:POWer:CLResponse:SOURce:OUTPut <source>

<source> ::= CHANnel<n>

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

The :POWer:CLResponse:SOURce:OUTPut command selects the oscilloscope channel that is probing the power supply output.

Query

```
:POWer:CLResponse:SOURce:OUTPut?
```

The :POWer:CLResponse:SOURce:OUTPut? query returns the channel selection.

Returned Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History

New in version 11.00.

:POWer:CLResponse:SWEep:POINts

Command

```
:POWer:CLResponse:SWEep:POINts <pts>
```

```
<pts> ::= number of points from 1 to 1000 in NR1 format
```

The :POWer:CLResponse:SWEep:POINts command specifies the total number of points in the frequency response analysis.

Query

```
:POWer:CLResponse:SWEep:POINts?
```

The :POWer:CLResponse:SWEep:POINts? query returns the number of points setting.

Returned Format

```
<pts><NL>
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History

New in version 11.00.

:POWer:CLResponse:WGEN:LOAD

Command `:POWer:CLResponse:WGEN:LOAD <impedance>`

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

The :POWer:CLResponse:WGEN:LOAD command sets the waveform generator expected output load impedance.

The output impedance of the Gen Out signal 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 `:POWer:CLResponse:WGEN:LOAD?`

The :POWer:CLResponse:WGEN:LOAD? query returns the waveform generator expected output load impedance setting.

Returned Format `<impedance><NL>`

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

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History New in version 11.00.

:POWer:CLResponse:WGEN:VOLTage

Command

```
:POWer:CLResponse:WGEN:VOLTage <amplitude>[,<range>]
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

```
<range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ
    | F10MHZ | F20MHZ}
```

The :POWer:CLResponse:WGEN:VOLTage command sets the waveform generator output amplitude(s).

When the waveform generator amplitude profile is enabled (with the :POWer:CLResponse:WGEN:VOLTage:PROFile command), you can set an initial ramp amplitude for each frequency range.

Without the <range> parameter, this command sets the waveform generator output amplitude used when the amplitude profile is disabled.

Query

```
:POWer:CLResponse:WGEN:VOLTage? [<range>]
```

The :POWer:CLResponse:WGEN:VOLTage? query returns the waveform generator output amplitude setting(s).

Returned Format

```
<amplitude><NL>
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 1330

History

New in version 11.00.

:POWer:CLResponse:WGEN:VOLTage:PROFile

Command

```
:POWer:CLResponse:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}}
```

The :POWer:CLResponse:WGEN:VOLTage:PROFile command enables or disables the ability to set initial waveform generator ramp amplitudes for each frequency range.

With amplitude profiling, you can use lower amplitudes at frequencies where the device under test (DUT) is sensitive to distortion and use higher amplitudes where the DUT is less sensitive to distortion. Power supply feedback networks are typically most sensitive near the 0 dB cross-over frequency.

You can often observe distortions during the test. If the input test sine wave begins to look lopsided, clipped, or somewhat triangular in shape (nonsinusoidal), you are probably encountering distortion due to overdriving your DUT. Optimizing test amplitudes to achieve the best dynamic range measurements is often an iterative process of running your frequency response measurements multiple times.

Query

```
:POWer:CLResponse:WGEN:VOLTage:PROFile?
```

The :POWer:CLResponse:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Returned Format

```
<setting><NL>

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

See Also

- ":POWer:CLResponse:APPLy" on page 1315
- ":POWer:CLResponse:DATA?" on page 1316
- ":POWer:CLResponse:FREQuency:MODE" on page 1321
- ":POWer:CLResponse:FREQuency:STARt" on page 1323
- ":POWer:CLResponse:FREQuency:STOP" on page 1324
- ":POWer:CLResponse:SWEep:POINts" on page 1327
- ":POWer:CLResponse:SOURce:INPut" on page 1325
- ":POWer:CLResponse:SOURce:OUTPut" on page 1326
- ":POWer:CLResponse:WGEN:LOAD" on page 1328
- ":POWer:CLResponse:WGEN:VOLTage" on page 1329

History New in version 11.00.

:POWer:EFFiciency:APPLy

Command `:POWer:EFFiciency:APPLy`

The :POWer:EFFiciency:APPLy command applies the efficiency power analysis.

Efficiency analysis tests the overall efficiency of the power supply by measuring the output power over the input power.

NOTE **Efficiency analysis requires a 4-channel oscilloscope because input voltage, input current, output voltage, and output current are measured.**

See Also

- ":POWer:EFFiciency:TYPE" on page 1332
- ":MEASure:EFFiciency" on page 1228
- ":MEASure:IPOWer" on page 1231
- ":MEASure:OUTPower" on page 1234

History New in version 11.00.

:POWer:EFFiciency:TYPE

Command

```
:POWer:EFFiciency:TYPE <type>
```

```
<type> ::= {DCDC | DCAC | ACDC | ACAC}
```

The :POWer:EFFiciency:TYPE command specifies the type of power that is being converted from the input to the output. This selection affects how the efficiency is measured.

Query

```
:POWer:EFFiciency:TYPE?
```

The :POWer:EFFiciency:TYPE? query returns the currently specified type setting.

Returned Format

```
<type><NL>
```

```
<type> ::= {DCDC | DCAC | ACDC | ACAC}
```

See Also

- ":POWer:EFFiciency:APPLy" on page 1331
- ":MEASure:EFFiciency" on page 1228
- ":MEASure:IPOWer" on page 1231
- ":MEASure:OUTPower" on page 1234

History New in version 11.00.

:POWer:ENABle

Command `:POWer:ENABle {{0 | OFF} | {1 | ON}}`

The :POWer:ENABle command enables or disables power analysis.

Query `:POWer:ENABle?`

The :POWer:ENABle query returns a 1 or a 0 showing whether power analysis is enabled or disabled, respectively.

Returned Format `{0 | 1}`

History New in version 11.00.

:POWer:HARMonics:APPLy

Command `:POWer:HARMonics:APPLy`

The :POWer:HARMonics:APPLy command applies the current harmonics analysis.

Switching power supplies draw a range of harmonics from the AC mains.

Standard limits are set for these harmonics because these harmonics can travel back to the supply grid and cause problems with other devices on the grid.

Use the Current Harmonics analysis to test a switching power supply's current harmonics to pre-compliance standard of IEC61000-3-2 (Class A, B, C, or D). The analysis presents up to 40 harmonics.

See Also

- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:DATA?

Query

```
:POWer:HARMonics:DATA?
```

The :POWer:HARMonics:DATA query returns the power harmonics results table data.

Returned Format

```
<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:DISPlay

Command

```
:POWer:HARMonics:DISPlay <display>

<display> ::= {TABLe | BAR | OFF}
```

The :POWer:HARMonics:DISPlay command specifies how to display the current harmonics analysis results:

- TABLe
- BAR — Bar chart.
- OFF — Harmonics measurement results are not displayed.

Query

```
:POWer:HARMonics:DISPlay?
```

The :POWer:HARMonics:DISPlay query returns the display setting.

Returned Format

```
<display><NL>

<display> ::= {TABL | BAR | OFF}
```

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:FAILcount?

Query `:POWer:HARMonics:FAILcount?`

Returns the current harmonics analysis' fail count. Non Spec values (that is, harmonics values not specified by the selected standard) are not counted.

Returned Format `<count><NL>`

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

See Also

- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:LINE

Command

```
:POWer:HARMonics:LINE <frequency>

<frequency> ::= {F50 | F60 | F400 | AUTO}
```

The :POWer:HARMonics:LINE command specifies the line frequency setting for the current harmonics analysis:

- F50 — 50 Hz.
- F60 — 60 Hz.
- F400 — 400 Hz.
- AUTO — Automatically determines the frequency of the Current waveform.

Query

```
:POWer:HARMonics:LINE?
```

The :POWer:HARMonics:LINE query returns the line frequency setting.

Returned Format

```
<frequency><NL>

<frequency> ::= {F50 | F60 | F400 | AUTO}
```

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:POWerfactor?

Query `:POWer:HARMonics:POWerfactor?`

The :POWer:HARMonics:POWerfactor query returns the power factor for IEC 61000-3-2 Standard Class C power factor value.

Returned Format `<value> ::= Class C power factor in NR3 format`

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:RPOWer

Command

```
:POWer:HARMonics:RPOWer <source>

<source> ::= {MEASured | USER}
```

When Class D is selected as the current harmonics analysis standard, the :POWer:HARMonics:RPOWer command specifies whether the Real Power value used for the current-per-watt measurement is measured by the oscilloscope or is defined by the user.

When USER is selected, use the :POWer:HARMonics:RPOWer:USER command to enter the user-defined value.

Query

```
:POWer:HARMonics:RPOWer?
```

The :POWer:HARMonics:RPOWer? query returns the Real Power source setting.

Returned Format

```
<source><NL>

<source> ::= {MEAS | USER}
```

See Also

- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:RPOWer:USER" on page 1341

History

New in version 11.00.

:POWer:HARMonics:RPOWer:USER

Command

```
:POWer:HARMonics:RPOWer:USER <value>
```

```
<value> ::= Watts from 1.0 to 600.0 in NR3 format
```

When Class D is selected as the current harmonics analysis standard and you have chosen to use a user-defined Real Power value (see :POWer:HARMonics:RPOWer), the :POWer:HARMonics:RPOWer:USER command specifies the Real Power value used in the current-per-watt measurement.

Query

```
:POWer:HARMonics:RPOWer:USER?
```

The :POWer:HARMonics:RPOWer:USER? query returns the user-defined Real Power value.

Returned Format

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

```
<value> ::= Watts from 1.0 to 600.0 in NR3 format
```

See Also

- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:RPOWer" on page 1340

History New in version 11.00.

:POWer:HARMonics:RUNCount?

Query

```
:POWer:HARMonics:RUNCount?
```

Returns the current harmonics analysis' run iteration count. Non Spec values (that is, harmonics values not specified by the selected standard) are not counted.

Returned Format

```
<count><NL>

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

See Also

- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:STANdard

Command

```
:POWer:HARMonics:STANdard <class>

<class> ::= {A | B | C | D}
```

The :POWer:HARMonics:STANdard command selects the standard to perform current harmonics compliance testing on.

- A — IEC 61000-3-2 Class A — for balanced three-phase equipment, household appliances (except equipment identified as Class D), tools excluding portable tools, dimmers for incandescent lamps, and audio equipment.
- B — IEC 61000-3-2 Class B — for portable tools.
- C — IEC 61000-3-2 Class C — for lighting equipment.
- D — IEC 61000-3-2 Class D — for equipment having a specified power according less than or equal to 600 W, of the following types: personal computers and personal computer monitors, television receivers.

Query

```
:POWer:HARMonics:STANdard?
```

The :POWer:HARMonics:STANdard query returns the currently set IEC 61000-3-2 standard.

Returned Format

```
<class><NL>

<class> ::= {A | B | C | D}
```

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STATus?" on page 1344
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:STATus?

Query `:POWer:HARMonics:STATus?`

The :POWer:HARMonics:STATus query returns the overall pass/fail status of the current harmonics analysis.

Returned Format `<status> ::= {PASS | FAIL | UNTested}`

See Also

- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:THD?" on page 1345

History New in version 11.00.

:POWer:HARMonics:THD?

Query `:POWer:HARMonics:THD?`

The :POWer:HARMonics:THD query returns the Total Harmonics Distortion (THD) results of the current harmonics analysis.

Returned Format `<value> ::= Total Harmonics Distortion in NR3 format`

See Also

- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:HARMonics:DATA?" on page 1335
- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:FAILcount?" on page 1337
- ":POWer:HARMonics:LINE" on page 1338
- ":POWer:HARMonics:POWerfactor?" on page 1339
- ":POWer:HARMonics:RUNCount?" on page 1342
- ":POWer:HARMonics:STANdard" on page 1343
- ":POWer:HARMonics:STATus?" on page 1344

History New in version 11.00.

:POWer:INRush:APPLy

Command `:POWer:INRush:APPLy`

The :POWer:INRush:APPLy command applies the inrush current analysis.

The Inrush current analysis measures the peak inrush current of the power supply when the power supply is first turned on.

See Also

- ":POWer:ITYPe" on page 1348
- ":POWer:INRush:NEXT" on page 1347
- ":MEASure:PCURrent" on page 1235

History New in version 11.00.

:POWer:INRush:NEXT

Command `:POWer:INRush:NEXT`

The :POWer:INRush:NEXT command goes to the next step of the inrush current analysis.

This command is equivalent to pressing the **Next** softkey on the oscilloscope front panel when prompted during the analysis.

See Also

- ":POWer:INRush:APPLy" on page 1346
- ":POWer:ITYPe" on page 1348

History New in version 11.00.

:POWer:ITYPe

Command

```
:POWer:ITYPe <type>
```

```
<type> ::= {DC | AC}
```

The :POWer:ITYPe command specifies the type of power that is being converted from the input (DC or AC). Your selection affects how the measurements are made.

This setting is used in the Inrush Current and Turn On/Turn Off tests.

Query

```
:POWer:ITYPe?
```

The :POWer:ITYPe? query returns the input power type setting.

Returned Format

```
<type><NL>
```

```
<type> ::= {DC | AC}
```

See Also

- ":POWer:INRush:APPLy" on page 1346
- ":POWer:ONOFf:APPLy" on page 1352

History New in version 11.00.

:POWer:MODulation:APPLy

Command `:POWer:MODulation:APPLy`

The :POWer:MODulation:APPLy command applies the selected modulation analysis type (:POWer:MODulation:TYPE).

The Modulation analysis measures the control pulse signal to a switching device (MOSFET) and observes the trending of the pulse width, duty cycle, period, frequency, etc. of the control pulse signal.

See Also

- ":POWer:MODulation:SOURce" on page 1350
- ":POWer:MODulation:TYPE" on page 1351
- ":MEASure:VAVerage" on page 1197
- ":MEASure:VRMS" on page 1206
- ":MEASure:PERiod" on page 1051
- ":MEASure:FREQuency" on page 947
- ":MEASure:PWIDth" on page 1077
- ":MEASure:NWIDth" on page 1007
- ":MEASure:DUTYcycle" on page 925
- ":MEASure:RISetime" on page 1085
- ":MEASure:FALLtime" on page 932

History New in version 11.00.

:POWer:MODulation:SOURce

Command

```
:POWer:MODulation:SOURce <source>
```

```
<source> ::= {V | I}
```

The :POWer:MODulation:SOURce command selects either the voltage source or the current source as the source for the modulation analysis.

Query

```
:POWer:MODulation:SOURce?
```

The :POWer:MODulation:SOURce query returns the selected source for the modulation analysis.

Returned Format

```
<source><NL>
```

```
<source> ::= {V | I}
```

See Also

- ":POWer:MODulation:APPLy" on page 1349
- ":POWer:MODulation:TYPE" on page 1351

History New in version 11.00.

:POWer:MODulation:TYPE

Command

```
:POWer:MODulation:TYPE <modulation>

<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency
                 | PWIDth | NWIDth | DUTYcycle | RISetime | FALLtime}
```

The :POWer:MODulation:TYPE command selects the type of measurement to make in the modulation analysis:

- VAVerage
- ACRMs
- VRATio
- PERiod
- FREQuency
- PWIDth (positive pulse width)
- NWIDth (negative pulse width)
- DUTYcycle
- RISetime
- FALLtime

Query

```
:POWer:MODulation:TYPE?
```

The :POWer:MODulation:TYPE query returns the modulation type setting.

Returned Format

```
<modulation><NL>

<modulation> ::= {VAV | ACRM | VRAT | PER | FREQ | PWID | NWID | DUTY
                 | RIS | FALL}
```

See Also

- ":POWer:MODulation:SOURce" on page 1350
- ":POWer:MODulation:APPLy" on page 1349
- ":MEASure:VAVerage" on page 1197
- ":MEASure:VRMS" on page 1206
- ":MEASure:PERiod" on page 1051
- ":MEASure:FREQuency" on page 947
- ":MEASure:PWIDth" on page 1077
- ":MEASure:NWIDth" on page 1007
- ":MEASure:DUTYcycle" on page 925
- ":MEASure:RISetime" on page 1085
- ":MEASure:FALLtime" on page 932

History New in version 11.00.

:POWer:ONOFf:APPLy

Command `:POWer:ONOFf:APPLy`

The :POWer:ONOFf:APPLy command applies the selected turn on/off analysis test (:POWer:ONOFf:TEST).

See Also

- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:THResholds" on page 1355
- ":POWer:ONOFf:TEST" on page 1354
- ":POWer:ONOFf:NEXT" on page 1353
- ":MEASure:ONTime" on page 1233
- ":MEASure:OFFTime" on page 1232

History New in version 11.00.

:POWer:ONOFf:NEXT

Command `:POWer:ONOFf:NEXT`

The :POWer:ONOFf:NEXT command goes to the next step of the turn on/turn off analysis.

This command is equivalent to pressing the **Next** softkey on the oscilloscope front panel when prompted during the analysis.

See Also

- ":POWer:ONOFf:THResholds" on page 1355
- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:ONOFf:TEST" on page 1354

History New in version 11.00.

:POWer:ONOFf:TEST

Command `:POWer:ONOFf:TEST {{0 | OFF} | {1 | ON}}`

The :POWer:ONOFf:TEST command selects whether turn on or turn off analysis is performed:

- ON — Turn On — measures the time taken to get the output voltage of the power supply after the input voltage is applied.
- OFF — Turn Off — measures the time taken for the output voltage of the power supply to turn off after the input voltage is removed.

Query `:POWer:ONOFf:TEST?`

The :POWer:ONOFf:TEST query returns the selected test type.

Returned Format `{0 | 1}`

See Also

- ":POWer:ONOFf:THResholds" on page 1355
- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:ONOFf:NEXT" on page 1353

History New in version 11.00.

:POWer:ONOFf:THResholds

Command

```
:POWer:ONOFf:THResholds <type>,<input_thr>,<output_thr>

<type> ::= {ON | OFF}

<input_thr> ::= percent from 0-100 in NR1 format

<output_thr> ::= percent from 0-100 in NR1 format
```

The :POWer:ONOFf:THResholds command specifies the input and output thresholds used in the Turn On/Turn Off analysis.

Turn On analysis determines how fast a turned on power supply takes to reach some percent of its steady state output. Turn on time is the time between T2 and T1 where:

- T1 = when the input voltage first rises to some percent (typically the 10% threshold) of its maximum amplitude.
- T2 = when the output DC voltage rises to some percent (typically the 90% threshold) of its maximum amplitude.

Turn Off analysis determines how fast a turned off power supply takes to reduce its output voltage to some percent of maximum. Turn off time is the time between T2 and T1 where:

- T1 = when the input voltage last falls to some percent (typically the 10% threshold) of its maximum amplitude.
- T2 = when the output DC voltage last falls to some percent (typically the 10% threshold) of its maximum amplitude.

Query

```
:POWer:ONOFf:THResholds? <type>
```

The :POWer:ONOFf:THResholds? query returns the input and output threshold settings for the turn on/turn off analysis type.

Returned Format

```
<input_thr>,<output_thr><NL>

<input_thr> ::= percent from 0-100 in NR1 format

<output_thr> ::= percent from 0-100 in NR1 format
```

See Also

- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:ONOFf:TEST" on page 1354
- ":POWer:ONOFf:NEXT" on page 1353
- ":MEASure:ONTime" on page 1233
- ":MEASure:OFFTime" on page 1232

History New in version 11.00.

:POWer:PSRR:APPLy

Command `:POWer:PSRR:APPLy`

The :POWer:PSRR:APPLy command applies the power supply rejection ratio (PSRR) analysis.

The Power Supply Rejection Ratio (PSRR) test is used to determine how well a voltage regulator rejects ripple noise over different frequency range.

This analysis provides a signal from the oscilloscope's waveform generator that sweeps its frequency. This signal is used to inject ripple to the DC voltage that feeds the voltage regulator.

The AC RMS ratio of the input over the output is measured and is plotted over the range of frequencies.

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

- "*ESR? — Event Status Register" on page 226
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:FREQuency:SINGle" on page 1361
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:DATA?

Query `:POWer:PSRR:DATA? [SWEep | SINGle]`

The :POWer:PSRR:DATA? query returns data from the Power Supply Rejection Ratio (PSRR) power analysis.

The comma-separated value format is suitable for spreadsheet analysis.

The SWEep or SINGle option specifies whether to get the data from a sweep or single-frequency analysis (see :POWer:PSRR:FREQuency:MODE). If this option is not specified, the data from the sweep analysis is returned by default.

Returned Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:FREQuency:SINGle" on page 1361
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:FREQuency:MAXimum

Command

```
:POWer:PSRR:FREQuency:MAXimum <value>[suffix]
```

```
<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 20000000}
```

```
[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:PSRR:FREQuency:MAXimum command sets the end sweep frequency value. The PSRR measurement is displayed on a log scale, so you can select from decade values in addition to the maximum frequency of 20 MHz.

Query

```
:POWer:PSRR:FREQuency:MAXimum?
```

The :POWer:PSRR:FREQuency:MAXimum query returns the maximum sweep frequency setting.

Returned Format

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

```
<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 20000000}
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:FREQuency:MINimum

Command

```
:POWer:PSRR:FREQuency:MINimum <value>[suffix]

<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:PSRR:FREQuency:MINimum command sets the start sweep frequency value. The measurement is displayed on a log scale, so you can select from decade values.

Query

```
:POWer:PSRR:FREQuency:MINimum?
```

The :POWer:PSRR:FREQuency:MINimum query returns the minimum sweep frequency setting.

Returned Format

```
<value><NL>

<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:FREQuency:MODE

Command

```
:POWer:PSRR:FREQuency:MODE <mode>
```

```
<mode> ::= {SWEep | SINGle}
```

The :POWer:PSRR:FREQuency:MODE command specifies whether the analysis should be performed by sweeping through a range of frequencies (SWEep) or at a single frequency (SINGle).

The SINGle mode is useful for evaluating amplitudes at a single frequency. After running the test at a single frequency, you can manually adjust (increase) the waveform generator's amplitude until you begin to observe distortion in the waveforms on the oscilloscope's display. You can then use that amplitude at all frequencies in SWEep mode, or you can evaluate amplitudes at other frequencies in order to determine an optimized amplitude profile (see :POWer:PSRR:WGEN:VOLTage:PROFile).

Query

```
:POWer:PSRR:FREQuency:MODE?
```

The :POWer:PSRR:FREQuency:MODE? query returns the frequency mode setting.

Returned Format

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

```
<mode> ::= {SWE | SING}
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:SINGle" on page 1361
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History

New in version 11.00.

:POWer:PSRR:FREQuency:SINGle

Command

```
:POWer:PSRR:FREQuency:SINGle <value>[suffix]
```

```
<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 2000000}
```

```
[suffix] ::= {Hz | kHz| MHz}
```

The :POWer:PSRR:FREQuency:SINGle command sets the single frequency value. The measurement is displayed on a log scale, so you can select from decade values.

Query

```
:POWer:PSRR:FREQuency:SINGle?
```

The :POWer:PSRR:FREQuency:SINGle query returns the single frequency setting.

Returned Format

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

```
<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
    | 2000000}
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History

New in version 11.00.

:POWer:PSRR:SOURce:INPut

Command

```
:POWer:PSRR:SOURce:INPut <source>
```

```
<source> ::= CHANnel<n>
```

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

The :POWer:PSRR:SOURce:INPut command selects the oscilloscope channel that is probing the power supply input.

Query

```
:POWer:PSRR:SOURce:INPut?
```

The :POWer:PSRR:SOURce:INPut? query returns the channel selection.

Returned Format

```
<source><NL>
```

```
<source> ::= CHAN<n>
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History

New in version 11.00.

:POWer:PSRR:SOURce:OUTPut

Command

```
:POWer:PSRR:SOURce:OUTPut <source>
```

```
<source> ::= CHANnel<n>
```

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

The :POWer:PSRR:SOURce:OUTPut command selects the oscilloscope channel that is probing the power supply output.

Query

```
:POWer:PSRR:SOURce:OUTPut?
```

The :POWer:PSRR:SOURce:OUTPut? query returns the channel selection.

Returned Format

```
<source><NL>
```

```
<source> ::= CHAN<n>
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History

New in version 11.00.

:POWer:PSRR:SWEep:POINts

Command

```
:POWer:PSRR:SWEep:POINts <pts>
```

```
<pts> ::= number of points from 1 to 1000 in NR1 format
```

The :POWer:PSRR:SWEep:POINts command specifies the total number of points in the frequency response analysis.

Query

```
:POWer:PSRR:SWEep:POINts?
```

The :POWer:PSRR:SWEep:POINts? query returns the number of points setting.

Returned Format

```
<pts><NL>
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:WGEN:LOAD

Command

```
:POWer:PSRR:WGEN:LOAD <impedance>

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

The :POWer:PSRR:WGEN:LOAD command sets the waveform generator expected output load impedance.

The output impedance of the Gen Out signal 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

```
:POWer:PSRR:WGEN:LOAD?
```

The :POWer:PSRR:WGEN:LOAD? query returns the waveform generator expected output load impedance setting.

Returned Format

```
<impedance><NL>

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

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:VOLTage" on page 1366
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History

New in version 11.00.

:POWer:PSRR:WGEN:VOLTage

Command

```
:POWer:PSRR:WGEN:VOLTage <amplitude>[,<range>]
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

```
<range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ
    | F10MHZ | F20MHZ}
```

The :POWer:PSRR:WGEN:VOLTage command sets the waveform generator output amplitude(s).

When the waveform generator amplitude profile is enabled (with the :POWer:PSRR:WGEN:VOLTage:PROFile command), you can set an initial ramp amplitude for each frequency range.

Without the <range> parameter, this command sets the waveform generator output amplitude used when the amplitude profile is disabled.

Query

```
:POWer:PSRR:WGEN:VOLTage? [<range>]
```

The :POWer:PSRR:WGEN:VOLTage? query returns the waveform generator output amplitude setting(s).

Returned Format

```
<amplitude><NL>
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 1367

History New in version 11.00.

:POWer:PSRR:WGEN:VOLTage:PROFile

Command

```
:POWer:PSRR:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}}
```

The :POWer:PSRR:WGEN:VOLTage:PROFile command enables or disables the ability to set initial waveform generator ramp amplitudes for each frequency range.

With amplitude profiling, you can use lower amplitudes at frequencies where the device under test (DUT) is sensitive to distortion and use higher amplitudes where the DUT is less sensitive to distortion.

You can often observe distortions during the test. If the input test sine wave begins to look lopsided, clipped, or somewhat triangular in shape (nonsinusoidal), you are probably encountering distortion due to overdriving your DUT. Optimizing test amplitudes to achieve the best dynamic range measurements is often an iterative process of running your frequency response measurements multiple times.

Query

```
:POWer:PSRR:WGEN:VOLTage:PROFile?
```

The :POWer:PSRR:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Returned Format

```
<setting><NL>

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

See Also

- ":POWer:PSRR:APPLy" on page 1356
- ":POWer:PSRR:DATA?" on page 1357
- ":POWer:PSRR:FREQuency:MAXimum" on page 1358
- ":POWer:PSRR:FREQuency:MINimum" on page 1359
- ":POWer:PSRR:FREQuency:MODE" on page 1360
- ":POWer:PSRR:SWEep:POINts" on page 1364
- ":POWer:PSRR:SOURce:INPut" on page 1362
- ":POWer:PSRR:SOURce:OUTPut" on page 1363
- ":POWer:PSRR:WGEN:LOAD" on page 1365
- ":POWer:PSRR:WGEN:VOLTage" on page 1366

History New in version 11.00.

:POWer:QUALity:APPLy

Command `:POWer:QUALity:APPLy`

The :POWer:QUALity:APPLy command applies the power quality analysis.

The power quality analysis shows the quality of the AC input line.

Some AC current may flow back into and back out of the load without delivering energy. This current, called reactive or harmonic current, gives rise to an "apparent" power which is larger than the actual power consumed. Power quality is gauged by these measurements: power factor, apparent power, true power, reactive power, crest factor, and phase angle of the current and voltage of the AC line.

See Also

- ":MEASure:FACTor" on page 1230
- ":MEASure:REAL" on page 1240
- ":MEASure:APParent" on page 1225
- ":MEASure:REACtive" on page 1238
- ":MEASure:CRESt" on page 1227
- ":MEASure:ANGLe" on page 1224

History New in version 11.00.

:POWer:RIPPle:APPLy

Command `:POWer:RIPPle:APPLy`

The :POWer:RIPPle:APPLy command applies the output ripple analysis.

See Also

- ":MEASure:RIPPle" on page 1241

History New in version 11.00.

:POWer:SIGNals:AUTosetup

Command `:POWer:SIGNals:AUTosetup`

The :POWer:SIGNals:AUTosetup command performs automated oscilloscope setup for the signals in the specified type of power analysis (see :POWer:ANALysis).

See Also

- ":POWer:ANALysis" on page 1314

History New in version 11.00.

:POWer:SIGNals:CYCLes:HARMonics

Command

```
:POWer:SIGNals:CYCLes:HARMonics <count>
```

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

```
Legal values are 1 to 100.
```

The :POWer:SIGNals:CYCLes:HARMonics command specifies the number of cycles to include in the current harmonics analysis.

Query

```
:POWer:SIGNals:CYCLes:HARMonics?
```

The :POWer:SIGNals:CYCLes:HARMonics query returns the number of cycles currently set.

Returned Format

```
<count><NL>
```

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

See Also

- ":POWer:HARMonics:DISPlay" on page 1336
- ":POWer:HARMonics:APPLy" on page 1334
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:CYCLes:QUALity

Command

```
:POWer:SIGNals:CYCLes:QUALity <count>
```

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

```
Legal values are 1 to 100.
```

The :POWer:SIGNals:CYCLes:QUALity command specifies the number of cycles to include in the power quality analysis.

Query

```
:POWer:SIGNals:CYCLes:QUALity?
```

The :POWer:SIGNals:CYCLes:QUALity query returns the number of cycles currently set.

Returned Format

```
<count><NL>
```

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

See Also

- ":POWer:QUALity:APPLy" on page 1368
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:DURation:EFFiciency

Command

```
:POWer:SIGNals:DURation:EFFiciency <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:EFFiciency command specifies the duration of the efficiency analysis.

Query

```
:POWer:SIGNals:DURation:EFFiciency?
```

The :POWer:SIGNals:DURation:EFFiciency query returns the set duration time value.

Returned Format

```
<value><NL>

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

See Also

- ":POWer:EFFiciency:APPLy" on page 1331
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:DURation:MODulation

Command

```
:POWer:SIGNals:DURation:MODulation <value>[suffix]
```

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

```
[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:MODulation command specifies the duration of the modulation analysis.

Query

```
:POWer:SIGNals:DURation:MODulation?
```

The :POWer:SIGNals:DURation:MODulation query returns the set duration time value.

Returned Format

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

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

See Also

- ":POWer:MODulation:APPLy" on page 1349
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:DURation:ONOFf:OFF

Command

```
:POWer:SIGNals:DURation:ONOFf:OFF <value>[suffix]
```

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

```
[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:ONOFf:OFF command specifies the duration of the turn off analysis.

Query

```
:POWer:SIGNals:DURation:ONOFf:OFF?
```

The :POWer:SIGNals:DURation:ONOFf:OFF query returns the set duration time value.

Returned Format

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

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

See Also

- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 1383
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:DURation:ONOFf:ON

Command

```
:POWer:SIGNals:DURation:ONOFf:ON <value>[suffix]
```

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

```
[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:ONOFf:ON command specifies the duration of the turn on analysis.

Query

```
:POWer:SIGNals:DURation:ONOFf:ON?
```

The :POWer:SIGNals:DURation:ONOFf:ON query returns the set duration time value.

Returned Format

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

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

See Also

- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 1384
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:DURation:RIPPle

Command

```
:POWer:SIGNals:DURation:RIPPle <value>[suffix]
```

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

```
[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:RIPPle command specifies the duration of the output ripple analysis.

Query

```
:POWer:SIGNals:DURation:RIPPle?
```

The :POWer:SIGNals:DURation:RIPPle query returns the set duration time value.

Returned Format

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

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

See Also

- ":POWer:RIPPle:APPLy" on page 1369
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:DURation:TRANsient

Command

```
:POWer:SIGNals:DURation:TRANsient <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation:TRANsient command specifies the duration of the transient response analysis.

Query

```
:POWer:SIGNals:DURation:TRANsient?
```

The :POWer:SIGNals:DURation:TRANsient query returns the set duration time value.

Returned Format

```
<value><NL>

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

See Also

- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VSTeady:TRANsient" on page 1387
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:IEXPected

Command

```
:POWer:SIGNals:IEXPected <value>[suffix]

<value> ::= Expected current value in NR3 format

[suffix] ::= {A | mA}
```

The :POWer:SIGNals:IEXPected command specifies the expected inrush current amplitude. This value is used to set the vertical scale of the channel probing current.

Query

```
:POWer:SIGNals:IEXPected?
```

The :POWer:SIGNals:IEXPected query returns the expected inrush current setting.

Returned Format

```
<value><NL>

<value> ::= Expected current value in NR3 format
```

See Also

- ":POWer:INRush:APPLy" on page 1346
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:INRush" on page 1382
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:OVERshoot

Command

```
:POWer:SIGNals:OVERshoot <value>

<value> ::= overshoot value in NR1 format
```

The :POWer:SIGNals:OVERshoot command specifies the overshoot value of the output voltage. This value is used to determine the settling band value for the transient response and to adjust the vertical scale of the oscilloscope.

Use the :POWer:SIGNals:OVERshoot:UNITs command to specify the units of the overshoot value.

Query

```
:POWer:SIGNals:OVERshoot?
```

The :POWer:SIGNals:OVERshoot query returns the overshoot value setting.

Returned Format

```
<value><NL>

<value> ::= overshoot value in NR1 format
```

See Also

- ":POWer:SIGNals:OVERshoot:UNITs" on page 1381
- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:TRANsient" on page 1378
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:VSTeady:TRANsient" on page 1387
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:OVERshoot:UNITs

Command

```
:POWer:SIGNals:OVERshoot:UNITs <units>
```

The :POWer:SIGNals:OVERshoot:UNITs command specifies the units of the overshoot value.

<units> {PERCent | VOLT}

Query

```
:POWer:SIGNals:OVERshoot:UNITs?
```

The :POWer:SIGNals:OVERshoot:UNITs? query returns the specified overshoot value units.

Returned Format

```
<units><NL>

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

See Also

- ":POWer:SIGNals:OVERshoot" on page 1380

History New in version 11.20.

:POWer:SIGNals:VMAXimum:INRush

Command

```
:POWer:SIGNals:VMAXimum:INRush <value>[suffix]
```

```
<value> ::= Maximum expected input Voltage in NR3 format
```

```
[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VMAXimum:INRush command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for inrush current analysis.

When the :POWer:ITYPe is DC, this command defines the maximum DC input voltage amplitude value. The values can be negative.

When the :POWer:ITYPe is AC, this command defines the maximum peak-to-peak input voltage. Only positive values are allowed.

Query

```
:POWer:SIGNals:VMAXimum:INRush?
```

The :POWer:SIGNals:VMAXimum:INRush query returns the expected maximum input voltage setting.

Returned Format

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

```
<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 1348
- ":POWer:INRush:APPLy" on page 1346
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:VMAXimum:ONOFf:OFF

Command

```
:POWer:SIGNals:VMAXimum:ONOFf:OFF <value>[suffix]
```

```
<value> ::= Maximum expected input Voltage in NR3 format
```

```
[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VMAXimum:ONOFf:OFF command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for turn off analysis.

When the :POWer:ITYPe is DC, this command defines the maximum DC input voltage amplitude value. The values can be negative.

When the :POWer:ITYPe is AC, this command defines the maximum peak-to-peak input voltage. Only positive values are allowed.

Query

```
:POWer:SIGNals:VMAXimum:ONOFf:OFF?
```

The :POWer:SIGNals:VMAXimum:ONOFf:OFF query returns the expected maximum input voltage setting.

Returned Format

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

```
<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 1375
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:VMAXimum:ONOFf:ON

Command

```
:POWer:SIGNals:VMAXimum:ONOFf:ON <value>[suffix]
```

```
<value> ::= Maximum expected input Voltage in NR3 format
```

```
[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VMAXimum:ONOFf:ON command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for turn on analysis.

When the :POWer:ITYPe is DC, this command defines the maximum DC input voltage amplitude value. The values can be negative.

When the :POWer:ITYPe is AC, this command defines the maximum peak-to-peak input voltage. Only positive values are allowed.

Query

```
:POWer:SIGNals:VMAXimum:ONOFf:ON?
```

The :POWer:SIGNals:VMAXimum:ONOFf:ON query returns the expected maximum input voltage setting.

Returned Format

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

```
<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 1348
- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 1376
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History New in version 11.00.

:POWer:SIGNals:VSTeady:ONOFf:OFF

Command

```
:POWer:SIGNals:VSTeady:ONOFf:OFF <value>[suffix]
```

```
<value> ::= Expected steady state output Voltage value in NR3 format
```

```
[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VSTeady:ONOFf:OFF command specifies the expected steady state output DC voltage of the power supply for turn off analysis.

Query

```
:POWer:SIGNals:VSTeady:ONOFf:OFF?
```

The :POWer:SIGNals:VSTeady:ONOFf:OFF query returns the expected steady state voltage setting.

Returned Format

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

```
<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 1375
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 1383
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:VSTeady:ONOFf:ON

Command

```
:POWer:SIGNals:VSTeady:ONOFf:ON <value>[suffix]

<value> ::= Expected steady state output Voltage value in NR3 format

[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VSTeady:ONOFf:ON command specifies the expected steady state output DC voltage of the power supply for turn on analysis.

Query

```
:POWer:SIGNals:VSTeady:ONOFf:ON?
```

The :POWer:SIGNals:VSTeady:ONOFf:ON query returns the expected steady state voltage setting.

Returned Format

```
<value><NL>

<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:ONOFf:APPLy" on page 1352
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 1376
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 1384
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:VSTeady:TRANsient

Command

```
:POWer:SIGNals:VSTeady:TRANsient <value>[suffix]

<value> ::= Expected steady state output Voltage value in NR3 format

[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VSTeady:TRANsient command specifies the expected steady state output DC voltage of the power supply for transient response analysis.

This value is used along with the overshoot percentage to specify the settling band for the transient response and to adjust the vertical scale of the oscilloscope.

Query

```
:POWer:SIGNals:VSTeady:TRANsient?
```

The :POWer:SIGNals:VSTeady:TRANsient query returns the expected steady state voltage setting.

Returned Format

```
<value><NL>

<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:DURation:TRANsient" on page 1378
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:SOURce:CURRent<I>

Command

```
:POWer:SIGNals:SOURce:CURRent<I> <source>
```

```
<I> ::= 1, 2 in NR1 format
```

```
<source> ::= CHANnel<N>
```

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

The :POWer:SIGNals:SOURce:CURRent<I> command specifies the first, and perhaps second, current source channel to be used in the power analysis.

Query

```
:POWer:SIGNals:SOURce:CURRent<I>?
```

The :POWer:SIGNals:SOURce:CURRent<I> query returns the current source channel setting.

Returned Format

```
<source><NL>
```

```
<source> ::= CHANnel<N>
```

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

See Also

- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:CYCLes:HARMonics" on page 1371
- ":POWer:SIGNals:CYCLes:QUALity" on page 1372
- ":POWer:SIGNals:DURation:EFFiciency" on page 1373
- ":POWer:SIGNals:DURation:MODulation" on page 1374
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 1375
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 1376
- ":POWer:SIGNals:DURation:RIPPle" on page 1377
- ":POWer:SIGNals:DURation:TRANsient" on page 1378
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:INRush" on page 1382
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 1383
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 1384
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:SIGNals:VSTeady:TRANsient" on page 1387
- ":POWer:SIGNals:SOURce:VOLTage<I>" on page 1389

History

New in version 11.00.

:POWer:SIGNals:SOURce:VOLTage<I>

Command

```
:POWer:SIGNals:SOURce:VOLTage<I> <source>
```

```
<I> ::= 1, 2 in NR1 format
```

```
<source> ::= CHANnel<N>
```

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

The :POWer:SIGNals:SOURce:VOLTage<I> command specifies the first, and perhaps second, voltage source channel to be used in the power analysis.

Query

```
:POWer:SIGNals:SOURce:VOLTage<I>?
```

The :POWer:SIGNals:SOURce:VOLTage<I> query returns the voltage source channel setting.

Returned Format

```
<source><NL>
```

```
<source> ::= CHANnel<N>
```

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

See Also

- ":POWer:SIGNals:AUTosetup" on page 1370
- ":POWer:SIGNals:CYCLes:HARMonics" on page 1371
- ":POWer:SIGNals:CYCLes:QUALity" on page 1372
- ":POWer:SIGNals:DURation:EFFiciency" on page 1373
- ":POWer:SIGNals:DURation:MODulation" on page 1374
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 1375
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 1376
- ":POWer:SIGNals:DURation:RIPPle" on page 1377
- ":POWer:SIGNals:DURation:TRANsient" on page 1378
- ":POWer:SIGNals:IEXPected" on page 1379
- ":POWer:SIGNals:OVERshoot" on page 1380
- ":POWer:SIGNals:VMAXimum:INRush" on page 1382
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 1383
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 1384
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 1385
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 1386
- ":POWer:SIGNals:VSTeady:TRANsient" on page 1387
- ":POWer:SIGNals:SOURce:CURRent<I>" on page 1388

History

New in version 11.00.

:POWer:SLEW:APPLy

Command `:POWer:SLEW:APPLy`

The :POWer:SLEW:APPLy command applies the slew rate analysis.

See Also

- ":POWer:SLEW:SOURce" on page 1391

History New in version 11.00.

:POWer:SLEW:SOURce

Command

```
:POWer:SLEW:SOURce <source>
```

```
<source> ::= {V | I}
```

The :POWer:SLEW:SOURce command selects either the voltage source or the current source as the source for the slew rate analysis.

Query

```
:POWer:SLEW:SOURce?
```

The :POWer:SLEW:SOURce query returns the selected source for the slew rate analysis.

Returned Format

```
<source><NL>
```

```
<source> ::= {V | I}
```

See Also

- ":POWer:SLEW:APPLy" on page 1390

History

New in version 11.00.

:POWer:SWITch:APPLy

Command `:POWer:SWITch:APPLy`

The :POWer:SWITch:APPLy command applies the switching loss analysis using the conduction calculation method, V reference, and I reference settings.

See Also

- ":POWer:SWITch:CONDuction" on page 1393
- ":POWer:SWITch:IREFerence" on page 1394
- ":POWer:SWITch:RDS" on page 1395
- ":POWer:SWITch:VCE" on page 1396
- ":POWer:SWITch:VREFerence" on page 1397
- ":MEASure:ELOSs" on page 1229
- ":MEASure:PLOSs" on page 1236

History New in version 11.00.

:POWer:SWITch:CONDuction

Command

```
:POWer:SWITch:CONDuction <conduction>

<conduction> ::= {WAVeform | RDS | VCE}
```

The :POWer:SWITch:CONDuction command specifies the conduction calculation method:

- WAVeform — The Power waveform uses the original voltage waveform data, and the calculation is: P = V x I
- RDS — Rds(on) — The Power waveform includes error correction:
  - In the On Zone (where the voltage level is below V Ref) – the Power calculation is: P = Id2 x Rds(on)

    Specify Rds(on) using the :POWer:SWITch:RDS command.
  - In the Off Zone (where the current level is below I Ref) – the Power calculation is: P = 0 Watt.
- VCE — Vce(sat) — The Power waveform includes error correction:
  - In the On Zone (where the voltage level is below V Ref) – the Power calculation is: P = Vce(sat) x Ic

    Specify Vce(sat) using the :POWer:SWITch:VCE command.
  - In the Off Zone (where the current level is below I Ref) – the Power calculation is: P = 0 Watt.

Query

```
:POWer:SWITch:CONDuction?
```

The :POWer:SWITch:CONDuction query returns the conduction calculation method.

Returned Format

```
<conduction><NL>

<conduction> ::= {WAV | RDS | VCE}
```

See Also

- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SWITch:IREFerence" on page 1394
- ":POWer:SWITch:RDS" on page 1395
- ":POWer:SWITch:VCE" on page 1396
- ":POWer:SWITch:VREFerence" on page 1397

History

New in version 11.00.

:POWer:SWITch:IREFerence

Command

```
:POWer:SWITch:IREFerence <percent>
```

```
<percent> ::= percent in NR1 format
```

The :POWer:SWITch:IREFerence command to specify the current switching level for the start of switching edges. The value is in percentage of the maximum switch current.

You can adjust this value to ignore noise floors or null offset that is difficult to eliminate in current probes.

This value specifies the threshold that is used to determine the switching edges.

Query

```
:POWer:SWITch:IREFerence?
```

The :POWer:SWITch:IREFerence query returns the current switching level percent value.

Returned Format

```
<percent><NL>
```

```
<percent> ::= percent in NR1 format
```

See Also

- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SWITch:CONDuction" on page 1393
- ":POWer:SWITch:RDS" on page 1395
- ":POWer:SWITch:VCE" on page 1396
- ":POWer:SWITch:VREFerence" on page 1397

History New in version 11.00.

:POWer:SWITch:RDS

Command

```
:POWer:SWITch:RDS <value>[suffix]
```

```
<value> ::= Rds(on) value in NR3 format
```

```
[suffix] ::= {OHM | mOHM}
```

The :POWer:SWITch:RDS command specifies the Rds(on) value when the RDS conduction calculation method is chosen (by :POWer:SWITch:CONDuction).

Query

```
:POWer:SWITch:RDS?
```

The :POWer:SWITch:RDS query returns the Rds(on) value.

Returned Format

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

```
<value> ::= Rds(on) value in NR3 format
```

See Also

- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SWITch:CONDuction" on page 1393
- ":POWer:SWITch:IREFerence" on page 1394
- ":POWer:SWITch:VCE" on page 1396
- ":POWer:SWITch:VREFerence" on page 1397

History New in version 11.00.

:POWer:SWITch:VCE

Command

```
:POWer:SWITch:VCE <value>[suffix]
```

```
<value> ::= Vce(sat) value in NR3 format
```

```
[suffix] ::= {V | mV}
```

The :POWer:SWITch:VCE command specifies the Vce(sat) value when the VCE conduction calculation method is chosen (by :POWer:SWITch:CONDuction).

Query

```
:POWer:SWITch:VCE?
```

The :POWer:SWITch:VCE query returns the Vce(sat) value.

Returned Format

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

```
<value> ::= Vce(sat) value in NR3 format
```

See Also

- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SWITch:CONDuction" on page 1393
- ":POWer:SWITch:IREFerence" on page 1394
- ":POWer:SWITch:RDS" on page 1395
- ":POWer:SWITch:VREFerence" on page 1397

History New in version 11.00.

:POWer:SWITch:VREFerence

Command

```
:POWer:SWITch:VREFerence <percent>
```

```
<percent> ::= percent in NR1 format
```

The :POWer:SWITch:VREFerence command to specify the voltage switching level for the switching edges. The value is in percentage of the maximum switch voltage.

You can adjust this value to ignore noise floors.

This value specifies the threshold that is used to determine the switching edges.

Query

```
:POWer:SWITch:VREFerence?
```

The :POWer:SWITch:VREFerence query returns the voltage switching level percent value.

Returned Format

```
<percent><NL>
```

```
<percent> ::= percent in NR1 format
```

See Also

- ":POWer:SWITch:APPLy" on page 1392
- ":POWer:SWITch:CONDuction" on page 1393
- ":POWer:SWITch:IREFerence" on page 1394
- ":POWer:SWITch:RDS" on page 1395
- ":POWer:SWITch:VCE" on page 1396

History New in version 11.00.

:POWer:TRANsient:APPLy

Command `:POWer:TRANsient:APPLy`

The :POWer:TRANsient:APPLy command applies the transient analysis using the initial current and new current settings.

See Also

- ":POWer:TRANsient:IINitial" on page 1399
- ":POWer:TRANsient:INEW" on page 1400
- ":POWer:TRANsient:NEXT" on page 1401
- ":MEASure:TRESponse" on page 1242

History New in version 11.00.

:POWer:TRANsient:IINitial

Command

```
:POWer:TRANsient:IINitial <value>[suffix]
```

```
<value> ::= Initial current value in NR3 format
```

```
[suffix] ::= {A | mA}
```

The :POWer:TRANsient:IINitial command to specify the initial load current value. The initial load current will be used as a reference and to trigger the oscilloscope.

Query

```
:POWer:TRANsient:IINitial?
```

The :POWer:TRANsient:IINitial query returns the initial load current value.

Returned Format

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

```
<value> ::= Initial current value in NR3 format
```

See Also

- ":POWer:SIGNals:VSTeady:TRANsient" on page 1387
- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:TRANsient:INEW" on page 1400
- ":POWer:TRANsient:NEXT" on page 1401

History New in version 11.00.

:POWer:TRANsient:INEW

Command

```
:POWer:TRANsient:INEW <value>[suffix]
```

```
<value> ::= New current value in NR3 format
```

```
[suffix] ::= {A | mA}
```

The :POWer:TRANsient:INEW command to specify the new load current value. The new load current will be used as a reference and to trigger the oscilloscope.

Query

```
:POWer:TRANsient:INEW?
```

The :POWer:TRANsient:INEW query returns the new load current value.

Returned Format

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

```
<value> ::= New current value in NR3 format
```

See Also

- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:TRANsient:IINitial" on page 1399
- ":POWer:TRANsient:NEXT" on page 1401

History New in version 11.00.

:POWer:TRANsient:NEXT

Command `:POWer:TRANsient:NEXT`

The :POWer:TRANsient:NEXT command goes to the next step of the transient analysis.

This command is equivalent to pressing the **Next** softkey on the oscilloscope front panel when prompted during the analysis.

See Also

- ":POWer:TRANsient:APPLy" on page 1398
- ":POWer:TRANsient:IINitial" on page 1399
- ":POWer:TRANsient:INEW" on page 1400

History New in version 11.00.

39 :REPository Commands

The :REPository commands and queries let you upload measurement data results to Keysight data analytics web service repositories (either the N8844A or the KS6800A Series) where the measurement data is stored and can be analyzed using the data analytics web server tools.

:REPository:CANCel

Command `:REPository:CANCel`

The :REPository:CANCel command cancels the repository action currently in progress.

See Also

- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:CONNect

Command `:REPository:CONNect`

The :REPository:CONNect command connects to the data analytics repository server specified by :REPository:SERVer:NAME.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:CONNect:STATe?

Query `:REPository:CONNect:STATe?`

The :REPository:CONNect:STATe? query returns the state of the connection to the data analytics repository server.

Returned Format `<state><NL>`

`<state> ::= {CONN | NCON}`

- CONN — connected.
- NCON — not connected.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:DISConnect

Command `:REPository:DISConnect`

When the data analytics repository server is connected (see :REPository:CONNect), the :REPository:DISConnect command disconnects from the data analytics repository server.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:DSET

Command `:REPository:DSET <dataset-name>`

The :REPository:DSET command specifies the dataset name for publishing. If the dataset name does not already exist on the repository server, data will be published to this new dataset name.

<dataset-name> Quoted ASCII string.

Query `:REPository:DSET?`

The :REPository:DSET? query returns the currently selected dataset name

Returned Format `<dataset-name><NL>`

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:DUT:MODel

Command

```
:REPository:DUT:MODel <model-number>
```

The :REPository:DUT:MODel command specifies the model number of the device under test (DUT). This information is saved as metadata along with measurement data that is published to the data analytics repository server.

<model-number> Quoted ASCII string.

Query

```
:REPository:DUT:MODel?
```

The :REPository:DUT:MODel? query returns the currently specified DUT model number.

Returned Format

```
<model-number><NL>
```

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:DUT:SERial

Command `:REPository:DUT:SERial <model-number>`

The :REPository:DUT:SERial command specifies the serial number of the device under test (DUT). This information is saved as metadata along with measurement data that is published to the data analytics repository server.

<serial-number> Quoted ASCII string.

Query `:REPository:DUT:SERial?`

The :REPository:DUT:SERial? query returns the currently specified DUT serial number.

Returned Format `<serial-number><NL>`

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:PMODe

Command

```
:REPository:MEASure:PMODe <publish-mode>
```

The :REPository:MEASure:PMODe command selects the measurement publish mode.

<publish-mode> {SELected | CUSTom}

- SELected — In this mode, you are only able to select the measurements you want to publish. The names of the measurements are the ones given in the oscilloscope, and you are not able to specify upper and lower measurement limits.
- CUSTom — In the custom mode, you can specify custom measurement names and upper and lower measurement limits. See :REPository:MEASure:PUBLish<N>:NAME, :REPository:MEASure:PUBLish<N>:ULIMit, and :REPository:MEASure:PUBLish<N>:LLIMit.

Query

```
:REPository:MEASure:PMODe?
```

The :REPository:MEASure:PMODe? query returns the selected measurement publish mode.

Returned Format

```
<publish-mode><NL>
```

```
<publish-mode> ::= {SEL | CUST}
```

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:PUBLish<N>:LLIMit

Command

```
:REPository:MEASure:PUBLish<N>:LLIMit <lower-limit>
```

When the measurement publish mode is CUSTom (see :REPository:MEASure:PMODe), the :REPository:MEASure:PUBLish<N>:LLIMit command sets a lower limit for the specified measurement.

This lower limit is used by the data analytics repository server in categorizing the measurement results.

<N> An integer, 1-20. This number represents the position of the measurement on-screen in the Measurements results window.

<lower-limit> A float value in NR3 format.

Query

```
:REPository:MEASure:PUBLish<N>:LLIMit?
```

The :REPository:MEASure:PUBLish<N>:LLIMit? query returns the lower limit for the specified measurement.

Returned Format

```
<lower-limit><NL>
```

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:PUBLish<N>:NAME

Command `:REPository:MEASure:PUBLish<N>:NAME <string>`

When the measurement publish mode is CUSTom (see :REPository:MEASure:PMODe), the :REPository:MEASure:PUBLish<N>:NAME command sets a custom name for the specified measurement.

<N> An integer, 1-20. This number represents the position of the measurement on-screen in the Measurements results window.

<string> A quoted ASCII string.

Query `:REPository:MEASure:PUBLish<N>:NAME?`

The :REPository:MEASure:PUBLish<N>:NAME? query returns the custom name for the specified measurement.

Returned Format `<string><NL>`

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:PUBLish<N>:STATe

Command

```
:REPository:MEASure:PUBLish<N>:STATe {{0 | OFF} | {1 | ON}}
```

The :REPository:MEASure:PUBLish<N>:STATe command selects or clears the specified measurement for publishing to the data analytics repository server.

To select or clear all active measurements, see :REPository:MEASure:SELection:ALL or :REPository:MEASure:SELection:CLEar.

To select a group of active measurements, see :REPository:MEASure:SELection.

<N> An integer, 1-20. This number represents the position of the measurement on-screen in the Measurements results window.

Query

```
:REPository:MEASure:PUBLish<N>:STATe?
```

The :REPository:MEASure:PUBLish<N>:STATe? query returns whether the specified measurement is selected or cleared for publishing to the data analytics repository server.

Returned Format

```
<setting><NL>

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

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:PUBLish<N>:ULIMit

Command
```
:REPository:MEASure:PUBLish<N>:ULIMit <upper-limit>
```

When the measurement publish mode is CUSTom (see :REPository:MEASure:PMODe), the :REPository:MEASure:PUBLish<N>:ULIMit command sets an upper limit for the specified measurement.

This upper limit is used by the data analytics repository server in categorizing the measurement results.

<N> An integer, 1-20. This number represents the position of the measurement on-screen in the Measurements results window.

<upper-limit> A float value in NR3 format.

Query
```
:REPository:MEASure:PUBLish<N>:ULIMit?
```

The :REPository:MEASure:PUBLish<N>:ULIMit? query returns the upper limit for the specified measurement.

Returned Format
```
<upper-limit><NL>
```

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:SELection

Command `:REPository:MEASure:SELection <indices>`

The :REPository:MEASure:SELection command selects a group of measurements for publishing to the data analytics repository server.Measurements are selected by their indices which are the numbers displayed in the front-panel graphical user interface's Measurements results window.

To select or clear all active measurements, see :REPository:MEASure:SELection:ALL or :REPository:MEASure:SELection:CLEar.

To select or clear individual active measurements, see :REPository:MEASure:PUBLish<N>:STATe.

<indices> A quoted, comma-separated list of measurement indices.

Query `:REPository:MEASure:SELection?`

The :REPository:MEASure:SELection? query returns a comma-separated list of indices of the measurements that are selected for publishing to the data analytics repository server.

Returned Format `<indices><NL>`

NOTE **The comma-separated list of measurement indices is quoted when sending the command; however, the query returns the comma-separated list of indices without quotes.**

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421

- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:SELection:ALL

Command `:REPository:MEASure:SELection:ALL`

The :REPository:MEASure:SELection:ALL command selects all active measurements for publishing to the data analytics repository server.

To select a group of active measurements, see :REPository:MEASure:SELection.

To select individual active measurements, see :REPository:MEASure:PUBLish<N>:STATe.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:MEASure:SELection:CLEar

Command `:REPository:MEASure:SELection:CLEar`

The :REPository:MEASure:SELection:CLEar command de-selects all active measurements for publishing to the data analytics repository server.

To clear individual active measurements, see :REPository:MEASure:PUBLish<N>:STATe.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:PUBLish

Command `:REPository:PUBLish`

When the data analytics repository server is connected (see :REPository:CONNect), the :REPository:PUBLish command publishes the selected measurements to the server.

To select all active measurements, see :REPository:MEASure:SELection:ALL.

To select a group of active measurements, see :REPository:MEASure:SELection.

To select individual active measurements, see :REPository:MEASure:PUBLish<N>:STATe.

To cancel the repository action currently in progress, see :REPository:CANCel.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:SERVer:NAME" on page 1422
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:SERVer:NAME

Command `:REPository:SERVer:NAME <server-URL>`

The :REPository:SERVer:NAME command specifies the URL of the data analytics repository server.

<server-URL> A quoted string of the data analytics server's URL. The URL typically contains a port number as well, for example, like: "http://wcoscygapp1.cos.is.keysight.com:5000".

Query `:REPository:SERVer:NAME?`

The :REPository:SERVer:NAME? query returns the currently specified data analytics repository server URL.

Returned Format `<server-URL><NL>`

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:VERSion?" on page 1423

History New in version 11.20.

:REPository:SERVer:VERSion?

Query `:REPository:SERVer:VERSion?`

When the data analytics repository server is connected (see :REPository:CONNect), the :REPository:SERVer:VERSion? query returns the version of the software running on the server.

Returned Format `<version-string><NL>`

<version-string> A quoted ASCII string.

See Also

- ":REPository:CANCel" on page 1404
- ":REPository:CONNect" on page 1405
- ":REPository:CONNect:STATe?" on page 1406
- ":REPository:DISConnect" on page 1407
- ":REPository:DSET" on page 1408
- ":REPository:DUT:MODel" on page 1409
- ":REPository:DUT:SERial" on page 1410
- ":REPository:MEASure:PMODe" on page 1411
- ":REPository:MEASure:PUBLish<N>:LLIMit" on page 1413
- ":REPository:MEASure:PUBLish<N>:NAME" on page 1414
- ":REPository:MEASure:PUBLish<N>:STATe" on page 1415
- ":REPository:MEASure:PUBLish<N>:ULIMit" on page 1416
- ":REPository:MEASure:SELection" on page 1417
- ":REPository:MEASure:SELection:ALL" on page 1419
- ":REPository:MEASure:SELection:CLEar" on page 1420
- ":REPository:PUBLish" on page 1421
- ":REPository:SERVer:NAME" on page 1422

History New in version 11.20.

40 :SBUS<N> (Serial Bus) Commands

The :SBUS<N> subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE **These commands are only valid when the corresponding serial decode option has been licensed.**

General :SBUS<N> Commands

:SBUS<N>[:DISPlay]

Command

```
: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.

Query

```
:SBUS<N>[:DISPlay]?
```

The :SBUS<N>[:DISPlay]? query returns the current display setting of the serial decode bus.

Returned Format

```
[:SBUS<N>[:DISPlay]] <display><NL>
```

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

See Also

- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:MODE

Command

```
:SBUS<N>:MODE <mode>
```

```
<mode> ::= {CAN | IIC | SPI | FLEXray | LIN | GENRaw | 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

```
:SBUS<N>:MODE?
```

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Returned Format

```
[:SBUS<N>:MODE] <mode><NL>
```

```
<mode> ::= {CAN | IIC | SPI | FLEX | LIN | GENR | UART}
```

See Also

- ":SBUS<N>:CAN Commands" on page 1431
- ":SBUS<N>:FLEXray Commands" on page 1447
- ":SBUS<N>:GENRaw Commands" on page 1457
- ":SBUS<N>:IIC Commands" on page 1465
- ":SBUS<N>:LIN Commands" on page 1474
- ":SBUS<N>:SPI Commands" on page 1483
- ":SBUS<N>:UART Commands" on page 1503

History

New in version 3.50.

Version 4.60: Added CAN mode option.

Version 5.20: Added the FLEXray and LIN mode options.

Version 6.20: Added the GENRaw mode option.

Version 11.10: Added the UART mode option.

:SBUS<N>:SEARch:ENABle

Command

```
:SBUS<N>:SEARch:ENABle {{0 | OFF} | {1 | ON}}
```

The :SBUS<N>:SEARch:ENABle command enables or disables protocol search within acquired data.

Query

```
:SBUS<N>:SEARch:ENABle?
```

The :SBUS<N>:SEARch:ENABle? query returns the protocol search setting.

Returned Format

```
<setting><NL>
```

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

See Also

- ":SBUS<N>:SEARch:TRIGger" on page 1430

History New in version 6.60.

:SBUS<N>:SEARch:TRIGger

Command

```
:SBUS<N>:SEARch:TRIGger {{0 | OFF} | {1 | ON}}
```

The :SBUS<N>:SEARch:TRIGger command enables or disables protocol search being used as a software trigger.

NOTE When enabled, the trigger (time = 0) position is set to the center of the first packet that is found. The trigger position is not at the actual hardware trigger point.

For more information on how the :SBUS<N>:SEARch:ENABle and :SBUS<N>:SEARch:TRIGger controls work, see the online help.

Query

```
:SBUS<N>:SEARch:TRIGger?
```

The :SBUS<N>:SEARch:TRIGger? query returns the protocol search being used as a software trigger setting.

Returned Format

```
<setting><NL>
```

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

See Also

- ":SBUS<N>:SEARch:ENABle" on page 1429

History New in version 6.60.

:SBUS<N>:CAN Commands

NOTE These commands are only valid when the automotive CAN serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:CAN:FDSPoint

Command

```
:SBUS<N>:CAN:FDSPoint <value>
```

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 beginning and end of the bit time.

<value> Percentage values in the range of 30 and 90 in NR3 format. Values are rounded off to nearest 0.5 resolution.

Query

```
:SBUS<n>:CAN:FDSPoint?
```

The :SBUS<N>:CAN:FDSPoint? query returns the set sample point percentage value

Returned Format

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

See Also

- ":SBUS<N>:CAN:SIGNal:FDBaudrate" on page 1436
- ":SBUS<N>:CAN:TYPE" on page 1446

History New in version 5.60.

:SBUS<N>:CAN:SAMPlepoint

Command

```
:SBUS<N>:CAN:SAMPlepoint <value>

<value> ::= {60 | 62.5 | 65 | 67.5 | 68 | 70 | 72.5 | 75 | 77.5
    | 80 | 82.5 | 85 | 87.5} 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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:CAN:SAMPlepoint?
```

The :SBUS<N>:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Returned Format

```
<value><NL>

<value> ::= {60 | 62.5 | 65 | 67.5 | 68 | 70 | 72.5 | 75 | 77.5
    | 80 | 82.5 | 85 | 87.5} in NR3 format
```

See Also

- ":SBUS<N>:MODE" on page 1428

History New in version 4.60.

:SBUS<N>:CAN:SIGNal:BAUDrate

Command

```
:SBUS<N>:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= a real number from 10E3 to 5E6
```

The :SBUS<N>:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 5 Mb/s.

If the baud rate you select does not match the system baud rate, false triggers may occur.

<N> An integer, 1-4.

Query

```
:SBUS<N>:CAN:SIGNal:BAUDrate?
```

The :SBUS<N>:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Returned Format

```
<baudrate><NL>

<baudrate> ::= a real number from 10E3 to 5E6
```

See Also

- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 1435
- ":SBUS<N>:CAN:SOURce" on page 1437

History New in version 4.60.

:SBUS<N>:CAN:SIGNal:DEFinition

Command

```
:SBUS<N>:CAN:SIGNal:DEFinition <value>

<value> ::= {CANH | CANL | DIFFerential | DIFL}
```

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.

Dominant low signals:

- CANL — the actual CAN_L differential bus signal.
- 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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:CAN:SIGNal:DEFinition?
```

The :SBUS<N>:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Returned Format

```
<value><NL>

<value> ::= {CANH | CANL | DIFL}
```

See Also

- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:SIGNal:BAUDrate" on page 1434
- ":SBUS<N>:CAN:SOURce" on page 1437

History New in version 4.60.

:SBUS<N>:CAN:SIGNal:FDBaudrate

Command

```
:SBUS<N>:CAN:SIGNal:FDBaudrate <baudrate>
```

The :SBUS<N>:CAN:SIGNal:FDBaudrate command sets the CAN FD baud rate in the range from 1 Mb/s to 10 Mb/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.

<baudrate> A real number from 1E6 to 10E6

Query

```
:SBUS<n>:CAN:SIGNal:FDBaudrate?
```

The :SBUS<N>:CAN:SIGNal:FDBaudrate? query returns the CAN FD baud rate setting.

Returned Format

```
<baudrate><NL>
```

See Also

- ":SBUS<N>:CAN:SIGNal:BAUDrate" on page 1434
- ":SBUS<N>:CAN:FDSPoint" on page 1432
- ":SBUS<N>:CAN:TYPE" on page 1446

History New in version 5.60.

:SBUS<N>:CAN:SOURce

Command

```
:SBUS<N>:CAN:SOURce <source>
```

```
<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:CAN:SOURce command sets the source for the CAN signal.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:CAN:SOURce?
```

The :SBUS<N>:CAN:SOURce? query returns the current source for the CAN signal.

Returned Format

```
<source><NL>
```

See Also

- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 1435

History New in version 4.60.

Version 5.20: The NONE parameter was added.

:SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:CAN:TRIGger <condition>

<condition> ::= {SOF | DATA | ERRor | IDData | IDRemote | ALLerrors
   | OVERload | ACKerror}

<condition> ::= {FDData | IDFDdata | EBActive | EBPassive | CRCDbit
   | BRSBit | STUFferror | FORMerror | CRCerror | EOF} for CAN FD
```

NOTE

You must set the proper :SBUS<N>:MODE, :SBUS<N>:CAN:TRIGger, and :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE before setting any other trigger values.

The :SBUS<N>:CAN:TRIGger command sets the CAN "trigger on" condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified ID, Data, and the DLC (Data length code).
- ERRor — will trigger on CAN Error frame.
- IDData — will trigger on CAN frames matching the specified ID of a Data frame.
- IDRemote — will trigger on CAN frames matching the specified ID of a Remote frame.
- ALLerrors — will trigger on CAN active error frames and unknown bus conditions.
- OVERload — will trigger on CAN overload frames.
- ACKerror — will trigger on a data or remote frame acknowledge bit that is recessive.

When :SBUS<N>:CAN:TYPE is set to CANFd, the previous IDData condition is no longer valid, and the following "trigger on" conditions are valid:

- FDData — will trigger on CAN FD Data frames matching the specified ID, Data, and the DLC (Data length code).
- IDFDdata — will trigger on CAN FD frames matching the specified ID of a Data frame.
- EBActive — will trigger on ESI bit if set to active.
- EBPassive — will trigger on ESI bit if set to passive.
- CRCDbit — will trigger on CRC delimiter bit.
- BRSBit — will trigger on BRS bit.
- STUFferror — will trigger on stuff error.
- FORMerror — will trigger on form error.
- CRCerror — will trigger on CRC error.

- EOF — will trigger on EOF.

NOTE

CAN FD triggering is supported on Infiniium S-Series oscilloscopes only.

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.

<N> An integer, 1-4.

Example To enable the SBUS1 trigger, set the trigger to CAN Data frame matching the specified Standard ID 0x1f, Data 0x7fffff, and DLC 3.

```
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:MODE CAN"
myScope.WriteString ":SBUS1:CAN:TRIGger DATA"
myScope.WriteString ":SBUS1:CAN:TRIGger:PATTern:ID:MODE STANdard"
myScope.WriteString ":SBUS1:CAN:TRIGger:PATTern:ID '0x1f'"
myScope.WriteString ":SBUS1:CAN:TRIGger:PATTern:DATA:LENGth 3"
myScope.WriteString ":SBUS1:CAN:TRIGger:PATTern:DATA '0x7fffff'"
```

Query

```
:SBUS<N>:CAN:TRIGger?
```

The :SBUS<N>:CAN:TRIGger? query returns the current CAN trigger on condition.

Returned Format

```
<condition><NL>

<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}

<condition> ::= {FDD | IDFD | EBA | EBP | CRCD| BRSB | STUF | FORM
   | CRC | EOF} for CAN FD
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:TYPE" on page 1446
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1441
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth (9000 Series, 9000H Series, S-Series)" on page 1442
- ":SBUS<N>:CAN:TRIGger:PATTern:ID (9000 Series, 9000H Series, S-Series)" on page 1444
- ":SBUS<N>:CAN:TRIGger:PATTern:ID:MODE (9000 Series, 9000H Series, S-Series)" on page 1445

- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 1435
- ":SBUS<N>:CAN:SOURce" on page 1437

History New in version 4.60.

Version 5.60: Additional trigger condition options for CAN FD have been added.

:SBUS<N>:CAN:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)

Command

```
: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 | $}
```

NOTE You must set the proper :SBUS<N>:MODE, :SBUS<N>:CAN:TRIGger, and :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE before setting any other trigger values.

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.

<N> An integer, 1-4.

Query

```
: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.

Returned Format

```
<string><NL> in nondecimal format
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)" on page 1438
- ":SBUS<N>:CAN:TRIGger:PATTern:ID:MODE (9000 Series, 9000H Series, S-Series)" on page 1445
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth (9000 Series, 9000H Series, S-Series)" on page 1442
- ":SBUS<N>:CAN:TRIGger:PATTern:ID (9000 Series, 9000H Series, S-Series)" on page 1444

History New in version 4.60.

:SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format for standard CAN

<length> ::= integer from 1 to 15 in NR1 format for CAN FD
```

NOTE You must set the proper :SBUS<N>:MODE, :SBUS<N>:CAN:TRIGger, and :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE before setting any other trigger values.

The :SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string.

When ":SBUS<N>:CAN:TYPE CANStandard" is selected, the number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits).

When ":SBUS<N>:CAN:TYPE CANFd" is selected, the number of bytes in the string can be anywhere from 1 bytes to 15 bytes (120 bits).

NOTE CAN FD triggering is supported on Infiniium S-Series oscilloscopes only.

The value for these bytes is set by the :SBUS<N>:CAN:TRIGger:PATTern:DATA command.

<N> An integer, 1-4.

Query

```
:SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth? query returns the current CAN data pattern length setting.

Returned Format

```
<length><NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:TYPE" on page 1446
- ":SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)" on page 1438
- ":SBUS<N>:CAN:TRIGger:PATTern:ID:MODE (9000 Series, 9000H Series, S-Series)" on page 1445
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1441
- ":SBUS<N>:CAN:SOURce" on page 1437

History New in version 4.60.

Version 5.60: Length can be from 1 to 15 bytes for CAN FD.

:SBUS<N>:CAN:TRIGger:PATTern:ID (9000 Series, 9000H Series, S-Series)

Command

```
: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 | $}
```

NOTE You must set the proper :SBUS<N>:MODE, :SBUS<N>:CAN:TRIGger, and :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE before setting this value; otherwise, this value is defaulted to "don't care" when the mode is changed.

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 size changes based on the :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE setting; it is 11 bits when the mode is STANdard, and it is 29 bits when the mode is EXTended.

A string longer than 29 bits is truncated to 29 bits when setting the ID pattern resource.

<N> An integer, 1-4.

Query

```
: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.

Returned Format

```
<string><NL> in 29-bit binary string format
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)" on page 1438
- ":SBUS<N>:CAN:TRIGger:PATTern:ID:MODE (9000 Series, 9000H Series, S-Series)" on page 1445
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1441

History New in version 4.60.

:SBUS<N>:CAN:TRIGger:PATTern:ID:MODE (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:CAN:TRIGger:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

NOTE You must set the proper :SBUS<N>:MODE, :SBUS<N>:CAN:TRIGger, and :SBUS<N>:CAN:TRIGger:PATTern:ID:MODE before setting any other trigger values.

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.

<N> An integer, 1-4.

Query

```
: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.

Returned Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)" on page 1438
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1441
- ":SBUS<N>:CAN:TRIGger:PATTern:DATA:LENGth (9000 Series, 9000H Series, S-Series)" on page 1442
- ":SBUS<N>:CAN:TRIGger:PATTern:ID (9000 Series, 9000H Series, S-Series)" on page 1444

History New in version 4.60.

:SBUS<N>:CAN:TYPE

Command

```
:SBUS<N>:CAN:TYPE <cantype>

<cantype> ::= {CANStandard | CANFd}
```

The :SBUS<N>:CAN:TYPE command selects between standard CAN or Flexible Data Rate CAN (CAN FD) decode types.

Query

```
:SBUS<n>:CAN:TYPE?
```

The :SBUS<N>:CAN:TYPE? query returns the selected decode type.

Returned Format

```
<cantype><NL>

<cantype> ::= {CANS | CANF}
```

See Also

- ":SBUS<N>:CAN:FDSPoint" on page 1432
- ":SBUS<N>:CAN:SIGNal:FDBaudrate" on page 1436
- ":SBUS<N>:CAN:TRIGger (9000 Series, 9000H Series, S-Series)" on page 1438

History New in version 5.60.

:SBUS<N>:FLEXray Commands

NOTE These commands are only valid when the automotive FLEXray serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:FLEXray:BAUDrate

Command

```
:SBUS<N>:FLEXray:BAUDrate <baudrate>
```

```
<baudrate> ::= {2500000 | 5000000 | 10000000}
```

The :SBUS<n>:FLEXray:BAUDrate command specifies the baud rate as 2.5 Mb/s, 5 Mb/s, or 10 Mb/s.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:BAUDrate?
```

The :SBUS<n>:FLEXray:BAUDrate? query returns the current baud rate setting.

Returned Format

```
<baudrate><NL>
```

```
<baudrate> ::= {2500000 | 5000000 | 10000000}
```

See Also

- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:CHANnel

Command

```
:SBUS<N>:FLEXray:CHANnel <channel>
```

```
<channel> ::= {A | B}
```

The :SBUS<n>:FLEXray:CHANnel command specifies the bus channel, A or B, of the FlexRay signal.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:CHANnel?
```

The :SBUS<n>:FLEXray:CHANnel? query returns the current bus channel setting.

Returned Format

```
<channel><NL>
```

```
<channel> ::= {A | B}
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:SOURce

Command

```
:SBUS<N>:FLEXray:SOURce <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
    | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:FLEXray:SOURce command sets the source for the FlexRay signal.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:FLEXray:SOURce?
```

The :SBUS<n>:FLEXray:SOURce? query returns the source of the FlexRay signal.

Returned Format

```
<source><NL>
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger

Command

```
:SBUS<N>:FLEXray:TRIGger <condition>

<condition> ::= {FRAMe | ERRor}
```

The :SBUS<n>:FLEXray:TRIGger command sets the FLEXray "trigger on" condition:

- FRAMe — triggers on specified frames (without errors).
- ERRor — triggers on selected active error frames and unknown bus conditions.

NOTE The FlexRay trigger is a software-based protocol trigger which operates on data that has acquired (using an available hardware-based trigger) and decoded. In other words, you are triggering on protocol search. Therefore, the :SBUS<n>:FLEXray:TRIGger commands are valid only when protocol decode is turned on.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger?
```

The :SBUS<n>:FLEXray:TRIGger? query returns the current FLEXray "trigger on" condition.

Returned Format

```
<condition><NL>

<condition> ::= {FRAM | ERR}
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger:ERRor:TYPE

Command

```
:SBUS<N>:FLEXray:TRIGger:ERRor:TYPE <error_type>
```

```
<error_type> ::= {ALL | HCRC | FCRC}
```

Selects the FlexRay error type to trigger on. The error type setting is only valid when the FlexRay trigger mode is set to ERRor.

- ALL — triggers on ALL errors.
- HCRC — triggers on only Header CRC errors.
- FCRC — triggers on only Frame CRC errors.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger:ERRor:TYPE?
```

The :SBUS<n>:FLEXray:TRIGger:ERRor:TYPE? query returns the currently selected FLEXray error type.

Returned Format

```
<error_type><NL>
```

```
<error_type> ::= {ALL | HCRC | FCRC}
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase

Command

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase <cycle_count_base>
```

```
<cycle_count_base> ::= integer from 0-63
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase command sets the base of the FlexRay cycle count (in the frame header) to trigger on. The cycle count base setting is only valid when the FlexRay trigger mode is set to FRAME.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase? query returns the current cycle count base setting for the FlexRay frame trigger setup.

Returned Format

```
<cycle_count_base><NL>
```

```
<cycle_count_base> ::= integer from 0-63
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition

Command

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition <cycle_count_repetition>
```

```
<cycle_count_repetition> ::= {ALL | <rep_#>}
```

```
<rep_#> ::= integer values 2, 4, 8, 16, 32, or 64
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition command sets the repetition number of the FlexRay cycle count (in the frame header) to trigger on. The cycle count repetition setting is only valid when the FlexRay trigger mode is set to FRAME.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition? query returns the current cycle count repetition setting for the FlexRay frame trigger setup.

Returned Format

```
<cycle_count_repetition><NL>
```

```
<cycle_count_repetition> ::= {ALL | <rep_#>}
```

```
<rep_#> ::= integer values 2, 4, 8, 16, 32, or 64
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger:FRAMe:ID

Command

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:ID <frame_id>

<frame_id> ::= {ALL | <frame_#>}

<frame_#> ::= integer from 1-2047
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:ID command sets the FlexRay frame ID to trigger on. The frame ID setting is only valid when the FlexRay trigger mode is set to FRAMe.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:ID?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:ID? query returns the current frame ID setting for the FlexRay frame trigger setup.

Returned Format

```
<frame_id><NL>

<frame_id> ::= {ALL | <frame_#>}

<frame_#> ::= integer from 1-2047
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE" on page 1456

History New in version 5.20.

:SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE

Command

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE <frame_type>
```

```
<frame_type> ::= {NORMal | STARtup | NULL | SYNC | NNULl | ALL}
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE command sets the FlexRay frame type to trigger on. The frame type setting is only valid when the FlexRay trigger mode is set to FRAME.

- NORMal — will trigger on only normal (NSTArtup & NNULl & NSYNc) frames.
- STARtup — will trigger on only startup frames.
- NULL — will trigger on only null frames.
- SYNC — will trigger on only sync frames.
- NNULl — will trigger on frames other than null frames.
- ALL — will trigger on all FlexRay frame types.

<N> An integer, 1-4.

Query

```
:SBUS<N>:FLEXray:TRIGger:FRAMe:TYPE?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE? query returns the current frame type setting for the FlexRay frame trigger setup.

Returned Format

```
<frame_type><NL>
```

```
<frame_type> ::= {NORM | STAR | NULL | SYNC | NNUL | ALL}
```

See Also

- ":SBUS<N>:FLEXray:BAUDrate" on page 1448
- ":SBUS<N>:FLEXray:CHANnel" on page 1449
- ":SBUS<N>:FLEXray:SOURce" on page 1450
- ":SBUS<N>:FLEXray:TRIGger" on page 1451
- ":SBUS<N>:FLEXray:TRIGger:ERRor:TYPE" on page 1452
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCBase" on page 1453
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 1454
- ":SBUS<N>:FLEXray:TRIGger:FRAMe:ID" on page 1455

History New in version 5.20.

:SBUS<N>:GENRaw Commands

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:GENRaw:SOURce

Command `:SBUS<N>:GENRaw:SOURce <source>`

The :SBUS<N>:GENRaw:SOURce command sets the source for the Generic Raw signal.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | NONE}

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

Query `:SBUS<N>:GENRaw:SOURce?`

The :SBUS<N>:GENRaw:SOURce? query returns the source of the Generic Raw signal.

Returned Format `<source><NL>`

See Also

- ":SBUS<N>:GENRaw:WSIZe" on page 1459

History New in version 6.20.

:SBUS<N>:GENRaw:WSIZe

Command

```
:SBUS<N>:GENRaw:WSIZe <word_size>
```

The :SBUS<N>:GENRaw:WSIZe command sets the Generic Raw protocol decode word size.

<word_size> An integer, 1-32.

Query

```
:SBUS<N>:GENRaw:WSIZe?
```

The :SBUS<N>:GENRaw:WSIZe? query returns the specified word size.

Returned Format

```
<word_size><NL>
```

See Also

- ":SBUS<N>:GENRaw:SOURce" on page 1458

History New in version 6.20.

:SBUS<N>:HS Commands

- ":SBUS<N>:HS:DESCramble" on page 1461
- ":SBUS<N>:HS:FORMat" on page 1462
- ":SBUS<N>:HS:IDLE" on page 1463
- ":SBUS<N>:HS:SOURce<S>" on page 1464

NOTE These commands are valid only when the high-speed (HS) serial decode type has been set with the :BUS:B<N>:TYPE command.

See Also

- ":BUS:B<N>:TYPE" on page 410

:SBUS<N>:HS:DESCramble

Command

```
:SBUS<N>:HS:DESCramble {{0 | OFF} | {1 | ON}}
```

The :SBUS<N>:HS:DESCramble command turns high-speed descrambling on or off if supported by the protocol type.

NOTE This command is only valid when a serial decode option has been licensed.

Query

```
:SBUS<N>:HS:DESCramble?
```

The :SBUS<N>:HS:DESCramble? query returns the current descrambling setting of the high-speed serial decode bus.

Returned Format

```
[:SBUS<N>:HS:DESCramble] {0 | 1}<NL>
```

See Also

- ":BUS:B<N>:TYPE" on page 410
- ":SBUS<N>:HS:FORMat" on page 1462
- ":SBUS<N>:HS:IDLE" on page 1463
- ":SBUS<N>:HS:SOURce<S>" on page 1464

History New in version 5.00.

:SBUS<N>:HS:FORMat

Command

```
:SBUS<N>:HS:FORMat <value>
```

```
<value> ::= {KDCode | LABel | F8Bit | F10Bit}
```

The :SBUS<N>:HS:FORMat command specifies the high-speed symbol display format.

<N> Is an integer, 1-4.

Example This example sets the K/D Code symbol display format.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":SBUS<N>:HS:FORMat KDCode"
```

Query

```
:SBUS<N>:HS:FORMat?
```

The :SBUS<N>:HS:FORMat? query returns the high-speed symbol display format setting.

Returned Format

```
[:SBUS<N>:HS:FORMat] <value><NL>
```

```
<value> ::= {KDCode | LABel | F8Bit | F10Bit}
```

See Also

- ":BUS:B<N>:TYPE" on page 410
- ":SBUS<N>:HS:DESCramble" on page 1461
- ":SBUS<N>:HS:IDLE" on page 1463
- ":SBUS<N>:HS:SOURce<S>" on page 1464

History New in version 5.00.

:SBUS<N>:HS:IDLE

Command
```
:SBUS<N>:HS:IDLE {{0 | OFF} | {1 | ON}}
```

The :SBUS<N>:HS:IDLE command specifies whether electrical idles are present in the signal.

NOTE This command is only valid when a serial decode option has been licensed.

Query
```
:SBUS<N>:HS:IDLE?
```

The :SBUS<N>:HS:IDLE? query returns the current ".electrical idles are present" setting.

Returned Format
```
[:SBUS<N>:HS:IDLE] {0 | 1}<NL>
```

See Also

- ":BUS:B<N>:TYPE" on page 410
- ":SBUS<N>:HS:DESCramble" on page 1461
- ":SBUS<N>:HS:FORMat" on page 1462
- ":SBUS<N>:HS:SOURce<S>" on page 1464

History New in version 5.00.

:SBUS<N>:HS:SOURce<S>

Command

```
:SBUS<N>:HS:SOURce<S> <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
    | WMEMory<R> | NONE}
```

The :SBUS<N>:HS:SOURce<S> command specifies the signal that is the high-speed data source.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<S> Is an integer, 1-4, for the high-speed serial source.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example specifies channel 2 is the high-speed data source 3 signal.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SBUS1:HS:SOURce3 CHANnel2"
```

Query

```
:SBUS<N>:HS:SOURce<S>?
```

The :SBUS<N>:HS:SOURce<S>? query returns the current signal for the high-speed data source.

Returned Format

```
[:SBUS<N>:HS:SOURce<S>] <source><NL>

<source> ::= {CHAN<N> | FUNC<F> | WMEM<N> | NONE}
```

See Also

- ":BUS:B<N>:TYPE" on page 410
- ":SBUS<N>:HS:DESCramble" on page 1461
- ":SBUS<N>:HS:FORMat" on page 1462
- ":SBUS<N>:HS:IDLE" on page 1463

History New in version 5.00.

:SBUS<N>:IIC Commands

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:IIC:ASIZe

Command

```
: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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:IIC:ASIZe?
```

The :SBUS<N>:IIC:ASIZe? query returns the current IIC address width setting.

Returned Format

```
[:SBUS<N>:IIC:ASIZe] <size><NL>
```

See Also

- ":SBUS<N>:IIC:TRIGger:PATTern:ADDRess (9000 Series, 9000H Series, S-Series)" on page 1469
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:IIC:SOURce:CLOCk

Command

```
:SBUS<N>:IIC:SOURce:CLOCk <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:IIC:SOURce:CLOCk command sets the source for the IIC serial clock (SCL).

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 2 as the source for IIC serial clock.

```
myScope.WriteString ":SBUS1:IIC:SOURce:CLOCk CHANnel2"
```

Query

```
:SBUS<N>:IIC:SOURce:CLOCk?
```

The :SBUS<N>:IIC:SOURce:CLOCk? query returns the current source for the IIC serial clock.

Returned Format

```
[:SBUS<N>:IIC:SOURce:CLOCk] <source><NL>
```

See Also

- ":SBUS<N>:IIC:SOURce:DATA" on page 1468
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:IIC:SOURce:DATA

Command

```
:SBUS<N>:IIC:SOURce:DATA <source>
```

```
<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
    | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:IIC:SOURce:DATA command sets the source for IIC serial data (SDA).

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 1 as the source for IIC serial data.

```
myScope.WriteString ":SBUS1:IIC:SOURce:DATA CHANnel1"
```

Query

```
:SBUS<N>:IIC:SOURce:DATA?
```

The :SBUS<N>:IIC:SOURce:DATA? query returns the current source for IIC serial data.

Returned Format

```
[:SBUS<N>:IIC:SOURce:DATA] <source><NL>
```

See Also

- ":SBUS<N>:IIC:SOURce:CLOCk" on page 1467
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:IIC:TRIGger:PATTern:ADDRess (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:IIC:TRIGger:PATTern:ADDRess <quoted_string>
```

NOTE You must set :SBUS<N>:MODE to IIC before you can send other :SBUS<N>:IIC:TRIGger commands.

The :SBUS<n>:IIC:TRIGger:PATTern:ADDRess command specifies the IIC address value to trigger on.The address can be a 7-, 8-, 10-, or 11-bit address depending upon the :SBUS<n>IIC:TRIGger:TYPE specification and the :SBUS<n>IIC:ASIZe setting.

The :SBUS<N>:IIC:TRIGger:TYPE command must select a type that includes an address value before you can use the :SBUS<N>:IIC:TRIGger:PATTern:ADDRess commands.

<N> An integer, 1-4.

<quoted_string> If the quoted string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters (for example, "0x34XF"); otherwise, it is a binary string made up of 0, 1, and X (don't care) characters (for example, "00110100XXXX1111").

Example To enable the SBUS1 trigger, set the trigger type to a 7-bit address frame read, and specify an address value of 0x3F:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":SBUS1:MODE IIC"
myScope.WriteString ":SBUS1:IIC:SOURce:DATA CHANnel1"
myScope.WriteString ":SBUS1:IIC:SOURce:CLOCk CHANnel2"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:IIC:TRIGger:TYPE READ7"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:ADDRess '0x3f'"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:DATA '0x7fffff'"
```

Query

```
:SBUS<N>:IIC:TRIGger:PATTern:ADDRess?
```

The :SBUS<N>:IIC:TRIGger:PATTern:ADDRess? query returns the current pattern for the IIC address.

Returned Format

```
[:SBUS<N>:IIC:TRIGger:PATTern:ADDRess] <binary_string><NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:IIC:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)" on page 1472
- ":SBUS<N>:IIC:ASIZe" on page 1466

History New in version 3.50.

:SBUS<N>:IIC:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:IIC:TRIGger:PATTern:DATA <quoted_string>
```

NOTE You must set :SBUS<N>:MODE to IIC before you can send other :SBUS<N>:IIC:TRIGger commands.

The :SBUS<N>:IIC:TRIGger:PATTern:DATA command sets IIC data.

You can specify 1 to 20 bytes of data in binary or hex format.

The :SBUS<N>:IIC:TRIGger:TYPE command must select a type that includes a data value before you can use the :SBUS<N>:IIC:TRIGger:PATTern:DATA commands.

<N> An integer, 1-4.

<quoted_string> If the quoted string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters (for example, "0x34XF"); otherwise, it is a binary string made up of 0, 1, and X (don't care) characters (for example, "00110100XXXX1111").

Example To enable the SBUS1 trigger, set the trigger type to a 7-bit address frame read, and specify a data value of 0x7FFFFF:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":SBUS1:MODE IIC"
myScope.WriteString ":SBUS1:IIC:SOURce:DATA CHANnel1"
myScope.WriteString ":SBUS1:IIC:SOURce:CLOCk CHANnel2"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:IIC:TRIGger:TYPE READ7"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:ADDRess '0x3f'"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:DATA '0x7fffff'"
```

Query

```
:SBUS<N>:IIC:TRIGger:PATTern:DATA?
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATA? query returns the current pattern for IIC data.

Returned Format

```
[:SBUS<N>:IIC:TRIGger:PATTern:DATA] <binary_string><NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:IIC:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)" on page 1472

History New in version 3.50.

:SBUS<N>:IIC:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:IIC:TRIGger:TYPE {STARt | STOP | RESTart7 | RESTart10 | AACK
                          | ANACk | READ7| WRITe7 | WRITe10}
```

NOTE You must set :SBUS<N>:MODE to IIC before you can send other :SBUS<N>:IIC:TRIGger commands.

The :SBUS<N>:IIC:TRIGger:TYPE command sets the IIC trigger type:

- STARt — Start condition.
- STOP — Stop condition.
- RESTart7 — Another 7-bit start condition occurs before a stop condition.
- RESTart10 — Another 10-bit start condition occurs before a stop condition.
- AACK — Address with acknowledge.
- ANACk — Address with no acknowledge.
- READ7 — 7-bit address frame containing (Start:Address7:Read:Ack:Data).
- WRITe7 — 7-bit address frame containing (Start:Address7:Write:Ack:Data).
- WRITe10 — 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).

NOTE The short form of READ7 (READ7), WRITe7 (WRIT7), WRITe10 (WRIT10), RESTart7 (REST7), and RESTart10 (REST10) do not follow the defined long form to short form truncation rules.

The :SBUS<N>:IIC:TRIGger:TYPE command must be sent before the :SBUS<N>:IIC:TRIGger:PATTern:ADDRess or :SBUS<N>:IIC:TRIGger:PATTern:DATA commands.

<N> An integer, 1-4.

Example To enable the SBUS1 trigger and set the IIC trigger type to a 7-bit address frame read:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":SBUS1:MODE IIC"
myScope.WriteString ":SBUS1:IIC:SOURce:DATA CHANnel1"
myScope.WriteString ":SBUS1:IIC:SOURce:CLOCk CHANnel2"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:IIC:TRIGger:TYPE READ7"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:ADDRess '0x3f'"
myScope.WriteString ":SBUS1:IIC:TRIGger:PATTern:DATA '0x7fffff'"
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428

- ":SBUS<N>:IIC:TRIGger:PATTern:ADDRess (9000 Series, 9000H Series, S-Series)" on page 1469
- ":SBUS<N>:IIC:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1471

History New in version 3.50.

:SBUS<N>:LIN Commands

NOTE These commands are only valid when the automotive LIN serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:LIN:SAMPlepoint

Command

```
:SBUS<N>:LIN:SAMPlepoint <value>

<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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:SAMPlepoint?
```

The :SBUS<n>:LIN:SAMPlepoint? query returns the current LIN sample point setting.

Returned Format

```
<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:SIGNal:BAUDrate

Command

```
:SBUS<N>:LIN:SIGNal:BAUDrate <baudrate>
```

```
<baudrate> ::= from 2400 to 625000 in NR3 format
```

The :SBUS<n>:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal from 2400 b/s to 625 kb/s.

If you enter a baud rate over 100 kb/s that is not divisible by 10 b/s, the baud rate is set to the nearest baud rate divisible by 10 b/s.

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:SIGNal:BAUDrate?
```

The :SBUS<n>:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

Returned Format

```
<baudrate><NL>
```

```
<baudrate> ::= from 2400 to 625000 in NR3 format
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:SOURce

Command

```
:SBUS<N>:LIN:SOURce <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<n>:LIN:SOURce command sets the source for the LIN signal.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:LIN:SOURce?
```

The :SBUS<n>:LIN:SOURce? query returns the current source for the LIN signal.

Returned Format

```
<source><NL>
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:STANdard

Command

```
:SBUS<N>:LIN:STANdard <std>

<std> ::= {LIN13 | LIN20}
```

The :SBUS<n>:LIN:STANdard command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:STANdard?
```

The :SBUS<n>:LIN:STANdard? query returns the current LIN standard setting, which is always LIN20.

When triggering, the oscilloscope looks for both the LIN 1.3 and 2.0 checksum.

Returned Format

```
<std><NL>

<std> ::= LIN20
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:TRIGger

Command

```
:SBUS<N>:LIN:TRIGger <condition>

<condition> ::= {ID | DATA | PARityerror | CSUMerror | ALLerrors}
```

The :SBUS<n>:LIN:TRIGger command sets the LIN trigger condition to be:

- 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.
- ALLerrors — all errors.

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:TRIGger?
```

The :SBUS<n>:LIN:TRIGger? query returns the current LIN trigger value.

Returned Format

```
<condition><NL>

<condition> ::= {ID | DATA | PAR | CSUM | ALL}
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:TRIGger:ID

Command

```
:SBUS<N>:LIN:TRIGger:ID <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

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.

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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:TRIGger:ID?
```

The :SBUS<n>:LIN:TRIGger:ID? query returns the current LIN identifier setting.

Returned Format

```
<string><NL> in 6-bit binary string format
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:TRIGger:PATTern:DATA

Command

```
:SBUS<N>:LIN: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>:LIN:TRIGger:PATTern:DATA command defines the LIN data pattern resource according to the string parameter. This pattern, along with the data length (set by the :SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth command), control the data pattern searched for in each LIN 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>:LIN:TRIGger:PATTern:DATA:LENGth command, the most significant bits will be truncated.**

<N> An integer, 1-4.

Query

```
:SBUS<N>:LIN:TRIGger:PATTern:DATA?
```

The :SBUS<N>:LIN:TRIGger:PATTern:DATA? query returns the current settings of the specified LIN data pattern resource in the binary string format.

Returned Format

```
<string><NL> in nondecimal format
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth" on page 1482

History New in version 5.20.

:SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth

Command

```
:SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8.
```

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

```
:SBUS<N>:LIN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? query returns the current LIN data pattern length setting.

Returned Format

```
<length><NL>

<length> ::= integer from 1 to 8.
```

See Also

- ":SBUS<N>:LIN:SAMPlepoint" on page 1475
- ":SBUS<N>:LIN:SIGNal:BAUDrate" on page 1476
- ":SBUS<N>:LIN:SOURce" on page 1477
- ":SBUS<N>:LIN:STANdard" on page 1478
- ":SBUS<N>:LIN:TRIGger" on page 1479
- ":SBUS<N>:LIN:TRIGger:ID" on page 1480
- ":SBUS<N>:LIN:TRIGger:PATTern:DATA" on page 1481

History New in version 5.20.

:SBUS<N>:SPI Commands

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:SPI:BITorder

Command

```
:SBUS<N>:SPI:BITorder <order>
```

```
<order> ::= {LSB | MSB}
```

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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:SPI:BITorder?
```

The :SBUS<N>:SPI:BITorder? query returns the current SPI decode bit order.

Returned Format

```
[:SBUS<N>:SPI:BITorder] <order><NL>
```

See Also

- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:SPI:CLOCk:SLOPe

Command

```
:SBUS<N>:SPI:CLOCk:SLOPe <slope>

<slope> ::= {POSitive | RISing | NEGative | FALLing}
```

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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:SPI:CLOCk:SLOPe?
```

The :SBUS<N>:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Returned Format

```
[:SBUS<N>:SPI:CLOCk:SLOPe] <slope><NL>

<slope> ::= {RIS | FALL}
```

See Also

- ":SBUS<N>:SPI:CLOCk:TIMeout" on page 1486
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:SPI:CLOCk:TIMeout

Command

```
: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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:SPI:CLOCk:TIMeout?
```

The :SBUS<N>:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Returned Format

```
[:SBUS<N>:SPI:CLOCk:TIMeout] <time value><NL>
```

See Also

- ":SBUS<N>:SPI:CLOCk:SLOPe" on page 1485
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:FRAMe:STATe" on page 1487
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:SPI:FRAMe:STATe

Command

```
:SBUS<N>:SPI:FRAMe:STATe <value>
```

```
<value> ::= {LOW | HIGH}
```

The :SBUS<N>:SPI:FRAMe:STATe command sets the SPI trigger frame state.

<N> An integer, 1-4.

Query

```
:SBUS<N>:SPI:FRAMe:STATe?
```

The :SBUS<N>:SPI:FRAMe:STATe? query returns the current SPI frame state.

Returned Format

```
[:SBUS<N>:SPI:FRAMe:STATe] <value><NL>
```

See Also

- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:SPI:SOURce:CLOCk

Command

```
:SBUS<N>:SPI:SOURce:CLOCk <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:SPI:SOURce:CLOCk?
```

The :SBUS<N>:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Returned Format

```
[:SBUS<N>:SPI:SOURce:CLOCk] <source><NL>
```

See Also

- ":SBUS<N>:SPI:CLOCk:SLOPe" on page 1485
- ":SBUS<N>:SPI:CLOCk:TIMeout" on page 1486
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:SPI:SOURce:DATA

Command

```
:SBUS<N>:SPI:SOURce:DATA <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

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.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:SPI:SOURce:DATA?
```

The :SBUS<N>:SPI:SOURce:DATA? query returns the current source for the SPI serial MOSI data.

Returned Format

```
[:SBUS<N>:SPI:SOURce:DATA] <source><NL>
```

See Also

- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:SPI:SOURce:FRAMe

Command

```
:SBUS<N>:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
    | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:SPI:SOURce:FRAMe command sets the frame source.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:SPI:SOURce:FRAMe?
```

The :SBUS<N>:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Returned Format

```
[:SBUS<N>:SPI:SOURce:FRAMe] <source><NL>
```

See Also

- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:SPI:FRAMe:STATe" on page 1487
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:SPI:SOURce:MISO

Command

```
:SBUS<N>:SPI:SOURce:MISO <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
    | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:SPI:SOURce:MISO command sets the source for the SPI serial MISO data.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:SPI:SOURce:MISO?
```

The :SBUS<N>:SPI:SOURce:MISO? query returns the current source for the SPI serial MISO data.

Returned Format

```
[:SBUS<N>:SPI:SOURce:MISO] <source><NL>
```

See Also

- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:SPI:SOURce:MOSI

Command

```
:SBUS<N>:SPI:SOURce:MOSI <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<R> | DIGital<M> | NONE}
```

The :SBUS<N>:SPI:SOURce:MOSI command sets the source for the SPI serial MOSI data.

You can also use the equivalent :SBUS<N>:SPI:SOURce:DATA command to set the MOSI data source.

The NONE parameter is the same as selecting "None" for the source in the user interface. It makes the previously selected channel, waveform memory, or math function available for other decodes.

<N> SBUS<N> is an integer, 1-4.

CHANnel<N> is an integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:SBUS<N>:SPI:SOURce:MOSI?
```

The :SBUS<N>:SPI:SOURce:MOSI? query returns the current source for the SPI serial MOSI data.

Returned Format

```
[:SBUS<N>:SPI:SOURce:MOSI] <source><NL>
```

See Also

- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

Version 5.20: The NONE parameter was added.

:SBUS<N>:SPI:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:SPI:TRIGger:PATTern:DATA <quoted_string>
```

NOTE You must set :SBUS<N>:MODE to SPI before you can send other :SBUS<N>:SPI:TRIGger commands.

The :SBUS<N>:SPI:TRIGger:PATTern: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.

The :SBUS<N>:SPI:TRIGger:PATTern:DATA command must receive the number of bytes specified by the :SBUS<N>:SPI:TRIGger:PATTern:WIDTh command.

NOTE The :SBUS<N>:SPI:TRIGger:PATTern:WIDTh should be set before :SBUS<N>:SPI:TRIGger:PATTern:DATA.

<N> An integer, 1-4.

<quoted_string> If the quoted string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters (for example, "0x34XF"); otherwise, it is a binary string made up of 0, 1, and X (don't care) characters (for example, "00110100XXXX1111").

Example To enable the SBUS1 trigger, set the SPI trigger type to MOSI, set a 32-bit data pattern width, and specify the 0x0080FFFF data pattern:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":CHANnel3:DISPlay ON"
myScope.WriteString ":CHANnel4:DISPlay ON"
myScope.WriteString ":SBUS1:MODE SPI"
myScope.WriteString ":SBUS1:SPI:WIDTh 16"
myScope.WriteString ":SBUS1:SPI:TYPE WIRE4"
myScope.WriteString ":SBUS1:SPI:SOURce:MOSI CHANnel1"
myScope.WriteString ":SBUS1:SPI:SOURce:CLOCk CHANnel2"
myScope.WriteString ":SBUS1:SPI:SOURce:FRAMe CHANnel3"
myScope.WriteString ":SBUS1:SPI:SOURce:MISO CHANnel4"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:SPI:TRIGger:TYPE MOSI"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:WIDTh 2"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:DATA '0x0080ffff'"
```

Query

```
:SBUS<N>:SPI:TRIGger:PATTern:DATA?
```

The :SBUS<N>:SPI:TRIGger:PATTern:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Returned Format

```
[:SBUS<N>:SPI:TRIGger:PATTern:DATA] <binary_string><NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:SPI:TRIGger:PATTern:WIDTh (9000 Series, 9000H Series, S-Series)" on page 1497
- ":SBUS<N>:SPI:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)" on page 1499

History New in version 3.50.

:SBUS<N>:SPI:TRIGger:PATTern:WIDTh (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:SPI:TRIGger:PATTern:WIDTh {1 - 20}
```

NOTE You must set :SBUS<N>:MODE to SPI before you can send other :SBUS<N>:SPI:TRIGger commands.

The :SBUS<N>:SPI:TRIGger:PATTern:WIDTh command sets the width of the SPI data pattern. You can specify the width in multiples of the SPI word size up to 20.

NOTE The :SBUS<N>:SPI:TRIGger:PATTern:WIDTh should be set before :SBUS<N>:SPI:TRIGger:PATTern:DATA.

<N> An integer, 1-4.

Example To enable the SBUS1 trigger, set the SPI trigger type to MOSI, set a 32-bit data pattern width (two 16-bit words), and specify the 0x0080FFFF data pattern:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":CHANnel3:DISPlay ON"
myScope.WriteString ":CHANnel4:DISPlay ON"
myScope.WriteString ":SBUS1:MODE SPI"
myScope.WriteString ":SBUS1:SPI:WIDTh 16"
myScope.WriteString ":SBUS1:SPI:TYPE WIRE4"
myScope.WriteString ":SBUS1:SPI:SOURce:MOSI CHANnel1"
myScope.WriteString ":SBUS1:SPI:SOURce:CLOCk CHANnel2"
myScope.WriteString ":SBUS1:SPI:SOURce:FRAMe CHANnel3"
myScope.WriteString ":SBUS1:SPI:SOURce:MISO CHANnel4"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:SPI:TRIGger:TYPE MOSI"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:WIDTh 2"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:DATA '0x0080ffff'"
```

Query

```
:SBUS<N>:SPI:TRIGger:PATTern:WIDTh?
```

The :SBUS<N>:SPI:TRIGger:PATTern:WIDTh? query returns the current SPI data pattern width setting.

Returned Format

```
[:SBUS<N>:SPI:TRIGger:PATTern:WIDTh] {1 - 20}<NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:SPI:WIDTh" on page 1502
- ":SBUS<N>:SPI:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)" on page 1499

- ":SBUS<N>:SPI:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1495

History New in version 3.50.

:SBUS<N>:SPI:TRIGger:TYPE (9000 Series, 9000H Series, S-Series)

Command

```
:SBUS<N>:SPI:TRIGger:TYPE <value>

<value> ::= {MOSI | MISO}
```

NOTE You must set :SBUS<N>:MODE to SPI before you can send other :SBUS<N>:SPI:TRIGger commands.

The :SBUS<N>:SPI:TRIGger:TYPE command specifies whether the SPI trigger will be on the MOSI data or the MISO data.

The data value is specified by the :SBUS<N>:SPI:TRIGger:PATTern:DATA and :SBUS<N>:SPI:TRIGger:PATTern:WIDTh commands.

<N> An integer, 1-4.

Example To enable the SBUS1 trigger and set the SPI trigger type to MOSI:

```
myScope.WriteString ":CHANnel1:DISPlay ON"
myScope.WriteString ":CHANnel2:DISPlay ON"
myScope.WriteString ":CHANnel3:DISPlay ON"
myScope.WriteString ":CHANnel4:DISPlay ON"
myScope.WriteString ":SBUS1:MODE SPI"
myScope.WriteString ":SBUS1:SPI:WIDTh 16"
myScope.WriteString ":SBUS1:SPI:TYPE WIRE4"
myScope.WriteString ":SBUS1:SPI:SOURce:MOSI CHANnel1"
myScope.WriteString ":SBUS1:SPI:SOURce:CLOCk CHANnel2"
myScope.WriteString ":SBUS1:SPI:SOURce:FRAMe CHANnel3"
myScope.WriteString ":SBUS1:SPI:SOURce:MISO CHANnel4"
myScope.WriteString ":TRIGger:MODE SBUS1"
myScope.WriteString ":SBUS1:SPI:TRIGger:TYPE MOSI"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:WIDTh 2"
myScope.WriteString ":SBUS1:SPI:TRIGger:PATTern:DATA '0x0080ffff'"
```

Query

```
:SBUS<N>:SPI:TRIGger:TYPE?
```

The :SBUS<N>:SPI:TRIGger:TYPE? query returns the trigger type setting.

Returned Format

```
[:SBUS<N>:SPI:TRIGger:TYPE] <value><NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":SBUS<N>:MODE" on page 1428
- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:TRIGger:PATTern:DATA (9000 Series, 9000H Series, S-Series)" on page 1495
- ":SBUS<N>:SPI:TRIGger:PATTern:WIDTh (9000 Series, 9000H Series, S-Series)" on page 1497

History New in version 3.50.

:SBUS<N>:SPI:TYPE

Command

```
:SBUS<N>:SPI:TYPE <value>
```

```
<value> ::= {WIRE2 | WIRE3 | WIRE4}
```

The :SBUS<N>:SPI:TYPE command specifies whether the type of SPI to decode.

<N> An integer, 1-4.

Example To set the 3-wire SPI decode type:

```
myScope.WriteString ":SBUS1:SPI:TYPE WIRE3"
```

Query

```
:SBUS<N>:SPI:TYPE?
```

The :SBUS<N>:SPI:TYPE? query returns the decode type setting.

Returned Format

```
[:SBUS<N>:SPI:TYPE] <value><NL>
```

See Also

- ":SBUS<N>:SPI:BITorder" on page 1484
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 1488
- ":SBUS<N>:SPI:SOURce:DATA" on page 1490
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 1491
- ":SBUS<N>:SPI:SOURce:MISO" on page 1492
- ":SBUS<N>:SPI:SOURce:MOSI" on page 1493
- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:SPI:WIDTh

Command

```
: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.

<N> An integer, 1-4.

Query

```
:SBUS<N>:SPI:WIDTh?
```

The :SBUS<N>:SPI:WIDTh? query returns the current SPI decode word width.

Returned Format

```
[:SBUS<N>:SPI:WIDTh] <word_width><NL>
```

```
<word_width> ::= integer 4-16 in NR1 format
```

See Also

- ":SBUS<N>:MODE" on page 1428

History New in version 3.50.

:SBUS<N>:UART Commands

NOTE These commands are only valid when the Low-Speed Protocol Decode and Trigger option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 1428

:SBUS<N>:UART:BAUDrate

Command

```
:SBUS<N>:UART:BAUDrate <baudrate>
```

The :SBUS<N>:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode.

<N> An integer, 1-4.

<baudrate> Bits/second in NR3 format.

Query

```
:SBUS<N>:UART:BAUDrate?
```

The :SBUS<N>:UART:BAUDrate? query returns the bit rate (in bps) setting.

Returned Format

```
<baudrate><NL>

<baudrate> ::= bits/second in NR3 format.
```

See Also

- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:BITorder

Command

```
:SBUS<N>:UART:BITorder {LSB | MSB}
```

The :SBUS<N>:UART:BITorder command selects whether the most significant bit (MSB) or least significant bit (LSB) is presented after the start bit in the signal from your device under test.

For RS-232, select LSB.

<N> An integer, 1-4.

Query

```
:SBUS<N>:UART:BITorder?
```

The :SBUS<N>:UART:BITorder? query returns the bit order setting.

Returned Format

```
<bitorder><NL>

<bitorder> ::= {LSB | MSB}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:DIRection

Command

```
:SBUS<N>:UART:DIRection {RXTX | RX | TX}
```

The :SBUS<N>:UART:DIRection command specifies whether you are decoding Rx only (RX), Tx only (TX), or Rx and Tx (RXTX).

<N> An integer, 1-4.

Query

```
:SBUS<N>:UART:DIRection?
```

The :SBUS<N>:UART:DIRection? query returns the direction setting.

Returned Format

```
<direction><NL>

<direction> ::= {RXTX | RX | TX}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:EOF:HEX

Command

```
:SBUS<N>:UART:EOF:HEX <value>
```

The :SBUS<N>:UART:EOF:HEX command specifies the End-Of-Frame hexadecimal byte value.

<value> End of frame hexadecimal byte value.

Query

```
:SBUS<N>:UART:EOF:HEX?
```

The :SBUS<N>:UART:EOF:HEX? query returns the End-Of-Frame hexadecimal byte value.

Returned Format

```
<value><NL>

<value> ::= End of frame hex byte value.
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:IDLE

Command

```
:SBUS<N>:UART:IDLE {LOW | HIGH}
```

The :SBUS<N>:UART:IDLE command selects LOW or HIGH to match the device under test's state when at idle.

For RS-232, select idle LOW.

Query

```
:SBUS<N>:UART:IDLE?
```

The :SBUS<N>:UART:IDLE? query returns the selected idle state value.

Returned Format

```
<polarity><NL>

<polarity> ::= {LOW | HIGH}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History

New in version 11.10.

:SBUS<N>:UART:PARity

Command

```
:SBUS<N>:UART:PARity {EVEN | ODD | NONE}
```

The :SBUS<N>:UART:PARity command specifies the type of parity being used.

Query

```
:SBUS<N>:UART:PARity?
```

The :SBUS<N>:UART:PARity? query returns the parity setting.

Returned Format

```
<parity><NL>

<parity> ::= {EVEN | ODD | NONE}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:SOURce:RX

Command

```
:SBUS<N>:UART:SOURce:RX <source>
```

The :SBUS<N>:UART:SOURce:RX command specifies the receiver signal source.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15.

Query

```
:SBUS<N>:UART:SOURce:RX?
```

The :SBUS<N>:UART:SOURce:RX? query returns the receiver signal source setting.

Returned Format

```
<source><NL>

<source> ::= {CHAN<N> | FUNC<F> | WMEM<N> | DIG<M>}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:TX" on page 1511
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:SOURce:TX

Command

```
:SBUS<N>:UART:SOURce:TX <source>
```

The :SBUS<N>:UART:SOURce:TX command specifies the transmitter signal source.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15.

Query

```
:SBUS<N>:UART:SOURce:TX?
```

The :SBUS<N>:UART:SOURce:TX? query returns the transmitter source signal setting.

Returned Format

```
<source><NL>

<source> ::= {CHAN<N> | FUNC<F> | WMEM<N> | DIG<M>}
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:WIDTh" on page 1512

History New in version 11.10.

:SBUS<N>:UART:WIDTh

Command

```
:SBUS<N>:UART:WIDTh <width>
```

The :SBUS<N>:UART:WIDTh command sets the number of bits in RS-232/UART words to match your device under test.

<width> An integer, 5-9.

Query

```
:SBUS<N>:UART:WIDTh?
```

The :SBUS<N>:UART:WIDTh? query returns the RS-232/UART word size setting.

Returned Format

```
<width><NL>
```

See Also

- ":SBUS<N>:UART:BAUDrate" on page 1504
- ":SBUS<N>:UART:BITorder" on page 1505
- ":SBUS<N>:UART:DIRection" on page 1506
- ":SBUS<N>:UART:EOF:HEX" on page 1507
- ":SBUS<N>:UART:IDLE" on page 1508
- ":SBUS<N>:UART:PARity" on page 1509
- ":SBUS<N>:UART:SOURce:RX" on page 1510
- ":SBUS<N>:UART:SOURce:TX" on page 1511

History New in version 11.10.

41 :SELFtest (Self-Test) Commands

The SELFtest subsystem commands set up the self-test dialog and run the Infiniium-Series Oscilloscopes Self-Tests.

NOTE

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

:SELFtest:CANCel

Command `:SELFtest:CANCel`

The :SELFtest:CANCel command stops the currently running selftest.

Example This example stops the currently running selftest.

```
myScope.WriteString ":SELF:CANC"
```

History Legacy command (existed before version 3.10).

:SELFtest:SCOPETEST

Command
```
:SELFtest:SCOPETEST
```

The :SELFtest:SCOPETEST command brings up the self-test dialog in customer self-test mode (Service Extensions Off) and runs the test, "Scope Self Tests." Use the :SELFtest:SCOPETEST? query to determine the status of the test.

Example This example brings up the self-test dialog and runs the oscilloscope self-tests.

```
myScope.WriteString ":SELF:SCOPETEST"
```

Query
```
:SELFtest:SCOPETEST?
```

Returned Format
```
[:SELFtest:SCOPETEST] <test_name>,<test_status>, <time_stamp><NL>
```

<test_status>	Status Description
FAILED	Test completed and failed.
PASSED	Test completed and passed.
WARNING	Test passed but warning message was issued.
CANCELLED	Test was cancelled by user.
NODATA	Self-tests have not been executed on this instrument.
INPROGRESS	Test is in progress.

<test_name> A string as follows: "Scope Self Tests".

<time_stamp> The time stamp follows the test name and test status, and is the part of the returned string that includes the date and time, in the format: "20 May 2009 10:13:35".

Example This example places the current status of the self-test in the string variable, strTxt, then prints the contents of the variable to the computer's screen.

```
Dim strTxt As String
myScope.WriteString ":SELF:SCOPETEST?"
strTxt = myScope.ReadString
Debug.Print strTxt
```

History Legacy command (existed before version 3.10).

42 :SYSTem Commands

SYSTem subsystem commands control the way query responses are formatted, send and receive setup strings, and enable reading and writing to the advisory line of the oscilloscope. You can also set and read the date and time in the oscilloscope using the SYSTem subsystem commands.

:SYSTem:CAPability:ACQuire?

Query `:SYSTem:CAPability:ACQuire? {HALFchannel | MEMory | SRATe | BANDwidth}`

For the following options, the :SYSTem:CAPability:ACQuire? query returns:

- HALFchannel — Returns whether the oscilloscope has a half-channel mode ("0" is false, "1" is true).
- MEMory — Returns the maximum analog input channel memory depth (bytes).
- SRATe — Returns the maximum analog input channel sampling rate (Sa/s).
- BANDwidth — Returns the oscilloscope's bandwidth (Hz).

Returned Format `<quoted_string><NL>`

See Also

- ":SYSTem:CAPability:CHANnel?" on page 1519
- ":SYSTem:CAPability:DIGital?" on page 1520

History New in version 10.00.

:SYSTem:CAPability:CHANnel?

Query `:SYSTem:CAPability:CHANnel? {COUNt}`

For the following options, the :SYSTem:CAPability:CHANnel? query returns:

- COUNt — Returns the oscilloscope's number of analog channels.

Returned Format `<quoted_string><NL>`

See Also

- ":SYSTem:CAPability:ACQuire?" on page 1518
- ":SYSTem:CAPability:DIGital?" on page 1520

History New in version 10.00.

:SYSTem:CAPability:DIGital?

Query `:SYSTem:CAPability:DIGital? {MEMory | SRATe | CHANnels}`

For the following options, the :SYSTem:CAPability:DIGital? query returns:

- MEMory — Returns the maximum digital input channel memory depth (bytes).
- SRATe — Returns the maximum digital input channel sampling rate (Sa/s).
- CHANnels — Returns the number of digital input channels.

Returned Format `<quoted_string><NL>`

See Also

- ":SYSTem:CAPability:ACQuire?" on page 1518
- ":SYSTem:CAPability:CHANnel?" on page 1519

History New in version 10.00.

:SYSTem:DATE

Command
```
:SYSTem:DATE <day>,<month>,<year>
```

The :SYSTem:DATE command sets the date in the oscilloscope, and is not affected by the *RST common command.

<year> Specifies the year in the format <yyyy> | <yy>. The values range from 1992 to 2035.

<month> Specifies the month in the format <1, 2, . . . 12> | <JAN, FEB, MAR . . .>.

<day> Specifies the day in the format <1 . . . 31>.

Example This example sets the date to December 1, 2002.

```
myScope.WriteString ":SYSTem:DATE 1,12,02"
```

Query
```
:SYSTem:DATE?
```

The :SYSTem:DATE? query returns the current date in the oscilloscope.

Returned Format
```
[:SYSTem:DATE] <day> <month> <year><NL>
```

Example This example queries the date.

```
Dim strDate As String
myScope.WriteString ":SYSTem:DATE?"
strDate = myScope.ReadString
Debug.Print strDate
```

History Legacy command (existed before version 3.10).

:SYSTem:DEBug

Command

```
:SYSTem:DEBug {{ON|1}[,<output_mode>[,"<file_name>" [,<create_mode>]]] |
 {OFF|0}}
```

The :SYSTem:DEBug command turns the debug mode on and off. This mode enables the tracing of incoming remote commands. If you select CREate mode, a new file is created, and/or an existing file is overwritten. If you select APPend mode, the information is appended to an existing file. The :SYSTem:DEBug command shows any header and/or parameter errors.

The default create mode is CREate, the default output mode is FileSCReen, and the default file name is "C:\Users\Public\Documents\Infiniium\debug.txt". In debug mode, the File View button lets you view the current debug file, or any other debug file. This is a read-only mode.

<output_mode> {FILE | SCReen | FileSCReen}

<file_name> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The file name assumes the present working directory if a path does not precede the file name.

<create_mode> {CREate | APPend}

Examples This example turns on the debug/trace mode and creates a debug file.

```
myScope.WriteString ":SYSTem:DEBUG ON,FILE,
""C:\Users\Public\Documents\Infiniium\pacq8xx.txt"",CREate"
```

The created file resembles:

```
Debug information file C:\Users\Public\Documents\Infiniium\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
Model: DSO90804A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTem:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTem:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTem:ERROR 0,"No error"$
```

This example appends information to the debug file.

```
myScope.WriteString ":SYSTem:DEBUG ON,FILE,
""C:\Users\Public\Documents\Infiniium\pacq8xx.txt"",APPend"
```

After appending information, the file resembles:

```
Debug information file C:\Users\Public\Documents\Infiniium\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
```

```
Model: DSO90804A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTem:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTem:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTem:ERROR 0,"No error"$

Debug information file C:\Users\Public\Documents\Infiniium\
pacq8xx.txt appended
Date:  1 DEC 2002
Time:  10:10:35
Model: DSO90804A
Serial#:   sn ?
>:syst:err? string$<NL>
<:SYSTem:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTem:ERROR -113,"Undefined header"$
```

Query `:SYSTem:DEBug?`

The :SYSTem:DEBug? query returns the current debug mode settings.

Returned Format

```
[:SYSTem:DEBug]  {{1,<output_mode>,"<file_name>", <create_mode>} | 0} <N
L>
```

History Legacy command (existed before version 3.10).

:SYSTem:DIMPedance — (MXR/EXR-Series)

Command `:SYSTem:DIMPedance {FIFTy | ONEMeg}`

The :SYSTem:DIMPedance command sets the **Default Impedance to 1M Ohm** user preference.

- ONEMeg — a default setup will set the analog channels' input impedance to 1M Ohm.
- FIFTy — a default setup does not change the analog channel input impedance settings.

Note that when the oscilloscope powers down, it sets the 1 MOhm high impedance mode on the input channels as a protection mechanism while the unit is off.

Query `:SYSTem:DIMPedance?`

The :SYSTem:DIMPedance? query returns the **Default Impedance to 1M Ohm** user preference setting.

Returned Format `<setting><NL>`

`<setting> ::= {FIFT | ONEM}`

See Also

- "*RST — Reset" on page 241
- ":SYSTem:PRESet" on page 1536

History New in version 11.10.

:SYSTem:DONTtabmeas

Command `:SYSTem:DONTtabmeas {{0 | OFF} | {1 | ON}}`

The :SYSTem:DONTtabmeas command enables or disables the **Never tab Measurement Results** user preference.

When this user preference is enabled, and you choose **Display > Windows > Tabbed/Custom Window Layout** in the graphical user interface, the sub panes within the Results pane remain stacked—they are not tabbed as they would have been had this user preference been disabled.

Query `:SYSTem:DONTtabmeas?`

The :SYSTem:DONTtabmeas? query returns the **Never tab Measurement Results** user preference setting.

Returned Format `<setting><NL>`

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

History New in version 5.60.

:SYSTem:DSP

Command

```
:SYSTem:DSP "<string>"
```

The :SYSTem:DSP command writes a quoted string, excluding quotation marks, to the advisory line of the instrument display. If you want to clear a message on the advisory line, send a null (empty) string.

<string> An alphanumeric character array up to 86 bytes long.

Example This example writes the message, "Test 1" to the advisory line of the oscilloscope.

```
myScope.WriteString ":SYSTem:DSP ""Test 1"""
```

Query

```
:SYSTem:DSP?
```

The :SYSTem:DSP? query returns the last string written to the advisory line. This may be a string written with a :SYSTem:DSP command, or an internally generated advisory.

The string is actually read from the message queue. The message queue is cleared when it is read. Therefore, the displayed message can only be read once over the bus.

Returned Format

```
[:SYSTem:DSP] <string><NL>
```

Example This example places the last string written to the advisory line of the oscilloscope in the string variable, strAdvisory. Then, it prints the contents of the variable to the computer's screen.

```
Dim strAdvisory As String   ' Dimension variable.
myScope.WriteString ":SYSTem:DSP?"
strAdvisory = myScope.ReadString
Debug.Print strAdvisory
```

History Legacy command (existed before version 3.10).

:SYSTem:ERRor?

Query `:SYSTem:ERRor? [{NUMBer | STRing}]`

The :SYSTem:ERRor? query outputs the next error number in the error queue over the remote interface. When either NUMBer or no parameter is specified in the query, only the numeric error code is output. When STRing is specified, the error number is output followed by a comma and a quoted string describing the error. Table 22 lists the error numbers and their corresponding error messages.

Returned Format `[:SYSTem:ERRor] <error_number>[,<quoted_string>]<NL>`

<error_number> A numeric error code.

<quoted_string> A quoted string describing the error.

Example This example reads the oldest error number and message in the error queue into the string variable, strCondition, then prints the contents of the variable to the computer's screen.

```
Dim strCondition As String   ' Dimension variable.
myScope.WriteString ":SYSTem:ERRor? STRing"
strCondition = myScope.ReadString
Debug.Print strCondition
```

Infiniium Oscilloscopes have an error queue that is 30 errors deep and operates on a first-in, first-out (FIFO) basis. Successively sending the :SYSTem:ERRor? query returns the error numbers in the order that they occurred until the queue is empty. When the queue is empty, this query returns headers of 0, "No error." Any further queries return zeros until another error occurs. Note that front-panel generated errors are also inserted in the error queue and the Event Status Register.

NOTE

Send *CLS Before Other Commands or Queries

Send the *CLS common command to clear the error queue and Event Status Register before you send any other commands or queries.

See Also The "Error Messages" chapter for more information on error messages and their possible causes.

History Legacy command (existed before version 3.10).

:SYSTem:GUI

Command `:SYSTem:GUI {ON | OFF | LOCK}`

The :SYSTem:GUI OFF command enables or disables the front panel user interface.

- ON — Enables the front panel user interface.
- OFF — Disables the front panel user interface and places a Remote Operations In Progress dialog box on the oscilloscope's screen. The front panel knobs, keys, and graphical user interface are disabled. Graphical user interface updates are also disabled.

  The :SYSTem:GUI OFF command lets Infiniium oscilloscopes behave like other Keysight instruments by locking out the GUI (graphical user interface) and the front panel while remote scripts are running. On Infiniium oscilloscopes, the GUI and front panel do not lock automatically during remote operation (as most other instruments do) to preserve the integrity and timing of legacy customer scripts. The recommendation is, however, that all scripts begin with :SYSTem:GUI OFF when convenient and possible to run more like other Keysight instruments and likely improve performance.

  The :SYSTem:GUI OFF command is similar to the :SYSTem:LOCK ON command, except the :SYSTem:LOCK ON command does not disable the graphical user interface (just the knobs and keys).
- LOCK — Disables the front panel graphical user interface as well as the front panel knobs and keys. A Remote Operations In Progress dialog box on the oscilloscope's screen. However, graphical user interface updates are not disabled.

The front panel user interface can be re-enabled by:

- Sending the :SYSTem:GUI ON command.
- Clicking **Enable** in the Remote Operations In Progress dialog box.

Example This example disables the oscilloscope's front panel user interface.

```
myScope.WriteString ":SYSTem:GUI OFF"
```

Query `:SYSTem:GUI?`

The :SYSTem:GUI? query returns the state of the :SYSTem:GUI command.

Returned Format `[:SYSTem:GUI] {ON | OFF | LOCK}<NL>`

See Also

- ":SYSTem:LOCK" on page 1531

History New in version 5.50.

Version 6.20: The LOCK parameter has been added and the query return value is now a string (ON, OFF, or LOCK) instead of the previous 1 or 0 return values.

:SYSTem:HEADer

Command

```
:SYSTem:HEADer {{ON|1} | {OFF|0}}
```

The :SYSTem:HEADer command specifies whether the instrument will output a header for query responses. When :SYSTem:HEADer is set to ON, the query responses include the command header.

Example This example sets up the oscilloscope to output command headers with query responses.

```
myScope.WriteString ":SYSTem:HEADer ON"
```

Query

```
:SYSTem:HEADer?
```

The :SYSTem:HEADer? query returns the state of the :SYSTem:HEADer command.

Returned Format

```
[:SYSTem:HEADer] {1|0}<NL>
```

Example This example prints the system header setting.

```
Dim strSetting As String
myScope.WriteString ":SYSTem:HEADer?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

NOTE

Turn Headers Off when Returning Values to Numeric Variables

Turn headers off when returning values to numeric variables. Headers are always off for all common command queries because headers are not defined in the IEEE 488.2 standard.

History Legacy command (existed before version 3.10).

:SYSTem:HLED

Command `:SYSTem:HLED {{0 | OFF} | {1 | ON}}`

The :SYSTem:HLED command turns the "hide front panel LEDs" setting ON or OFF. When ON, all LEDs on the front panel (except the power button LED) will turn off.

You may want to hide front panel LEDs when making measurements in a light-sensitive environment.

Query `:SYSTem:HLED?`

The :SYSTem:HLED? query returns the hide LEDs setting.

Returned Format `<setting><NL>`

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

History New in version 6.00.

:SYSTem:LOCK

Command

```
:SYSTem:LOCK {{ON | 1} | {OFF | 0}}
```

The :SYSTem:LOCK ON command disables the front panel knobs and keys.

The front panel knobs and keys can be re-enabled by sending the :SYSTem:LOCK OFF command.

This :SYSTem:LOCK ON command is similar to the :SYSTem:GUI OFF command, except the :SYSTem:GUI OFF command also disables the graphical user interface and GUI updates.

Example

This example disables the oscilloscope's front panel.

```
myScope.WriteString ":SYSTem:LOCK ON"
```

Query

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the state of the :SYSTem:LOCK command.

Returned Format

```
[:SYSTem:LOCK] {1 | 0}<NL>
```

See Also

- ":SYSTem:GUI" on page 1528

History

Legacy command (existed before version 3.10).

:SYSTem:LONGform

Command

```
:SYSTem:LONGform {{ON | 1} | {OFF | 0}}
```

The :SYSTem:LONGform command specifies the format for query responses. If the LONGform is set to OFF, command headers and alpha arguments are sent from the oscilloscope in the short form (abbreviated spelling). If LONGform is set to ON, the whole word is output.

Example This example sets the format for query responses from the oscilloscope to the short form (abbreviated spelling).

```
myScope.WriteString ":SYSTem:LONGform OFF"
```

Query

```
:SYSTem:LONGform?
```

The :SYSTem:LONGform? query returns the current state of the :SYSTem:LONGform command.

Returned Format

```
[:SYSTem:LONGform] {1 | 0}<NL>
```

Example This example checks the current format for query responses from the oscilloscope, and places the result in the string variable, strResult. Then, it prints the contents of the variable to the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:LONGform?"
strResult = myScope.ReadString
Debug.Print strResult
```

NOTE

LONGform Does Not Affect Input Headers and Arguments

LONGform has no effect on input headers and arguments sent to the instrument. You may send headers and arguments to the oscilloscope in either the long form or short form, regardless of the current state of the :SYSTem:LONGform command.

History Legacy command (existed before version 3.10).

:SYSTem:MENU?

Query

```
:SYSTem:MENU? <menu_string>
```

```
<menu_string> ::= quoted string
```

The :SYSTem:MENU? query returns front panel graphical user interface (GUI) menu strings.

NOTE This query is intended to list GUI menu strings that launch other executable programs. Currently, these appear in the "Analyze" menu only. For other menus, this query may return strings that do not actually appear in the GUI.

Returned Format

```
<items_in_menu><NL>
```

```
<items_in_menu> ::= comma-separated list of items in the menu
```

Example This example shows queries and responses from the Keysight Interactive IO's Instrument Session History:

```
-> :SYSTem:MENU? "Analyze"
<- Histogram..., Mask Test..., Automated Test Apps,
Measurement Analysis (EZJIT)..., Jitter/Noise (EZJIT Complete)...,
RTEye/Clock Recovery (SDA)..., Equalization...
-> :SYSTem:MENU? "Analyze\Automated Test Apps"
<- N8829A 100GBASE-KR4 Test App, N8830A 100GBASE-CR4 Test App
```

History New in version 5.50.0033.

:SYSTem:MODE (MXR-Series)

Command

```
:SYSTem:MODE {OSCilloscope | RTSA}}
```

The :SYSTem:MODE command sets the system mode:

- OSCilloscope — Normal oscilloscope operation.

NOTE **The system must be in OSCilloscope mode to use the Spectrum Analysis (DDC) feature.**

- RTSA — Real-Time Spectrum Analyzer (RTSA) mode. This mode is available in MXR-Series oscilloscopes only.

  RTSA (Real-Time Spectral Analysis) is an oscilloscope mode where DDC input data is continuously acquired on multiple phase-coherent channels, converted to the frequency domain, and plotted without any dead time. All input data is handled.

  All channels become RTSA channels. Time domain functionality is no longer present, and the related remote commands are no longer valid. The oscilloscope becomes a multi-channel spectrum analyzer (SA) instrument.

  In RTSA mode, you use the :CHANnel<N>:SPECtral:SPAN command to set the span and the :CHANnel<N>:SPECtral:CFRequency command to set the center frequency. The selected span applies to all channels, but each channel can have a different center frequency. For more information on the RTSA mode, see the online help.

Query

```
:SYSTem:MODE?
```

The :SYSTem:MODE? query returns the mode setting.

Returned Format

```
<mode><NL>

<mode> ::= {OSC | RTSA}
```

See Also

- ":CHANnel<N>:SPECtral:CFRequency" on page 504
- ":CHANnel<N>:SPECtral:SPAN (MXR-Series)" on page 506

History New in version 11.05.

:SYSTem:PERSona

Command

```
:SYSTem:PERSona {<manufacturer_string>, <model_string> | <manufacturer_s
tring> | DEFault}
```

```
<manufacturer_string> ::= quoted string, 1-31 characters
```

```
<model_string> ::= quoted string, 1-10 characters
```

The :SYSTem:PERSona command sets the manufacturer string and the model number string returned by the *IDN? query.

Query

```
:SYSTem:PERSona?
```

The :SYSTem:PERSona? query returns the manufacturer string and the model number string.

Returned Format

```
[:SYSTem:PERSona] <manufacturer_string>, <model_string><NL>
```

```
<manufacturer_string> ::= quoted string, 1-31 characters
```

```
<model_string> ::= quoted string, 1-10 characters
```

See Also

- "*IDN? — Identification Number" on page 227

History New in version 5.20.

:SYSTem:PRESet

Command `:SYSTem:PRESet [ {DEFault | FACTory} ]`

The :SYSTem:PRESet command initializes the oscilloscope to a known state. You can use these parameters:

- DEFault (or no parameter) — performs a Default Setup just like the oscilloscope's front panel **[Default Setup]** key.
- FACTory — performs a Factory Default.

Default Setup Default Setup returns Infiniium oscilloscope settings, except user preferences and a few other settings, to their defaults.

- Markers, functions, waveforms, bookmarks, and measurements are all turned off in a default setup.
- Multiple waveform windows are closed, leaving only one waveform window in a default setup.
- These are the default settings for the controls that change:

Control	Default Setting
Run/Stop	Run
Channel 1	On, 1 V/div, 0 offset
Horizontal Scale	100 ns/
Horizontal Position	0 s
Reference	Center
Zoom	Disabled
Trigger Mode	Edge
Trigger Level	0 V
Trigger Sweep	Auto
Edge Trigger Source	Channel 1
Edge Trigger Slope	Rising

- Default Setup does not change any of the control settings found in the User Preferences dialog box, display color settings, screen options, probe skew, probe external adapter settings for differential probes, or probe internal attenuation and gain settings for differential probes.

Factory Default The Factory Default selection returns the oscilloscope to the settings it had when it left the factory. This places the oscilloscope in a known operating condition. You can use Factory Default when you want to set all values (even the ones not defaulted by Default Setup) back to their default values.

These controls are reset during a factory default (but are not reset during a Default Setup):

- User Preferences dialog box settings
- Customize Multipurpose settings
- Tabbed window layout
- Digital memory size
- Waveform memories
- Channel skew
- Display colors
- Waveform intensity and grid line intensity settings
- Probe skew
- Probe external adapter settings for differential probes
- Probe internal attenuation and gain setting for differential probes
- Lock Display Results (not selected)

Example This example performs an oscilloscope default setup.

```
myScope.WriteString ":SYSTem:PRESet"
```

See Also

- "*RST — Reset" on page 241

History Legacy command (existed before version 3.10).

:SYSTem:SETup

Command `:SYSTem:SETup <binary_block_data>`

The :SYSTem:SETup command sets up the oscilloscope as defined by the data in the binary block of data from the computer.

CAUTION

Setups saved from Infiniium software versions prior to 2.00 may not load correctly in software versions 4.30 and greater.

You can remedy this by re-saving any pre-2.00 setups using any version of software from version 2.00 to version 4.20.

Setups saved from software versions between 2.00 and 4.20 should load correctly into version 4.30 and greater.

<binary _block_data> A binary block of data, consisting of bytes of setup information. The number of bytes is a dynamic number that is read and allocated by oscilloscope's software.

Example This example reads setup information from a file and restores it to the oscilloscope.

```
' Read setup from a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Dim varSetup As Variant
Open strPath For Binary Access Read As hFile   ' Open file for input.
Get hFile, , varSetup   ' Read data.
Close hFile   ' Close file.

' Write setup to oscilloscope.
myScope.WriteIEEEBlock ":SYSTem:SETup", varSetup
Debug.Print "Setup bytes restored: " + CStr(LenB(varSetup))
```

Query `:SYSTem:SETup?`

The :SYSTem:SETup? query outputs the oscilloscope's current setup to the computer in binary block data format as defined in the IEEE 488.2 standard.

Returned Format `[:SYSTem:SETup] #NX...X<setup_data_string><NL>`

The first character in the setup data block is a number added for disk operations.

Example This example stores the current oscilloscope setup to the variable, varSetup, and then saves it to a file.

```
' Get setup from the oscilloscope.
Dim varSetup As Variant
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SYSTem:SETup?"
varSetup = myScope.ReadIEEEBlock(BinaryType_UI1)

' Output setup string to a file:
```

```
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varSetup   ' Write data.
Close hFile   ' Close file.
Debug.Print "Setup bytes saved: " + CStr(LenB(varSetup))
```

NOTE

:SYSTem:SETup Can Operate Just Like *LRN?

When headers and LONGform are on, the :SYSTem:SETup? query operates the same as the *LRN? query in the common commands. Otherwise, *LRN? and :SYSTem:SETup are not interchangeable.

See Also

- "Definite-Length Block Response Data" on page 123

History

Legacy command (existed before version 3.10).

:SYSTem:TIME

Command `:SYSTem:TIME <hour>,<minute>,<second>`

The :SYSTem:TIME command sets the time in the oscilloscope and is not affected by the *RST common command.

<hour> 0...23

<minute> 0...59

<second> 0...59

Example This example sets the oscilloscope time to 10:30:45 a.m.

```
myScope.WriteString ":SYSTem:TIME 10,30,45"
```

Query `:SYSTem:TIME?`

The :SYSTem:TIME? query returns the current time in the oscilloscope.

Returned Format `[:SYSTem:TIME] <hour>,<minute>,<second>`

History Legacy command (existed before version 3.10).

43 :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X axis) oscilloscope functions.

:TIMebase:POSition

Command

```
:TIMebase:POSition <position_value>
```

The :TIMebase:POSition command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the :TIMebase:REFerence command.

<position _value> A real number for the time in seconds from trigger to the delay reference point.

Example This example sets the delay position to 2 ms.

```
myScope.WriteString ":TIMebase:POSition 2E-3"
```

Query

```
:TIMebase:POSition?
```

The :TIMebase:POSition? query returns the current delay value in seconds.

Returned Format

```
[:TIMebase:POSition] <position_value><NL>
```

Example This example places the current delay value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":TIMebase:POSition?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:RANGe

Command

```
:TIMebase:RANGe <full_scale_range>
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds. The range value is ten times the time-per-division value.

<full_scale_range> A real number for the horizontal time, in seconds. The timebase range is 50 ps (5 ps/div) to 200 s (20 s/div).

Example This example sets the full-scale horizontal range to 10 ms.

```
myScope.WriteString ":TIMebase:RANGe 10E-3"
```

Query

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe? query returns the current full-scale horizontal time.

Returned Format

```
[:TIMebase:RANGe] <full_scale_range><NL>
```

Example This example places the current full-scale horizontal range value in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:RANGe?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:REFClock

Command `:TIMebase:REFClock {{ON | 1} | {OFF | 0}}`

The :TIMebase:REFClock command enables or disables (ON or OFF) the **10 MHz In** BNC input located on the rear panel of the oscilloscope.

The oscilloscope's **10 MHz In** BNC input is used to synchronize (phase lock) the oscilloscope's horizontal timebase system to a reference clock that you provide. This input can also be used to provide a lower phase noise reference than the internal reference so that the oscilloscope's jitter measurement floor is lowered on long (≥10 ms) acquisitions.

The clock you provide must meet the following specifications:

- **Input frequency lock range:** 10 MHz ±20 ppm
- **Amplitude, sine wave input:** 630 mVpp (0 dBm) min to 3.54 Vpp (+15 dBm) max
- **Amplitude, square wave input:** 500 mVpp min to 2.83 Vpp max
- **Input impedance:** 50 Ω (typical)

Notes:

- The oscilloscope will lock to input amplitudes ranging from -5 dBm to +15 dBm (sine wave); however, the best intrinsic jitter performance is achieved between 0 dBm and +15 dBm.
- For inputs at or slightly below -5 dBm, the oscillator assembly will disconnect from the external reference input, the oscilloscope application will automatically connect to the internal reference signal, and a message will appear indicating the external reference signal amplitude is too low and the oscilloscope is now using the internal reference. To continue using the external reference, you need to increase the input amplitude above -5 dBm so the hardware will remain connected to the external reference signal.
- For inputs at or slightly above +15 dBm, the oscillator assembly will disconnect from the external reference input, the oscilloscope application will automatically connect to the internal reference signal, and a message will appear indicating the external reference signal amplitude is too high and the oscilloscope is now using the internal reference. To continue using the external reference, you need to reduce the input amplitude to +15 dBm or less so the hardware will remain connected to the external reference signal.

Example This example turns on the 10 MHz reference clock mode.

```
myScope.WriteString ":TIMebase:REFClock ON"
```

Query `:TIMebase:REFClock?`

The :TIMebase:REFClock? query returns the current state of the reference clock mode control.

Returned Format `[:TIMebase:REFClock] {1 | 0}<NL>`

Example This example places the current value of the reference clock mode control in the variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:REFClock?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

Version 10.00: The HFRequency option is not supported on Keysight UXR Series oscilloscopes.

:TIMebase:REFerence

Command

```
:TIMebase:REFerence {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the .horizontal reference position to the left, center, or right side of the screen.

Example This example sets the horizontal reference position to the center of the display.

```
myScope.WriteString ":TIMebase:REFerence CENTer"
```

Query

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current horizontal reference position.

Returned Format

```
[:TIMebase:REFerence] {LEFT | CENTer | RIGHt | PERCent}<NL>
```

PERC is returned when the horizontal reference position is set to a percent-of-screen location (either in the user interface or with the :TIMebase:REFerence:PERCent command).

Example This example places the current horizontal reference position in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":TIMebase:REFerence?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":TIMebase:REFerence:PERCent" on page 1547

History Legacy command (existed before version 3.10).

Version 5.00: Query can now return PERC when a reference position percent value is being used.

:TIMebase:REFerence:PERCent

Command

```
:TIMebase:REFerence:PERCent <percent>
```

The :TIMebase:REFerence:PERCent command sets the horizontal reference position to a percent-of-screen location, from left to right.

<percent> Integer from 0-100.

Example This example sets the horizontal reference position to a 25% of screen location.

```
myScope.WriteString ":TIMebase:REFerence:PERCent 25"
```

Query

```
:TIMebase:REFerence:PERCent?
```

The :TIMebase:REFerence:PERCent? query returns the current horizontal reference position as a percent-of-screen value.

Returned Format

```
[:TIMebase:REFerence:PERCent] <percent><NL>
```

Example This example places the current horizontal reference position in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":TIMebase:REFerence:PERCent?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":TIMebase:REFerence" on page 1546

History New in version 5.00.

:TIMebase:ROLL:ENABle

Command

```
:TIMebase:ROLL:ENABle {{ON | 1} | {OFF | 0}}
```

The :TIMebase:ROLL:ENABle command enables or disables the roll mode feature.

Example This example tuns on the roll mode.

```
myScope.WriteString ":TIMebase:ROLL:ENABle ON"
```

Query

```
:TIMebase:ROLL:ENABle?
```

The :TIMebase:ROLL:ENABle? query returns the current state of the roll mode enable control.

Returned Format

```
[:TIMebase:ROLL:ENABle] {1 | 0}<NL>
```

Example This example places the current value of the roll mode enable control in the variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:ROLL:ENABle?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:SCALe

Command

```
:TIMebase:SCALe <time>
```

The :TIMebase:SCALe command sets the time base scale. This corresponds to the horizontal scale value displayed as time/div on the oscilloscope screen.

<time> A real number for the time value, in seconds per division. The timebase scale is 5 ps/div to 20 s/div.

Example This example sets the scale to 10 ms/div.

```
myScope.WriteString ":TIMebase:SCALe 10E-3"
```

Query

```
:TIMebase:SCALe?
```

The :TIMebase:SCALe? query returns the current scale time setting.

Returned Format

```
[:TIMebase:SCALe] <time><NL>
```

Example This example places the current scale value in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:SCALe?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:VIEW

Command

```
:TIMebase:VIEW {MAIN | WINDow}
```

The :TIMebase:VIEW command turns the horizontal zoom mode on and off. This is the same as using the front panel **[Zoom]** key.

Example This example turns the horizontal zoom mode on.

```
myScope.WriteString ":TIMebase:VIEW WINDow"
```

Query

```
:TIMebase:VIEW?
```

The :TIMebase:VIEW? query returns the horizontal zoom mode setting.

Returned Format

```
[:TIMebase:VIEW] {MAIN | WINDow}<NL>
```

Example This example places the current view in the string variable, strState, then prints the contents of the variable to the computer's screen.

```
Dim strState As String   ' Dimension variable.
myScope.WriteString ":TIMebase:VIEW?"
strState = myScope.ReadString
Debug.Print strState
```

History Legacy command (existed before version 3.10).

:TIMebase:WINDow:DELay

Command

```
:TIMebase:WINDow:DELay <delay_value>
```

The :TIMebase:WINDow:DELay sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

NOTE

This Command is Provided for Compatibility

This command is the same as the :TIMebase:WINDow:POSition command, and is provided for compatibility with programs written for previous oscilloscopes. The preferred command for compatibility with Infiniium oscilloscopes is :TIMebase:WINDow:POSition.

<delay_value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example This example sets the time base window delay position to 20 ns.

```
myScope.WriteString ":TIMebase:WINDow:DELay 20E-9"
```

Query

```
:TIMebase:WINDow:DELay?
```

The :TIMebase:WINDow:DELay? query returns the current horizontal position in the delayed view.

Returned Format

```
[:TIMebase:WINDow:DELay] <delay_position><NL>
```

Example This example places the current horizontal position in the delayed view in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:WINDow:DELay?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

See Also The :TIMebase:WINDow:POSition command performs the same function as this command and should be used in new programs.

History Legacy command (existed before version 3.10).

:TIMebase:WINDow:POSition

Command `:TIMebase:WINDow:POSition <position_value>`

The :TIMebase:WINDow:POSition sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

<position _value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example This example sets the time base window delay position to 20 ns.

```
myScope.WriteString ":TIMebase:WINDow:POSition 20E-9"
```

Query `:TIMebase:WINDow:POSition?`

The :TIMebase:WINDow:POSition? query returns the current horizontal position in the delayed view.

Returned Format `[:TIMebase:WINDow:POSition] <position_value><NL>`

Example This example places the current horizontal position in the delayed view in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:WINDow:POSition?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:WINDow:RANGe

Command

```
:TIMebase:WINDow:RANGe <full_scale_range>
```

The :TIMebase:WINDow:RANGe command sets the full-scale range of the delayed view. The range value is ten times the time per division of the delayed view. The maximum range of the delayed view is the current main range. The minimum delayed view range is 10 ps (1 ps/div).

<full_scale_range> A real number for the full-scale range of the time base window, in seconds.

Example This example sets the full-scale range of the delayed view to 100 ns.

```
myScope.WriteString ":TIMebase:WINDow:RANGe 100E-9"
```

Query

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current full-scale range of the delayed view.

Returned Format

```
[:TIMebase:WINDow:RANGe] <full_scale_range><NL>
```

Example This example reads the current full-scale range of the delayed view into the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":TIMebase:WINDow:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:TIMebase:WINDow:SCALe

Command `:TIMebase:WINDow:SCALe <time>`

The :TIMebase:WINDow:SCALe command sets the time/div in the delayed view. This command rescales the horizontal components of displayed waveforms.

<time> A real number for the delayed windows scale.

Example This example sets the scale of the time base window to 2 milliseconds/div.

```
myScope.WriteString ":TIMebase:WINDow:SCALe 2E-3"
```

Query `:TIMebase:WINDow:SCALe?`

The :TIMebase:WINDow:SCALe? query returns the scaled window time, in seconds/div.

Returned Format `[:TIMebase:WINDow:SCALe] <time><NL>`

History Legacy command (existed before version 3.10).

44 :TRIGger Commands

The oscilloscope trigger circuitry helps you locate the waveform you want to view. There are several different types of triggering, but the one that is used most often is edge triggering. Edge triggering identifies a trigger condition by looking for the slope (rising or falling) and voltage level (trigger level) on the source you select. Any input channel, auxiliary input trigger, or line can be used as the trigger source.

The commands in the TRIGger subsystem define the conditions for triggering. Many of the commands in the TRIGger subsystem are used in more than one of the trigger modes. The command set has been defined to closely represent the front-panel trigger menus. As a trade-off, there may be less compatibility between Infiniium Oscilloscopes and command sets for previous oscilloscopes. Infiniium Oscilloscopes still accept some commands for compatibility with previous instruments. An alternative command that is accepted by the oscilloscope is noted for a particular command.

Summary of Trigger Modes and Commands

Make sure the oscilloscope is in the proper trigger mode for the command you want to send. One method of ensuring that the oscilloscope is in the proper trigger mode is to send the :TRIGger:MODE command in the same program message as the parameter to be set.

For example, to place the instrument in the proper triggering mode you select:

```
:TRIGger:MODE <Trigger_mode>
```

<Trigger_mode>

The trigger modes include DELay, EDGE, GLITch, PATTern, PWIDth, RUNT, SEQuence, SHOLd, STATe, TIMeout, TRANsition, and WINDow. Each mode is described with its command set in this chapter.

General :TRIGger Commands

:TRIGger:AND:ENABle

Command `:TRIGger:AND[{1 | 2}]:ENABle {{ON | 1} | {OFF | 0}}`

The :TRIGger:AND:ENABle command enables the ability to further qualify the trigger using other channels.

The optional [{1 | 2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:AND:ENABle?`

The query returns the current state of the AND qualifier.

Returned Format `[:TRIGger:AND:ENABle] {1 | 0}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:AND:LTYPe

Command

```
:TRIGger:AND[{1 | 2}]:LTYPe {AND | OR}
```

When the AND (Qualifier) is enabled, the :TRIGger:AND:LTYPe command specifies whether multiple events in the AND (Qualifier) are ANDed or ORed.

- AND — The trigger event is qualified when all of the multiple events are true.
- OR — The trigger event is qualified when any one of the multiple events are true.

The optional [{1 | 2}] parameter sets whether the AND qualifier logic type goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:AND:LTYPe?
```

The :TRIGger:AND:LTYPe? query returns the currently specified AND (Qualifier) logic type setting.

Returned Format

```
<logic_type><NL>

<logic_type> ::= {AND | OR}
```

See Also

- ":TRIGger:AND:ENABle" on page 1558

History

New in version 10.00.

:TRIGger:AND:SOURce

Command

```
:TRIGger:AND[{1 | 2}]:SOURce CHANnel<N>,{HIGH | LOW | DONTcare}
```

The :TRIGger:AND:SOURce command sets the logic value used to qualify the trigger for the specified channel. The TRIGger:LEVel command determines what voltage level is considered a HIGH or a LOW logic value. If you set more than one channel to a HIGH or a LOW, then the multiple channels are used to qualify the trigger.

The optional [{1 | 2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query

```
:TRIGger:AND:SOURce? CHANnel<N>
```

The query returns the logic value for the designated channel.

Returned Format

```
[:TRIGger:AND:SOURce CHANnel<N>] {HIGH | LOW | DONTcare}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:FORCe

Command `:TRIGger:FORCe`

The :TRIGger:FORCe command causes an acquisition to be captured even though the trigger condition has not been met.

See Also

- ":TRIGger:SWEep" on page 1574
- ":TRIGger:LEVel" on page 1569
- ":TRIGger:LEVel:FIFTy" on page 1570

History New in version 10.10.

:TRIGger:HIGH

Command `:TRIGger:HIGH {0 | OFF}`

OFF (0) is the only option available because high-bandwidth trigger is not needed with the Keysight MXR/EXR-Series oscilloscopes.

Query `:TRIGger:HIGH?`

The :TRIGger:HIGH? query returns the high-bandwidth trigger setting.

Returned Format `<setting><NL>`

`<setting> ::= 0`

History New in version 10.00.

Version 11.00: OFF (0) is the only option available because high-bandwidth trigger is not needed with the Keysight MXR/EXR-Series oscilloscopes.

:TRIGger:HOLDoff

Command `:TRIGger:HOLDoff <holdoff_time>`

The :TRIGger:HOLDoff command specifies the amount of time the oscilloscope should wait after receiving a trigger before enabling the trigger again.

<holdoff_time> A real number for the holdoff time, ranging from 100 ns to 10 s.

Query `:TRIGger:HOLDoff?`

The query returns the current holdoff value for the current mode.

Returned Format `[:TRIGger:HOLDoff] <holdoff><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:HOLDoff:MAX

Command

```
:TRIGger:HOLDoff:MAX <holdoff_time>
```

This command is only used when you set the :TRIGger:HOLDoff:MODe command to RANDom. The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random 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 increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

<holdoff_time> A real number for the maximum random holdoff time.

Query

```
:TRIGger:HOLDoff:MAX?
```

The query returns the current maximum holdoff value for the random holdoff mode.

Returned Format

```
[:TRIGger:HOLDoff:MAX] <holdoff><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:HOLDoff:MIN

Command

```
:TRIGger:HOLDoff:MIN <holdoff_time>
```

This command is only used when you set the :TRIGger:HOLDoff:MODe command to RANDom. The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random 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 increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

<holdoff_time> A real number for the minimum random holdoff time.

Query

```
:TRIGger:HOLDoff:MIN?
```

The query returns the current minimum holdoff value for the random holdoff mode.

Returned Format

```
[:TRIGger:HOLDoff:MIN] <holdoff><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:HOLDoff:MODE

Command `:TRIGger:HOLDoff:MODE {FIXed | RANDom}`

The Fixed mode sets the amount of time that the oscilloscope waits before re-arming the trigger circuitry. It can be used to stabilize the display of complex waveforms.

The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random 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 increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

NOTE **The RANDom holdoff mode is not available when the "high-bandwidth trigger" setting is enabled (see :TRIGger:HIGH).**

Query `:TRIGger:HOLDoff:MODE?`

The query returns the current holdoff mode.

Returned Format `[:TRIGger:HOLDoff:MODE] {FIXed | RANDom}<NL>`

See Also

- ":TRIGger:HIGH" on page 1562

History Legacy command (existed before version 3.10).

:TRIGger:HTHReshold

Command

```
:TRIGger:HTHReshold {{CHANnel<N> | AUXiliary},<level>}
```

This command specifies the high threshold voltage level for the selected trigger source. Set the high threshold level to a value considered to be a high level for your logic family; your data book gives two values, V_{IH} and V_{OH}.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage level for the trigger source.

Query

```
:TRIGger:HTHReshold? {CHANnel<N> | AUXiliary}
```

The query returns the currently defined high threshold voltage level for the trigger source.

Returned Format

```
[:TRIGger:HTHReshold {CHANnel<N> | AUXiliary},] <level><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:HYSTeresis

Command `:TRIGger:HYSTeresis {NORMal | NREJect | HSENsitivity}`

The :TRIGger:HYSTeresis command specifies the trigger hysteresis (noise reject) as:

- NORMal — the typical hysteresis selection.
- NREJect (noise reject) — gives maximum hysteresis but the lowest trigger bandwidth.
- HSENsitivity — lowers the hysteresis of the trigger circuitry and should be used for waveforms of 4 GHz and above.

Query `:TRIGger:HYSTeresis?`

The query returns the current hysteresis setting.

Returned Format `[:TRIGger:HYSTeresis] {NORMal | NREJect | HSENsitivity}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:LEVel

Command

```
:TRIGger:LEVel {{CHANnel<N> | AUX},<level>}
```

The :TRIGger:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialog boxes (Edge, Glitch, and Advanced). This level also applies to all the High Threshold (HTHReshold) values in the Advanced Violation menus.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the trigger level on the specified channel or Auxiliary Trigger Input.

Query

```
:TRIGger:LEVel? {CHANnel<N> | AUX}
```

The query returns the specified channel's trigger level.

Returned Format

```
[:TRIGger:LEVel {CHANnel<N> | AUX},] <level><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:LEVel:FIFTy

Command `:TRIGger:LEVel:FIFTy`

The :TRIGger:LEVel:FIFTy command sets the trigger level to 50%.

This performs the same action as the "push for 50%" front panel trigger level knob.

See Also

- ":TRIGger:LEVel" on page 1569

History New in version 4.30.

:TRIGger:LTHReshold

Command `:TRIGger:LTHReshold CHANnel<N>,<level>`

This command specifies the low threshold voltage level for the selected trigger source. This command specifies the low threshold voltage level for the selected trigger source. Set the low threshold level to a value considered to be a low level for your logic family; your data book gives two values, V_{IL} and V_{OL}.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage level for the trigger source.

Query `:TRIGger:LTHReshold? CHANnel<N>`

The query returns the currently defined low threshold for the trigger source.

Returned Format `[:TRIGger:LTHReshold CHANnel<N>,] <level><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:MODE

Command

```
:TRIGger:MODE {EDGE | GLITch | PWIDth | PATTern | STATe | RUNT | SHOLd
    | TRANsition | DELay | TIMeout | WINDow | OR | EBURst | SEQuence
    | SBUS<N> | IFMagn}
```

The :TRIGger:MODE command selects the trigger mode.

Table 17 :TRIGger:MODE Settings

Mode	Definition
DELay	Delay by Events mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. Delay by Time mode lets you view pulses in your waveform that occur a long time after a specified waveform edge.
EBURst	Nth Edge Burst trigger mode.
EDGE	Edge trigger mode.
GLITch	Trigger on a pulse that has a width less than a specified amount of time.
IFMagn	Available on MXR-Series oscilloscopes, trigger on Spectrum Analysis (DDC) IF magnitude edges or levels.
OR	ORed Edges trigger mode.
PATTern	Pattern triggering lets you trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.
PWIDth	Pulse width triggering lets you trigger on a pulse that is greater than or less than a specified width and of a certain polarity.
RUNT	Runt triggering lets you trigger on positive or negative pulses that are smaller in amplitude than other pulses in your waveform.
SBUS<N>	Serial triggering on SBUS1, SBUS2, SBUS3, or SBUS4.
SEQuence	Sequential triggering lets you use multiple events or time/pattern qualifications to define your trigger.
SHOLd	Setup and Hold triggering let you trigger on Setup or Hold violations in your circuit.
STATe	State triggering lets you set the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.
TIMeout	Timeout triggering lets you trigger when the waveform remains high too long, low to long, or unchanged too long.

Table 17 :TRIGger:MODE Settings (continued)

Mode	Definition
TRANsition	Edge Transition triggering lets you trigger on an edge that violates a rise time or fall time specification.
WINDow	Window triggering lets you define a window on screen and then trigger when the waveform exits the window, enters it, or stays inside/outside the window for too long/short.

Query

```
:TRIGger:MODE?
```

The query returns the currently selected trigger mode.

Returned Format

```
[:TRIGger:MODE] {EDGE | GLIT | PWID | PATT | STAT | RUNT | SHOLd
   | TRAN | DELay | TIM | WIND | OR | EBUR | COMM | SEQ | SBUS<N>
   | IFM}<NL>
```

History Legacy command (existed before version 3.10).

Version 3.50: Added the SBUS1, SBUS2, SBUS3, and SBUS4 selections for triggering on serial buses.

Version 10.00: The OR and EBURst options are added. The COMM and TV options are removed. The advanced trigger mode and commands have been deprecated. The advanced COMM and TV trigger options are removed.

Version 11.20: Available on MXR-Series oscilloscopes, the IFMagn option is added.

:TRIGger:SWEep

Command `:TRIGger:SWEep {AUTO | TRIGgered | SINGle}`

The :TRIGger:SWEep command selects the oscilloscope sweep mode. New programs should use :RUN and :SINGle for run control and this command for AUTO and TRIGgered for sweep control. The SINGle sweep control should not be used.

AUTO When you select AUTO, if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 50 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.

TRIGgered When you select TRIGgered, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen.

SINGle When you select SINGle, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen. Do not use in new programs.

Query `:TRIGger:SWEep?`

The query returns the specified channel's trigger level.

Returned Format `[:TRIGger:SWEep] {AUTO | TRIGgered}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay (Edge Then Edge Trigger) Commands

:TRIGger:DELay:ARM:SLOPe

Command `:TRIGger:DELay:ARM:SLOPe {NEGative | POSitive}`

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

Query `:TRIGger:DELay:ARM:SLOPe?`

The query returns the currently defined slope for the Delay trigger mode.

Returned Format `[:TRIGger:DELay:ARM:SLOPe] {NEGative | POSitive}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:ARM:SOURce

Command `:TRIGger:DELay:ARM:SOURce {CHANnel<N>}`

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:DELay:ARM:SOURce?`

The query returns the currently defined Arm On source for the Delay trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:ARM:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:EDELay:COUNt

Command `:TRIGger:DELay:EDELay:COUNt <edge_number>`

This command sets the event count for a Delay By Event trigger event.

<edge_num> An integer from 0 to 65,000,000,000 specifying the number of edges to delay.

Query `:TRIGger:DELay:EDELay:COUNt?`

The query returns the currently defined number of events to delay before triggering on the next Trigger On condition in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:COUNt] <edge_number><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:EDELay:SLOPe

Command `:TRIGger:DELay:EDELay:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay By Event trigger event.

Query `:TRIGger:DELay:EDELay:SLOPe?`

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SLOPe] {NEGative | POSitive}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:EDELay:SOURce

Command `:TRIGger:DELay:EDELay:SOURce {CHANnel<N>}`

This command sets the Event source for a Delay By Event trigger event.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:DELay:EDELay:SOURce?`

The query returns the currently defined Event source in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:MODE

Command `:TRIGger:DELay:MODE {EDELay | TDELay}`

The :TRIGger:DELay:MODE command selects the type of delay trigger mode to either events or to time.

Query `:TRIGger:DELay:MODE?`

The query returns the currently selected delay trigger mode.

Returned Format `[:TRIGger:DELay:MODE] {EDELay | TDELay}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:TDELay:TIME

Command `:TRIGger:DELay:TDELay:TIME <delay>`

This command sets the delay for a Delay By Time trigger event.

<delay> Time, in seconds, set for the delay trigger, from 10 ns to 10 s.

Query `:TRIGger:DELay:TDELay:TIME?`

The query returns the currently defined time delay before triggering on the next Trigger On condition in the Delay By Time trigger mode.

Returned Format `[:TRIGger:DELay:TDELay:TIME] <delay><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:TRIGger:COUNt

Command

```
:TRIGger:DELay:TRIGger:COUNt <edge_count>
```

The :TRIGger:DELay:TRIGger:COUNt command specifies the Nth edge to trigger on.

<edge_count> An integer from 1 to 65,000,000,000 in NR1 format.

Query

```
:TRIGger:DELay:TRIGger:COUNt?
```

The :TRIGger:DELay:TRIGger:COUNt? query returns the currently specified edge count.

Returned Format

```
<edge_count><NL>
```

```
<edge_count> ::= an integer from 1 to 65,000,000,000 in NR1 format.
```

See Also

- ":TRIGger:DELay:TRIGger:SOURce" on page 1585
- ":TRIGger:DELay:TRIGger:SLOPe" on page 1584

History New in version 10.00.

:TRIGger:DELay:TRIGger:SLOPe

Command `:TRIGger:DELay:TRIGger:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay trigger event.

Query `:TRIGger:DELay:TRIGger:SLOPe?`

The query returns the currently defined slope for an event in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SLOPe] {NEGative | POSitive}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:DELay:TRIGger:SOURce

Command `:TRIGger:DELay:TRIGger:SOURce {CHANnel<N>}`

This command sets the Trigger On source for a Delay trigger event.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:DELay:TRIGger:SOURce?`

The query returns the currently defined Trigger On source in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:EBURst (Burst Trigger) Commands

- ":TRIGger:EBURst:COUNt" on page 1587
- ":TRIGger:EBURst:IDLE" on page 1588
- ":TRIGger:EBURst:SLOPe" on page 1589
- ":TRIGger:EBURst:SOURce" on page 1590

In the Burst trigger mode, you can specify one source (an analog input channel), one slope (positive or negative), a count value (1 to 65 billion), and a minimum idle time. The oscilloscope will trigger when the minimum idle time has been satisfied (no edges for the specified duration) and then the edge count on the specified source is found.

:TRIGger:EBURst:COUNt

Command

```
:TRIGger:EBURst[{1 | 2}]:COUNt <edge_number>
```

The :TRIGger:EBURst:COUNt command specifies the Burst trigger edge number. The edge number specifies which edge in a burst will generate a trigger.

<edge_number> An integer from 1 to 65,000,000,000 in NR1 format.

Query

```
:TRIGger:EBURst:COUNt?
```

The :TRIGger:EBURst:COUNt? query returns the currently specified Burst trigger edge number.

Returned Format

```
[:TRIGger:EBURst:COUNt] <edge_count><NL>
```

See Also

- ":TRIGger:EBURst:IDLE" on page 1588
- ":TRIGger:EBURst:SLOPe" on page 1589
- ":TRIGger:EBURst:SOURce" on page 1590

History New in version 10.00.

:TRIGger:EBURst:IDLE

Command

```
:TRIGger:EBURst[{1 | 2}]:IDLE <min_time>
```

The :TRIGger:EBURst:IDLE command specifies the Burst trigger idle time. The timer is used to set the minimum time before the next burst.

<min_time> Minimum idle time in NR3 format.

Query

```
:TRIGger:EBURst:IDLE?
```

The :TRIGger:EBURst:IDLE? query returns the currently specified Burst trigger idle time.

Returned Format

```
[:TRIGger:EBURst:IDLE] <min_time><NL>
```

See Also

- ":TRIGger:EBURst:COUNt" on page 1587
- ":TRIGger:EBURst:SLOPe" on page 1589
- ":TRIGger:EBURst:SOURce" on page 1590

History New in version 10.00.

:TRIGger:EBURst:SLOPe

Command

```
:TRIGger:EBURst[{1 | 2}]:SLOPe {NEGative | POSitive}
```

The :TRIGger:EBURst:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.

Query

```
:TRIGger:EBURst:SLOPe?
```

The :TRIGger:EBURst:SLOPe? query returns the currently specified Burst trigger slope.

Returned Format

```
[:TRIGger:EBURst:SLOPe] <edge_dir><NL>
```

```
<edge_dir> ::= {NEG | POS}
```

See Also

- ":TRIGger:EBURst:COUNt" on page 1587
- ":TRIGger:EBURst:IDLE" on page 1588
- ":TRIGger:EBURst:SOURce" on page 1590

History New in version 10.00.

:TRIGger:EBURst:SOURce

Command

```
:TRIGger:EBURst[{1 | 2}]:SOURce {CHANnel<N>}
```

The :TRIGger:EBURst:SOURce command specifies the analog channel input source of the Burst trigger.

<N> 1 to (# analog channels) in NR1 format.

Query

```
:TRIGger:EBURst:SOURce?
```

The :TRIGger:EBURst:SOURce? query returns the currently specified Burst trigger source.

Returned Format

```
[:TRIGger:EBURst:SOURce] <source><NL>

<source> ::= {CHANnel<N>}
```

See Also

- ":TRIGger:EBURst:COUNt" on page 1587
- ":TRIGger:EBURst:IDLE" on page 1588
- ":TRIGger:EBURst:SLOPe" on page 1589

History New in version 10.00.

:TRIGger:EDGE (Edge Trigger) Commands

:TRIGger:EDGE:COUPling

Command

```
:TRIGger:EDGE:COUPling {AC | DC | LFReject | HFReject}
```

When the EDGE trigger mode is selected (see :TRIGger:MODE), the :TRIGger:EDGE:COUPling command sets the trigger coupling when :TRIGger:EDGE:SOURce is set to one of the analog input channels.

Query

```
:TRIGger:EDGE:COUPling?
```

The query returns the currently selected coupling for the specified edge trigger source.

Returned Format

```
[:TRIGger:EDGE:COUPling] {AC | DC | LFReject | HFReject}<NL>
```

See Also

- ":TRIGger:MODE" on page 1572
- ":TRIGger:EDGE:SOURce" on page 1594
- ":TRIGger:EDGE:SLOPe" on page 1593

History Legacy command (existed before version 3.10).

:TRIGger:EDGE:SLOPe

Command `:TRIGger:EDGE[{1 | 2}]:SLOPe {POSitive | NEGative | EITHer | ALTernate}`

The :TRIGger:EDGE:SLOPe command sets the slope of the trigger source previously selected by the :TRIGger:EDGE:SOURce command.

THE ALTernate option specifies that the oscilloscope will trigger on alternating rising and falling edges. For example, the oscilloscope will be set up for rising edge trigger, then immediately after a trigger occurs it will be reconfigured for falling edge trigger, then immediately after a trigger occurs it will be reconfigured for rising edge again, and so on.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:EDGE:SLOPe?`

The query returns the currently selected slope for the specified edge trigger source.

Returned Format `[:TRIGger:EDGE:SLOPe] {POS | NEG | EITH | ALT}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: The ALTernate parameter has been added to specify alternating rising and falling edges.

:TRIGger:EDGE:SOURce

Command

```
:TRIGger:EDGE[{1 | 2}]:SOURce {CHANnel<N> | DIGital<M> | AUXiliary | LIN
E}
```

The :TRIGger:EDGE:SOURce command selects the source for edge mode triggering. This is the source that will be used for subsequent :TRIGger:EDGE:SLOPe commands or queries.

NOTE **The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used. Sequential triggering is available on MXR/EXR-Series oscilloscopes.**

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15.

Query

```
:TRIGger:EDGE:SOURce?
```

The query returns the currently selected edge mode trigger source.

Returned Format

```
[:TRIGger:EDGE:SOURce] {CHAN<N> | DIG<M> | AUX | LINE}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:GLITch (Glitch Trigger) Commands

:TRIGger:GLITch:POLarity

Command `:TRIGger:GLITch[{1 | 2}]:POLarity {POSitive | NEGative}`

This command defines the polarity of the glitch as positive or negative. The trigger source must be set using the :TRIGger:GLITch:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:GLITch:POLarity?`

The query returns the currently selected glitch polarity.

Returned Format `[:TRIGger:GLITch:POLarity] {POS | NEG}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:GLITch:SOURce

Command `:TRIGger:GLITch[{1 | 2}]:SOURce {CHANnel<N>}`

This command sets the source for the glitch trigger mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:GLITch:SOURce?`

The query returns the currently selected source for the glitch trigger mode.

Returned Format `[:TRIGger:GLITch:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:GLITch:WIDTh

Command `:TRIGger:GLITch[{1 | 2}]:WIDTh <width>`

This command sets the glitch width. The oscilloscope will trigger on a pulse that has a width less than the specified width.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<width> A real number for the glitch width, ranging from the minimum detectable pulse width to 10 s.

Query `:TRIGger:GLITch:WIDTh?`

The query returns the currently specified glitch width.

Returned Format `[:TRIGger:GLITch:WIDTh] <width><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:IFMagn (IF Magnitude Trigger) Commands

Commands for the IF Magnitude trigger mode:

When the **Spectrum Analysis (DDC)** signal type is selected (:ANALyze:SIGNal:TYPE SPECtral), you can use the IFMagn trigger mode (:TRIGger:MODE IFMagn) to trigger on IF magnitude edges or levels in the complex IQ digitally down-converted (DDC) data.

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SLOPe" on page 1604
- ":TRIGger:IFMagn:SOURce" on page 1605

The IF Magnitude trigger acts as a filter on the digitally down-converted IQ data that is sent to the oscilloscope's FFT (and displayed) or sent to the PathWave Vector Signal Analysis (89600 VSA) software. Only when the trigger condition occurs are acquisitions are captured and stored.

While the complex IQ digitally down-converted time-domain data waveform is not displayed in the oscilloscope, it can be displayed in the VSA software; this display can be helpful in determining the dBm levels to trigger on.

How Other Trigger Controls are Affected in the IF Magnitude Mode

- The trigger sweep setting is still valid.
- Trigger conditioning holdoff settings still apply.
- Trigger conditioning sensitivity settings do not apply.
- Trigger actions still apply.
- Trigger thresholds settings do not apply.
- Sequence triggering is not allowed.

See Also

- ":ANALyze:SIGNal:TYPE" on page 404
- ":TRIGger:MODE" on page 1572

:TRIGger:IFMagn:HYSTeresis

Command

```
:TRIGger:IFMagn:HYSTeresis <source>[,<hysteresis>]
```

When :TRIGger:IFMagn:MODE EDGE is selected, the :TRIGger:IFMagn:HYSTeresis command provides some noise immunity by specifying how much the IF magnitude must change (in dB) before the trigger occurs.

When :TRIGger:IFMagn:SLOPe POSitive is selected, the IF magnitude must have been below the specified level by the :TRIGger:IFMagn:HYSTeresis amount before the trigger occurs.

Likewise, when :TRIGger:IFMagn:SLOPe NEGative is selected, the IF magnitude must have been above the specified level by the :TRIGger:IFMagn:HYSTeresis amount before the trigger occurs.

<source> {CHANnel<N>}

<N> 1 to (# analog channels) in NR1 format.

<hysteresis> dB value in NR3 format. If <hysteresis> is not included, the command tries to set a level of 0.0 dB.

Query

```
:TRIGger:IFMagn:HYSTeresis? <source>
```

The :TRIGger:IFMagn:HYSTeresis? query returns the edge trigger hysteresis value in dB.

Returned Format

```
<hysteresis><NL>

<hysteresis> ::= dB value in NR3 format
```

See Also

- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SLOPe" on page 1604
- ":TRIGger:IFMagn:SOURce" on page 1605

History New in version 11.10.

:TRIGger:IFMagn:LEVel

Command

```
:TRIGger:IFMagn:LEVel <source>[,<level>]
```

The :TRIGger:IFMagn:LEVel command specifies the IF magnitude level (in dBm) of the trigger.

<source> {CHANnel<N>}

<N> 1 to (# analog channels) in NR1 format.

<level> dBm value in NR3 format. If <level> is not included, the command tries to set a level of 0.0 dBm.

Query

```
:TRIGger:IFMagn:LEVel? <source>
```

The :TRIGger:IFMagn:LEVel? query returns the IF magnitude level (in dBm) of the trigger.

Returned Format

```
<level><NL>

<level> ::= dBm value in NR3 format
```

See Also

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SLOPe" on page 1604
- ":TRIGger:IFMagn:SOURce" on page 1605

History New in version 11.10.

:TRIGger:IFMagn:MODE

Command

```
:TRIGger:IFMagn:MODE <mode>
```

The :TRIGger:IFMagn:MODE command selects the IF Magnitude trigger mode.

<mode> {EDGE | LEVel}

- EDGE — Triggers occur when an increasing (rising) or decreasing (falling) IF magnitude transitions through the specified level.

  In the EDGE mode, the :TRIGger:IFMagn:HYSTeresis value provides some noise immunity by specifying how much the IF magnitude must change (in dB) before the trigger occurs.

  When :TRIGger:IFMagn:SLOPe POSitive is selected, the IF magnitude must have been below the specified level by the :TRIGger:IFMagn:HYSTeresis amount before the trigger occurs.

  Likewise, when :TRIGger:IFMagn:SLOPe NEGative is selected, the IF magnitude must have been above the specified level by the :TRIGger:IFMagn:HYSTeresis amount before the trigger occurs.

  Level crossings are required for triggers to occur.
- LEVel — Triggers occur repeatedly if the IF magnitude is above or below the specified level.

  Level crossings are not required for triggers to occur.

Query

```
:TRIGger:IFMagn:MODE?
```

The :TRIGger:IFMagn:MODE? query returns the IF Magnitude trigger mode selection.

Returned Format

```
<mode><NL>

<mode> ::= {EDGE | LEV}
```

See Also

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SLOPe" on page 1604
- ":TRIGger:IFMagn:SOURce" on page 1605

History New in version 11.10.

:TRIGger:IFMagn:POLarity

Command `:TRIGger:IFMagn:POLarity <polarity>`

When :TRIGger:IFMagn:MODE LEVel is selected, the :TRIGger:IFMagn:POLarity command specifies whether to trigger when the IF Magnitude is above the level or below the level.

<polarity> {POSitive | NEGative}

Query `:TRIGger:IFMagn:POLarity?`

The :TRIGger:IFMagn:POLarity? query returns the above level or below level setting.

Returned Format `<polarity><NL>`

`<polarity> ::= {POS | NEG}`

See Also

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:SLOPe" on page 1604
- ":TRIGger:IFMagn:SOURce" on page 1605

History New in version 11.10.

:TRIGger:IFMagn:SLOPe

Command `:TRIGger:IFMagn:SLOPe <edge_dir>`

When :TRIGger:IFMagn:MODE EDGE is selected, the :TRIGger:IFMagn:SLOPe command specifies the direction of the IF Magnitude edge to trigger on.

<edge_dir> {NEGative | POSitive}

Query `:TRIGger:IFMagn:SLOPe?`

The :TRIGger:IFMagn:SLOPe? query returns the edge direction.

Returned Format `<edge_dir><NL>`

`<edge_dir> ::= {NEG | POS}`

See Also

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SOURce" on page 1605

History New in version 11.10.

:TRIGger:IFMagn:SOURce

Command

```
:TRIGger:IFMagn:SOURce <source>
```

The :TRIGger:IFMagn:SOURce command selects the analog input channel on which to trigger.

<source> {CHANnel<N>}

<N> 1 to (# analog channels) in NR1 format.

Query

```
:TRIGger:IFMagn:SOURce?
```

The :TRIGger:IFMagn:SOURce? query returns the analog input channel selected.

Returned Format

```
<source><NL>

<source> ::= {CHAN<N>}
```

See Also

- ":TRIGger:IFMagn:HYSTeresis" on page 1600
- ":TRIGger:IFMagn:LEVel" on page 1601
- ":TRIGger:IFMagn:MODE" on page 1602
- ":TRIGger:IFMagn:POLarity" on page 1603
- ":TRIGger:IFMagn:SLOPe" on page 1604

History New in version 11.10.

:TRIGger:NEDGe (Nth Edge Trigger) Commands

- ":TRIGger:NEDGe:COUNt" on page 1607
- ":TRIGger:NEDGe:SLOPe" on page 1608
- ":TRIGger:NEDGe:SOURce" on page 1609

The Nth Edge trigger mode lets sequence triggers count edges. (The Edge Then Edge trigger mode is not allowed with sequence triggers.)

In a sequence trigger, the Nth Edge trigger mode can be selected for the TERM2 state in the sequential trigger (the Trigger (B) state in the Trigger Setup dialog box on the oscilloscope).

In the Nth Edge trigger mode, you can specify one source (an analog input channel), one slope (positive or negative), and a count value (1 to 65 billion). The oscilloscope will trigger when the edge count on the specified source is found.

See Also

- ":TRIGger:SEQuence (Sequence Trigger) Commands" on page 1627

:TRIGger:NEDGe:COUNt

Command

```
:TRIGger:NEDGe[{1 | 2}]:COUNt <edge_count>
```

The :TRIGger:NEDGe:COUNt command specifies the Nth Edge trigger edge number. The edge number specifies which edge will generate a trigger.

<edge_count> An integer from 1 to 65,000,000,000 in NR1 format.

Query

```
:TRIGger:NEDGe:COUNt?
```

The :TRIGger:NEDGe:COUNt? query returns the currently specified Nth Edge trigger edge count number.

Returned Format

```
[:TRIGger:NEDGe:COUNt] <edge_count><NL>
```

See Also

- ":TRIGger:NEDGe (Nth Edge Trigger) Commands" on page 1606
- ":TRIGger:NEDGe:SLOPe" on page 1608
- ":TRIGger:NEDGe:SOURce" on page 1609

History New in version 10.00.

:TRIGger:NEDGe:SLOPe

Command

```
:TRIGger:NEDGe[{1 | 2}]:SLOPe {NEGative | POSitive}
```

The :TRIGger:NEDGe:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge will generate a trigger.

Query

```
:TRIGger:NEDGe:SLOPe?
```

The :TRIGger:NEDGe:SLOPe? query returns the currently specified Nth Edge trigger slope.

Returned Format

```
[:TRIGger:NEDGe:SLOPe] <edge_dir><NL>

<edge_dir> ::= {NEG | POS}
```

See Also

- ":TRIGger:NEDGe (Nth Edge Trigger) Commands" on page 1606
- ":TRIGger:NEDGe:COUNt" on page 1607
- ":TRIGger:NEDGe:SOURce" on page 1609

History New in version 10.00.

:TRIGger:NEDGe:SOURce

Command `:TRIGger:NEDGe[{1 | 2}]:SOURce {CHANnel<N>}`

The :TRIGger:NEDGe:SOURce command specifies the analog channel input source of the Nth Edge trigger.

<N> 1 to (# analog channels) in NR1 format.

Query `:TRIGger:NEDGe:SOURce?`

The :TRIGger:NEDGe:SOURce? query returns the currently specified Nth Edge trigger source.

Returned Format `[:TRIGger:NEDGe:SOURce] <source><NL>`

`<source> ::= {CHANnel<N>}`

See Also

- ":TRIGger:NEDGe (Nth Edge Trigger) Commands" on page 1606
- ":TRIGger:NEDGe:COUNt" on page 1607
- ":TRIGger:NEDGe:SLOPe" on page 1608

History New in version 10.00.

:TRIGger:OR (ORed Edges Trigger) Commands

- ":TRIGger:OR:LOGic" on page 1611

The ORed Edges trigger mode lets you select rising, falling or either edge for each of up to four analog input channels. A trigger event will occur when any of the selected edges are seen by the oscilloscope.

:TRIGger:OR:LOGic

Command

```
:TRIGger:OR[{1 | 2}]:LOGic {CHANnel<N>},
   {RISing | FALLing | DONTcare | EITHer}
```

This command defines the OR trigger logic criteria for a selected channel.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query

```
:TRIGger:OR:LOGic? {CHANnel<N>}
```

The query returns the current OR trigger logic criteria for a selected channel.

Returned Format

```
[:TRIGger:OR:LOGic {CHANnel<N>},]
   {RISing | FALLing | DONTcare | EITHer}<NL>
```

History New in version 10.00.

:TRIGger:PATTern (Pattern Trigger) Commands

:TRIGger:PATTern:CONDition

Command

```
:TRIGger:PATTern[{1 | 2}]:CONDition {
   ENTered
   | EXITed
   | {GT,<time>[,{PEXits | TIMeout}]}
   | {LT,<time>}
   | {RANGe,<gt_time>,<lt_time>}
   | {ORANGe,<gt_time>,<lt_time>}
}
```

This command describes the condition applied to the trigger pattern to actually generate a trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

The RANGe option specifies "inside range", and the ORANGe option specifies "outside range".

<gt_time> The minimum time (greater than time) for the trigger pattern.

<lt_time> The maximum time (less than time) for the trigger pattern.

<time> The time condition, in seconds, for the pattern trigger.

When using the GT (Present >) parameter, the PEXits (Pattern Exits) or the TIMeout parameter controls when the trigger is generated.

Query

```
:TRIGger:PATTern:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:PATTern:CONDition] {
   ENTered
   | EXITed
   | {GT,<time>[,{PEXits | TIMeout}]}
   | {LT,<time>}
   | {RANGe,<gt_time>,<lt_time>}
   | {ORANGe,<gt_time>,<lt_time>}
}<NL>
```

History Legacy command (existed before version 3.10).

Version 6.20: The OR parameter has been added.

Version 10.00: The outside range (ORANGe) option is added. The OR option is removed.

:TRIGger:PATTern:LOGic

Command

```
:TRIGger:PATTern[{1 | 2}]:LOGic {CHANnel<N> | DIGital<M>},
   {HIGH | LOW | DONTcare | RISing | FALLing}
```

This command defines the logic criteria for a selected channel.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:TRIGger:PATTern:LOGic? {CHANnel<N> | DIGital<M>}
```

The query returns the current logic criteria for a selected channel.

Returned Format

```
[:TRIGger:PATTern:LOGic {CHANnel<N> | DIGital<M>},]
   {HIGH | LOW | DONTcare | RISing | FALLing}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:PWIDth (Pulse Width Trigger) Commands

:TRIGger:PWIDth:MODE

Command

```
:TRIGger:PWIDth[{1 | 2}]:MODE <mode>

<mode> ::= {GTHan | LTHan | RANGe | ORANge}
```

The :TRIGger:PWIDth:MODE command lets you look for:

- Pulse width violations that are greater than or less than the time specified by the :TRIGger:PWIDth:WIDTh command.
- Pulse width violations that are inside or outside of a time range specified by the :TRIGger:PWIDth:RANGe command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<mode> {GTHan | LTHan | RANGe | ORANge}

Query

```
:TRIGger:PWIDth:MODE?
```

The :TRIGger:PWIDth:MODE? query returns the currently set pulse width violation trigger mode.

Returned Format

```
[:TRIGger:PWIDth:MODE] <mode><NL>

<mode> ::= {GTH | LTH | RANG | ORAN}
```

See Also

- ":TRIGger:PWIDth:MODE" on page 1616
- ":TRIGger:PWIDth:WIDTh" on page 1621
- ":TRIGger:PWIDth:RANGe" on page 1618

History New in version 10.00.

:TRIGger:PWIDth:POLarity

Command `:TRIGger:PWIDth[{1 | 2}]:POLarity {NEGative | POSitive}`

This command specifies the pulse polarity that the oscilloscope uses to determine a pulse width violation. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:POLarity?`

The query returns the currently defined polarity for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:POLarity] {NEGative | POSitive}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:PWIDth:RANGe

Command `:TRIGger:PWIDth[{1 | 2}]:RANGe <gt_time>,<lt_time>`

When the pulse width violation trigger mode is inside range (RANge) or outside range (ORANge), the :TRIGger:PWIDth:RANGe command specified the time range.

<gt_time> The minimum time bound of the range (greater than time) in NR3 format.

<lt_time> The maximum time bound of the range (less than time) in NR3 format.

Query `:TRIGger:PWIDth:RANGe?`

The :TRIGger:PWIDth:RANGe? query returns the currently specified bounds of the time range.

Returned Format `[:TRIGger:PWIDth:RANGe]  <gt_time>,<lt_time><NL>`

See Also

- ":TRIGger:PWIDth:MODE" on page 1616

History New in version 10.00.

:TRIGger:PWIDth:SOURce

Command `:TRIGger:PWIDth[{1 | 2}]:SOURce {CHANnel<N> | DIGital<M>}`

This command specifies the channel source used to trigger the oscilloscope with the pulse width trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:PWIDth:SOURce?`

The query returns the currently defined channel source for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:PWIDth:TPOint

Command `:TRIGger:PWIDth[{1 | 2}]:TPOint {EPULse | TIMeout}`

This command specifies whether the pulse width trigger should occur at the end of the pulse or at a specified timeout period. This command is only available if the pulse direction is set to GTHan.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:TPOint?`

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:TPOint] {EPULse | TIMeout}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:PWIDth:WIDTh

Command `:TRIGger:PWIDth[{1 | 2}]:WIDTh <width>`

This command specifies how wide a pulse must be to trigger the oscilloscope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<width> Pulse width, which can range from 250 ps to 10 s.

Query `:TRIGger:PWIDth:WIDTh?`

The query returns the currently defined width for the pulse.

Returned Format `[:TRIGger:PWIDth:WIDTh] <width><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:RUNT (Runt Trigger) Commands

:TRIGger:RUNT:POLarity

Command

```
:TRIGger:RUNT[{1 | 2}]:POLarity {POSitive | NEGative}
```

This command defines the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:RUNT:POLarity?
```

The query returns the currently selected runt pulse polarity.

Returned Format

```
[:TRIGger:RUNT:POLarity] {POSitive | NEGative}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:RUNT:QUALified

Command `:TRIGger:RUNT[{1 | 2}]:QUALified {{ON | 1} | {OFF | 0}}`

This command enables the time qualified runt pulse feature the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:RUNT:QUALified?`

The query returns the current state of the time qualified runt pulse feature.

Returned Format `[:TRIGger:RUNT:QUALified] {1 | 0}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:RUNT:SOURce

Command

```
:TRIGger:RUNT[{1 | 2}]:SOURce CHANnel<N>
```

This command sets the source for the runt trigger mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query

```
:TRIGger:RUNT:SOURce?
```

The query returns the currently selected source for the runt trigger mode.

Returned Format

```
[:TRIGger:RUNT:SOURce] CHANnel<N><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:RUNT:TIME

Command

```
:TRIGger:RUNT[{1 | 2}]:TIME <time>
```

This command sets the time qualifier. The oscilloscope will trigger on a runt pulse that has a width greater than the specified time.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> A real number for the time greater than qualifier, ranging from 250 ps to 30 ns.

Query

```
:TRIGger:RUNT:TIME?
```

The query returns the currently specified glitch width.

Returned Format

```
[:TRIGger:RUNT:TIME] <time><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence (Sequence Trigger) Commands

The sequence trigger commands are available on MXR/EXR-Series oscilloscopes.

:TRIGger:SEQuence:TERM1

Command `:TRIGger:SEQuence:TERM1 { EDGE1 | GLITch1 | PWIDth1 | PATTern1 | RUNT1 | SHOLd1 | STATe1 | TIMeout1 | TRANsition1 | WINDow1 | OR1 | EBURst1}`

This command specifies the trigger mode for the TERM1 state in the sequential trigger (the Find (A) state in the Trigger Setup dialog box on the oscilloscope).

For the Pattern/State trigger mode, use STATe1 when there is an edge in the pattern; otherwise, use PATTern1 when there are only zeros, ones, and don't cares in the pattern.

There are five limitations associated with sequential triggering:

1. The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.
2. The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.
3. The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.
4. You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.
5. The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query `:TRIGger:SEQuence:TERM1?`

The query returns the currently defined trigger mode for the TERM1 state.

History Legacy command (existed before version 3.10).

Version 10.00: The OR1 and EBURst1 options are added.

:TRIGger:SEQuence:TERM2

Command

```
:TRIGger:SEQuence:TERM2 { EDGE2 | GLITch2 | PWIDth2 | PATTern2 | RUNT2
    | SHOLd2 | STATe2 | TIMeout2 | TRANsition2 | WINDow2 | OR2 | NEDGe2
    | EBURst2}
```

This command specifies the trigger mode for the TERM2 state in the sequential trigger (the Trigger (B) state in the Trigger Setup dialog box on the oscilloscope).

For the Pattern/State trigger mode, use STATe2 when there is an edge in the pattern; otherwise, use PATTern2 when there are only zeros, ones, and don't cares in the pattern.

There are five limitations associated with sequential triggering:

1. The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.
2. The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.
3. The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.
4. You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.
5. The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query

```
:TRIGger:SEQuence:TERM2?
```

The query returns the currently defined trigger mode for the TERM2 state.

History Legacy command (existed before version 3.10).

Version 10.00: The OR2, NEDGe2, and EBURst2 options are added.

:TRIGger:SEQuence:RESet:ENABle

Command `:TRIGger:SEQuence:RESet:ENABle {{ON | 1} | {OFF | 0}}`

This command turns the Reset feature on or off for the sequential trigger.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

If the Reset feature is enabled, the AND qualifier cannot be used for the TERM1 state.

Query `:TRIGger:SEQuence:RESet:ENABle?`

The query returns whether or not the Reset feature is enabled.

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence:RESet:TYPE

Command

```
:TRIGger:SEQuence:RESet:TYPE { TIME | EVENt }
```

This command specifies whether the Reset condition is based upon a length of time or a logical pattern.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

Query

```
:TRIGger:SEQuence:RESet:TYPE?
```

The query returns whether the Reset condition is based upon a length of time or an event.

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence:RESet:EVENt

Command `:TRIGger:SEQuence:RESet:EVENt {CHANnel<N>}, { HIGH | LOW | DONTcare }`

This command defines the logical pattern used for an event Reset condition.

You can specify for each channel whether you want the value to be high (1), low (0), or you don't care (X).

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:SEQuence:RESet:EVENt? {CHANnel<N>}`

The query returns the logical pattern used for an event Reset condition.

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence:RESet:EVENt:LTYPe

Command

```
:TRIGger:SEQuence:RESet:EVENt:LTYPe {AND | OR}
```

The :TRIGger:SEQuence:RESet:EVENt:LTYPe command specifies whether multiple events that reset the sequence are ANDed or ORed.

- AND — The sequence reset occurs when all of the multiple events are true.
- OR — The sequence reset occurs when any one of the multiple events are true.

Query

```
:TRIGger:SEQuence:RESet:EVENt:LTYPe?
```

The :TRIGger:SEQuence:RESet:EVENt:LTYPe? query returns the currently specified event reset logic type setting.

Returned Format

```
<logic_type><NL>

<logic_type> ::= {AND | OR}
```

See Also

- ":TRIGger:SEQuence:RESet:EVENt" on page 1632

History

New in version 10.00.

:TRIGger:SEQuence:RESet:TIME

Command `:TRIGger:SEQuence:RESet:TIME <time>`

This command defines the length of time to use for the time-based Reset condition.

<time> A length of time in seconds.

Query `:TRIGger:SEQuence:RESet:TIME?`

The query returns the length of time used for the Reset condition.

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence:WAIT:ENABle

Command

```
:TRIGger:SEQuence:WAIT:ENABle { {ON|1} | {OFF|0}}
```

This command turns the Delay feature on or off for the sequential trigger.

The Delay feature allows you to define a length of time for the sequential trigger system to wait after the TERM1 event occurs before it starts searching for the TERM2 event.

Query

```
:TRIGger:SEQuence:WAIT:ENABle?
```

The query returns whether or not the Delay feature is turned on.

History Legacy command (existed before version 3.10).

:TRIGger:SEQuence:WAIT:TIME

Command `:TRIGger:SEQuence:WAIT:TIME <time>`

This command defines the length of time to use for the Delay condition.

<time> A length of time in seconds.

Query `:TRIGger:SEQuence:WAIT:TIME?`

The query returns the length of time used for the Delay condition.

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd (Setup and Hold Trigger) Commands

:TRIGger:SHOLd:CSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:CSOurce CHANnel<N>`

This command specifies the clock source for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:SHOLd:CSOurce?`

The query returns the currently defined clock source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:CSOurce] CHANnel<N><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd:CSOurce:EDGE

Command `:TRIGger:SHOLd[{1 | 2}]:CSOurce:EDGE {RISing | FALLing}`

This command specifies the clock source trigger edge for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:SHOLd:CSOurce:EDGE?`

The query returns the currently defined clock source edge for the trigger setup and hold violation level for the clock source.

Returned Format [:TRIGger:SHOLd:CSOurce:EDGE] {RISing | FALLing}<NL>

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd:DSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:DSOurce CHANnel<N>`

The data source commands specify the data source for the trigger setup and hold violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:SHOLd:DSOurce?`

The query returns the currently defined data source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:DSOurce] CHANnel<N><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd:HoldTIMe (HTIMe)

Command `:TRIGger:SHOLd[{1 | 2}]:HoldTIMe <time>`

This command specifies the amount of hold time used to test for both a setup and hold trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> Hold time, in seconds.

Query `:TRIGger:SHOLD:HoldTIMe?`

The query returns the currently defined hold time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLD:HoldTIMe] <time><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd:MODE

Command `:TRIGger:SHOLd[{1 | 2}]:MODE {SETup | HOLD | SHOLd}`

SETup When using the setup time mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the trigger level thresholds during this time window. If the waveform crosses a threshold during this time window, a violation event occurs and the oscilloscope triggers.

HOLD When using the hold time mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

SHOLd When using the setup and hold time mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers. The total time allowed for the sum of setup time plus hold time is 24 ns maximum.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:SHOLd:MODE?`

The query returns the currently selected trigger setup violation mode.

Returned Format `[:TRIGger:SHOLd:MODE] {SETup | HOLD | SHOLd}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:SHOLd:SetupTIMe

Command `:TRIGger:SHOLd[{1 | 2}]:SetupTIMe <time>`

This command specifies the amount of setup time used to test for both a setup and hold trigger violation. The setup time is the amount of time that the data must be stable and valid before a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> Setup time, in seconds.

Query `:TRIGger:SHOLd:SetupTIMe?`

The query returns the currently defined setup time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLd:SetupTIMe] <time><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:STATe (State Trigger) Commands

:TRIGger:STATe:CLOCk

Command

```
:TRIGger:STATe[{1 | 2}]:CLOCk {CHANnel<N> | DIGital<M>}
```

This command selects the source for the clock waveform in the State Trigger Mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used. Sequential triggering is available on 90000A Series, 90000 X-Series, V-Series, 90000 Q-Series, and Z-Series oscilloscopes.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:TRIGger:STATe:CLOCk?
```

The query returns the currently selected clock source.

Returned Format

```
[:TRIGger:STATe:CLOCk] {CHANnel<N> | DIGital<M>}<NL>
```

See Also

- ":TRIGger:STATe:SLOPe" on page 1648
- ":TRIGger:STATe:LOGic" on page 1646

History Legacy command (existed before version 3.10).

:TRIGger:STATe:LOGic

Command

```
:TRIGger:STATe[{1 | 2}]:LOGic {CHANnel<N> | DIGital<M>},
   {LOW | HIGH | DONTcare | RISing | FALLing | EITHer}
```

This command defines the logic state of the specified source for the state pattern.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

The RISing, FALLing, or EITHer options will make the specified source the clock source. In mixed-signal (MSO) oscilloscopes, the EITHer option is available only when all digital channel states are DONTcare.

<N> An integer, 1 to the number of analog input channels.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query

```
:TRIGger:STATe:LOGic? {CHANnel<N> | DIGital<M>}
```

The query returns the logic state definition for the specified source.

Returned Format

```
[:TRIGger:STATe:LOGic {CHANnel<N> | DIGital<M>},]
   {LOW | HIGH | DONT | RIS | FALL | EITH}<NL>
```

See Also

- ":TRIGger:STATe:CLOCk" on page 1645

History Legacy command (existed before version 3.10).

:TRIGger:STATe:LTYPe

Command `:TRIGger:STATe[{1 | 2}]:LTYPe {AND | NAND}`

This command defines the state trigger logic type. If the logic type is set to AND, then a trigger is generated on the edge of the clock when the input waveforms match the pattern specified by the :TRIGger:STATe:LOGic command. If the logic type is set to NAND, then a trigger is generated on the edge of the clock when the input waveforms do not match the specified pattern.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:STATe:LTYPe?`

The query returns the currently specified state trigger logic type.

Returned Format `[:TRIGger:STATe:LTYPe] {AND | NAND}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:STATe:SLOPe

Command

```
:TRIGger:STATe[{1 | 2}]:SLOPe {RISing | FALLing | EITHer}
```

This command specifies the edge of the clock that is used to generate a trigger. The waveform source used for the clock is selected by using the :TRIGger:STATe:CLOCk command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

In mixed-signal (MSO) oscilloscopes, the EITHer option is available only when all digital channel states are DONTcare.

Query

```
:TRIGger:STATe:SLOPe?
```

The query returns the currently defined slope for the clock in State Trigger Mode.

Returned Format

```
[:TRIGger:STATe:SLOPe] {RIS | FALL | EITH}<NL>
```

See Also

- ":TRIGger:STATe:CLOCk" on page 1645

History Legacy command (existed before version 3.10).

:TRIGger:TIMeout (Timeout Trigger) Commands

:TRIGger:TIMeout:CONDition

Command `:TRIGger:TIMeout[{1 | 2}]:CONDition {HIGH | LOW | UNCHanged}`

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

This command sets the condition used for the timeout trigger.

HIGH Trigger when the waveform has been high for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

LOW Trigger when the waveform has been low for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

UNCHanged Trigger when the waveform has not changed state for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

Query `:TRIGger:TIMeout:CONDition?`

The query returns the currently defined trigger condition for the timeout trigger.

Returned Format `[:TRIGger:TIMeout:CONDition] {HIGH | LOW | UNCHanged}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:TIMeout:SOURce

Command
```
:TRIGger:TIMeout[{1 | 2}]:SOURce CHANnel<N>
```

This command specifies the channel source used to trigger the oscilloscope with the timeout trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query
```
:TRIGger:TIMeout:SOURce?
```

The query returns the currently defined channel source for the timeout trigger.

Returned Format
```
[:TRIGger:TIMeout:SOURce] CHANnel<N><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:TIMeout:TIME

Command

```
:TRIGger:TIMeout[{1 | 2}]:TIME <time>
```

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the timeout trigger, in seconds.

Query

```
:TRIGger:TIMeout:TIME?
```

The query returns the currently defined time for the trigger trigger.

Returned Format

```
[:TRIGger:TIMeout:TIME] <time><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:TRANsition (Transition Trigger) Commands

:TRIGger:TRANsition:MODE

Command

```
:TRIGger:TRANsition[{1 | 2}]:MODE <mode>
```

The :TRIGger:TRANsition:MODE command lets you look for:

- Transition violations that are greater than or less than the time specified by the :TRIGger:TRANsition:TIME command.
- Transition violations that are inside or outside of a time range specified by the :TRIGger:TRANsition:RANGe command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<mode> {GTHan | LTHan | RANGe | ORANGe}

Query

```
:TRIGger:TRANsition:MODE?
```

The :TRIGger:TRANsition:MODE? query returns the currently set transition violation trigger mode.

Returned Format

```
[:TRIGger:TRANsition:MODE] <mode><NL>

<mode> ::= {GTH | LTH | RANG | ORAN}
```

See Also

- ":TRIGger:TRANsition:TIME" on page 1657
- ":TRIGger:TRANsition:RANGe" on page 1655

History New in version 10.00.

:TRIGger:TRANsition:RANGe

Command

```
:TRIGger:TRANsition[{1 | 2}]:RANGe <gt_time>,<lt_time>
```

When the transition violation trigger mode is inside range (RANge) or outside range (ORANge), the :TRIGger:TRANsition:RANGe command specified the time range.

<gt_time> The minimum time bound of the range (greater than time) in NR3 format.

<lt_time> The maximum time bound of the range (less than time) in NR3 format.

Query

```
:TRIGger:TRANsition:RANGe?
```

The :TRIGger:TRANsition:RANGe? query returns the currently specified bounds of the time range.

Returned Format

```
[:TRIGger:TRANsition:RANGe]  <gt_time>,<lt_time><NL>
```

See Also

- ":TRIGger:TRANsition:MODE" on page 1654

History New in version 10.00.

:TRIGger:TRANsition:SOURce

Command `:TRIGger:TRANsition[{1 | 2}]:SOURce CHANnel<N>`

The transition source command lets you find any edge in your waveform that violates a rise time or fall time specification. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:TRANsition:SOURce?`

The query returns the currently defined transition source for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:SOURce] CHANnel<N><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:TRANsition:TIME

Command

```
:TRIGger:TRANsition[{1 | 2}]:TIME <time>
```

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the trigger violation transition, in seconds.

Query

```
:TRIGger:TRANsition:TIME?
```

The query returns the currently defined time for the trigger transition violation.

Returned Format

```
[:TRIGger:TRANsition:TIME] <time><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:TRANsition:TYPE

Command

```
:TRIGger:TRANsition[{1 | 2}]:TYPE {RISetime | FALLtime}
```

This command lets you select either a rise time or fall time transition violation trigger event.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:TRANsition:TYPE?
```

The query returns the currently defined transition type for the trigger transition violation.

Returned Format

```
[:TRIGger:TRANsition:TYPE] {RISetime | FALLtime}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:WINDow (Window Trigger) Commands

:TRIGger:WINDow:CONDition

Command

```
:TRIGger:WINDow[{1 | 2}]:CONDition {ENTer | EXIT
   | INSide [,{GTHan | LTHan}]
   | OUTSide [,{GTHan | LTHan}]}
```

This command describes the condition applied to the trigger window to actually generate a trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:WINDow:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:WINDow:CONDition] {ENTer | EXIT
   | INSide,{GTHan | LTHan}
   | OUTSide,{GTHan | LTHan}}<NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:WINDow:SOURce

Command `:TRIGger:WINDow[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the window trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:WINDow:SOURce?`

The query returns the currently defined channel source for the window trigger.

Returned Format `[:TRIGger:WINDow:SOURce] CHANnel<N><NL>`

History Legacy command (existed before version 3.10).

:TRIGger:WINDow:TIME

Command

```
:TRIGger:WINDow[{1 | 2}]:TIME <time>
```

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the trigger violation transition, in seconds.

Query

```
:TRIGger:WINDow:TIME?
```

The query returns the currently defined time for the trigger window timeout.

Returned Format

```
[:TRIGger:WINDow:TIME] <time><NL>
```

History Legacy command (existed before version 3.10).

:TRIGger:WINDow:TPOint

Command `:TRIGger:WINDow[{1 | 2}]:TPOint {BOUNdary | TIMeout}`

This command specifies whether the window trigger should occur at the boundary of the window or at a specified timeout period.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:WINDow:TPOint?`

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:TPOint] {BOUNdary | TIMeout}<NL>`

History Legacy command (existed before version 3.10).

45 :WAVeform Commands

KEYSIGHT TECHNOLOGIES

:WAVeform:YUNits? / 1729

The WAVeform subsystem is used to transfer waveform data between a computer and the oscilloscope. It contains commands to set up the waveform transfer and to send or receive waveform records to or from the oscilloscope.

Data Acquisition When data is acquired using the DIGitize command, the data is placed in the channel or function memory of the specified source. After the DIGitize command executes, the oscilloscope is stopped. If the oscilloscope is restarted by your program or from the front panel, the data acquired with the DIGitize command is overwritten.

You can query the preamble, elements of the preamble, or waveform data while the oscilloscope is running, but the data will reflect only the current acquisition, and subsequent queries will not reflect consistent data. For example, if the oscilloscope is running and you query the X origin, the data is queried in a separate command, it is likely that the first point in the data will have a different time than that of the X origin. This is due to data acquisitions that may have occurred between the queries. For this reason, Keysight Technologies does not recommend this mode of operation. Instead, you should use the DIGitize command to stop the oscilloscope so that all subsequent queries will be consistent.

NOTE Function and channel data are volatile and must be read following a DIGitize command or the data will be lost when the oscilloscope is turned off.

Waveform Data and Preamble The waveform record consists of two parts: the preamble and the waveform data. The waveform data is the actual sampled data acquired for the specified source. The preamble contains the information for interpreting the waveform data, including the number of points acquired, the format of the acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data.

The values in the preamble are set when you execute the DIGitize command. The preamble values are based on the current settings of the oscilloscope's controls.

Data Conversion Data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y origins and X and Y increments. These values can be read using the :WAVeform:XORigin?, WAVeform:YORigin?, WAVeform:XINCrement?, and WAVeform:YINCreament? queries.

Conversion from Data Values to Units To convert the waveform data values (essentially A/D counts) to real-world units, such as volts, use the following scaling formulas:

Y-axis Units = data value x Yincrement + Yorigin (analog channels) X-axis Units = data index x Xincrement + Xorigin, where the data index starts at zero: 0, 1, 2, ..., n-1.

The first data point for the time (X-axis units) must be zero, so the time of the first data point is the X origin.

Data Format for Data Transfer

There are four types of data formats that you can select using the :WAVeform:FORMat command: ASCii, BYTE, WORD, and BINary. Refer to the FORMat command in this chapter for more information on data formats.

:WAVeform:BANDpass?

Query `:WAVeform:BANDpass?`

The :WAVeform:BANDpass? query returns an estimate of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limits are computed as a function of the coupling and the selected filter mode. The cutoff frequencies are derived from the acquisition path and software filtering.

Returned Format `[:WAVeform:BANDpass]<lower_cutoff>,<upper_cutoff><NL>`

<lower_cutoff> Minimum frequency passed by the acquisition system.

<upper_cutoff> Maximum frequency passed by the acquisition system.

Example This example places the estimated maximum and minimum bandwidth limits of the source waveform in the string variable, strBandwidth, then prints the contents of the variable to the computer's screen.

```
Dim strBandwidth As String   ' Dimension variable.
myScope.WriteString ":WAVeform:BANDpass?"
strBandwidth = myScope.ReadString
Debug.Print strBandwidth
```

History Legacy command (existed before version 3.10).

:WAVeform:BYTeorder

Command

```
:WAVeform:BYTeorder {MSBFirst | LSBFirst}
```

The :WAVeform:BYTeorder command selects the order in which bytes are transferred from (or to) the oscilloscope using WORD and LONG formats. If MSBFirst is selected, the most significant byte is transferred first. Otherwise, the least significant byte is transferred first. The default setting is MSBFirst.

MSBFirst is for microprocessors, where the most significant byte resides at the lower address. LSBFirst is for microprocessors, where the least significant byte resides at the lower address.

NOTE **The data transfer rate is faster using the LSBFirst byte order because this is the oscilloscope's internal data alignment and waveform data points can simply be copied. The MSBFirst byte alignment (and formats other than WORD) require transformation from (or to) int16 during the transfer.**

Example This example sets up the oscilloscope to send the least significant byte first during data transmission.

```
myScope.WriteString ":WAVeform:BYTeorder LSBFirst"
```

Query

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder? query returns the current setting for the byte order.

Returned Format

```
[:WAVeform:BYTeorder] {MSBF | LSBF}<NL>
```

Example This example places the current setting for the byte order in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":WAVeform:BYTeorder?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

:WAVeform:CGRade:HEIGht?

Query `:WAVeform:CGRade:HEIGht?`

When the CGRade waveform view is selected (:WAVeform:VIEW CGRade), the :WAVeform:CGRade:HEIGht? query returns the color grade (pixel) database data height.

Returned Format `<height><NL>`

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

See Also

- "Getting Color Grade (Pixel) Database Count Values" on page 1716
- ":WAVeform:VIEW" on page 1715
- ":WAVeform:FORMat" on page 1698
- ":WAVeform:CGRade:WIDTh?" on page 1671

History New in version 6.00.

:WAVeform:CGRade:WIDTh?

Query `:WAVeform:CGRade:WIDTh?`

When the CGRade waveform view is selected (:WAVeform:VIEW CGRade), the :WAVeform:CGRade:WIDTh? query returns the color grade (pixel) database data width.

Returned Format `<width><NL>`

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

See Also

- "Getting Color Grade (Pixel) Database Count Values" on page 1716
- ":WAVeform:VIEW" on page 1715
- ":WAVeform:FORMat" on page 1698
- ":WAVeform:CGRade:HEIGht?" on page 1670

History New in version 6.00.

:WAVeform:COMPlete?

Query `:WAVeform:COMPlete?`

The :WAVeform:COMPlete? query returns the percent of time buckets that are complete for the currently selected waveform.

For the NORMal, RAW, and INTerpolate waveform types, the percent complete is the percent of the number of time buckets that have data in them, compared to the memory depth.

For the AVERage waveform type, the percent complete is the number of time buckets that have had the specified number of hits divided by the memory depth. The hits are specified by the :ACQuire:AVERage:COUNt command.

For the VERSus waveform type, percent complete is the least complete of the X-axis and Y-axis waveforms.

Returned Format `[:WAVeform:COMPlete] <criteria><NL>`

<criteria> 0 to 100 percent, rounded down to the closest integer.

Example This example places the current completion criteria in the string variable, strCriteria, then prints the contents of the variable to the computer's screen.

```
Dim strCriteria As String   ' Dimension variable.
myScope.WriteString ":WAVeform:COMPlete?"
strCriteria = myScope.ReadString
Debug.Print strCriteria
```

History Legacy command (existed before version 3.10).

:WAVeform:COUNt?

Query `:WAVeform:COUNt?`

The :WAVeform:COUNt? query returns the fewest number of hits in all of the time buckets for the currently selected waveform. For the AVERage waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value specified with the :ACQuire:AVERage:COUNt command.

For the NORMal, RAW, INTerpolate, and VERSus waveform types, the count value returned is one, unless the data contains holes (sample points where no data is acquired). If the data contains holes, zero is returned.

Returned Format `[:WAVeform:COUNt] <number><NL>`

<number> An integer. Values range from 0 to 1 for NORMal, RAW, or INTerpolate types, and VERSus type. If averaging is on values range from 0 to 65536.

Example This example places the current count field value in the string variable, strCount, then prints the contents of the variable to the computer's screen.

```
Dim strCount As String   ' Dimension variable.
myScope.WriteString ":WAVeform:COUNt?"
strCount = myScope.ReadString
Debug.Print strCount
```

History Legacy command (existed before version 3.10).

:WAVeform:COUPling?

Query `:WAVeform:COUPling?`

The :WAVeform:COUPling? query returns the input coupling of the currently selected source.

Returned Format `[:WAVeform:COUPling] DC<NL>`

This query always returns DC (and is provided for compatibility with other Infiniium oscilloscopes).

History Legacy command (existed before version 3.10).

:WAVeform:DATA

Command

```
:WAVeform:DATA <X_origin>,<X_increment>,<Y_origin>,<Y_increment>,
   <IEEE_block_data>
```

With Infiniium Offline only, the :WAVeform:DATA command copies the waveform points in the IEEE block data to the channel source specified by the :WAVeform:SOURce command.

After waveform data is uploaded for all channels being used, a "trigger" occurs and then analysis is performed. For more information, see Chapter 11, “Analyzing Multiple Acquisitions in Infiniium Offline,” starting on page 211.

The :WAVeform:DATA command is not allowed when the Infiniium software is running on an oscilloscope with active hardware.

<X_origin>, <X_increment>, <Y_origin>, <Y_increment> These values are 64-bit double-precision floating-point numbers.

<IEEE_block_data> This a definite-length block of BYTE (8-bit integer) or WORD (16-bit integer) format Q (quantization) data values, as described below. Both "Streaming On" and "Streaming Off" formats are supported.

The maximum and minimum Q values indicate clipped data.

You must specify the :WAVeform:FORMat and :WAVeform:BYTeorder appropriately before sending the :WAVeform:DATA. The WORD format with the LSBFirst byte order is the most efficient data alignment because it is the oscilloscope's internal data alignment and can simply be copied; all other formats (BYTE or WORD with MSBFirst) require waveform points to be transformed to int16 during the copy.

Example For an example of loading waveform files from multiple acquisitions into Infiniium Offline for analysis, see "Example: Loading Multiple Acquisitions Into Infiniium Offline" on page 218.

Query

```
:WAVeform:DATA? [<start>[,<size>]]
```

The :WAVeform:DATA? query outputs waveform data to the computer over the remote interface. The data is copied from a waveform memory, function, channel, bus, pod, or digital channel previously specified with the :WAVeform:SOURce command.

The preamble queries, such as :WAVeform:XINCrement, can be used to determine the vertical scaling, the horizontal scaling, and so on.

NOTE **When an acquisition is made on multiple channels, the data for each channel has the same X origin and the same number of points.**

<start> An integer value which is the starting point in the source memory which is the first waveform point to transfer.

<size> An integer value which is the number of points in the source memory to transfer. If the size specified is greater than the amount of available data then the size is adjusted to be the maximum available memory depth minus the <start> value.

Returned Format

```
[:WAVeform:DATA] <block_data>[,<block_data>]<NL>
```

NOTE The data's returned response depends upon the setting of the :WAVeform:STReaming command. See "Streaming Off" on page 1676 or "Streaming On" on page 1677 for more detail.

NOTE If the waveform data is ASCII formatted, no header information indicating the number of bytes being downloaded is included, and the waveform data is separated by commas.

When :ANALyze:SIGNal:TYPE is set to SPECtral, the format of the data returned from the :WAV:DATA? query is complex IQ pairs in the following order: <Real>,<Imaginary>,<Real>,<Imaginary>,.... The ASCii, BINary, BYTE, and WORD waveform formats are supported, although accuracy is lost with the BYTE format.

Streaming Off The returned waveform data response depends upon the setting of the :WAVeform:STReaming command. When the data format is BYTE and streaming is off, the number of waveform points must be less than 1,000,000,000 or an error occurs and only 999,999,999 bytes of data are sent. When the data format is WORD and streaming is off, the number of waveform points must be less than 500,000,000 or an error occurs and only 499,999,999 words of data are sent.

The returned waveform data in response to the :WAVeform:DATA? query is in the following order.

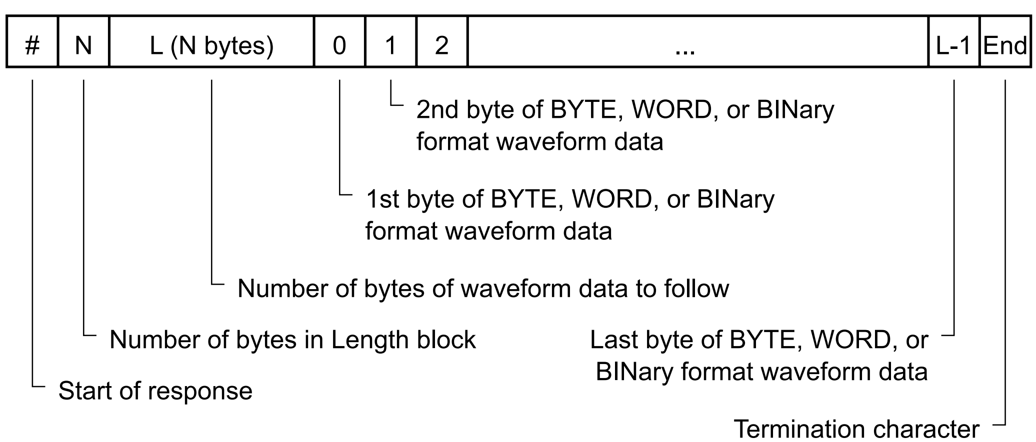


Figure 5 Streaming Off

Streaming On When streaming is on there is no limit on the number of waveform data points that are returned. It is recommended that any new programs use streaming on to send waveform data points. The waveform data response when streaming is on is as follows.

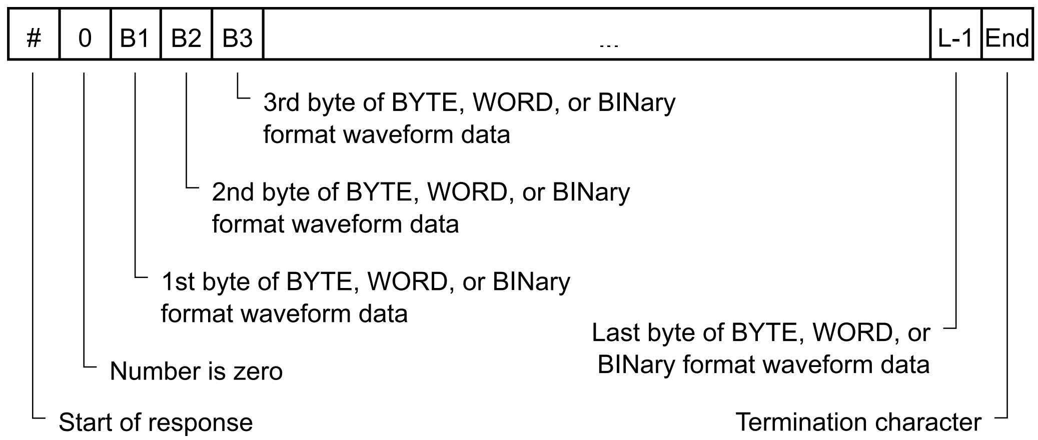


Figure 6 Streaming On

Example This example places the current waveform data from channel 1 into the varWavData array in the word format.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:SOURce CHANnel1   ' Select source.
myScope.WriteString ":WAVeform:FORMat WORD"   ' Select word format.
myScope.WriteString ":WAVeform:DATA?"
varWavData = myScope.ReadIEEEBlock(BinaryType_I2)
```

The format of the waveform data must match the format previously specified by the :WAVeform:FORMat and :WAVeform:BYTeorder commands.

DATA? Example for Analog Channels The following C example shows how to transfer WORD formatted waveform data for analog channels to a computer.

```
/* readdata. c */

/* Reading Word format example.  This program demonstrates the order
 * of commands suggested for operation of the Infiniium oscilloscope
 * via LAN.  This program initializes the oscilloscope, acquires data,
 * transfers data in WORD format, converts the data into time and
 * voltage values, and stores the data in a file as comma-separated
 * ascii values. This format is useful for spreadsheet and MATLAB
 * applications. It requires a waveform which is connected to Channel 1.
*/

#include <stdio.h>      /* location of: printf() */
#include <stdlib.h>     /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"
```

```
/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetVoltageConversionFactors( double *yInc, double *yOrg );
void GetTimeConversionFactors( double *xInc, double *xOrg );
void WriteCsvToFile( unsigned long ByteToRead );
void SetupDataTransfer( void );

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Globals */
INST bus;
INST scope;
char buffer[MAX_LENGTH];  /* Buffer for reading data */
double xOrg=0L, xInc=0L;  /* Values used to create time data */
double yOrg=0L, yInc=0L;  /* Values used to convert data to volts */


void main( void )
{
   unsigned long BytesToRead;

   if ( !InitIO() )
   {
      exit( 1 );
   }

   AcquireData();

   WriteIO( ":WAVeform:FORMat WORD" );    /* Setup transfer format */
   WriteIO( ":WAVeform:BYTeorder LSBFirst" ); /* Setup transfer of
                                                 LSB first */
   WriteIO( ":WAVeform:SOURce CHANnel1" ); /* Waveform data source
                                              channel 1 */
   WriteIO( ":WAVeform:STReaming 1" ); /* Turn on waveform
                                          streaming of data */

   GetVoltageConversionFactors( &yInc, &yOrg );
   GetTimeConversionFactors( &xInc, &xOrg );
   BytesToRead = MAX_LENGTH;
   SetupDataTransfer();
   WriteCsvToFile( BytesToRead );

   CloseIO( );

}
```

```
/*****************************************************************
 * Function name:  InitIO
 *   Parameters:   none
 *   Return value: TRUE if successful otherwise FALSE
 *   Description:  This routine initializes the SICL environment.
 *                 It sets up error handling, opens both an interface
 *                 and device session, sets timeout values, clears
 *                 the LAN interface card, and clears the
 *                 oscilloscope's LAN interface by performing a
 *                 Selected Device Clear.
 ******************************************************************/
int InitIO( void )
{

   ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

   bus = iopen( INTERFACE );   /* open interface session */
   if ( bus == 0 )
   {
      printf( "Bus session invalid\n" );
      return FALSE;
   }

   itimeout( bus, IO_TIMEOUT );  /* set bus timeout */
   iclear( bus );                /* clear the interface */

   scope = bus;                  /* open the scope device session */

   return TRUE;
}

/*******************************************************************
 * Function name:  WriteIO
 *   Parameters:   char *buffer which is a pointer to the character
 *                    string to be output
 *   Return value: none
 *   Description:  This routine outputs strings to the oscilloscope
 *                 device session using SICL commands.
 ******************************************************************/

void WriteIO( char *buffer )
{
   unsigned long actualcnt;
   unsigned long BytesToRead;
   int send_end = 1;

   BytesToRead = strlen( buffer );

   iwrite( scope, buffer, BytesToRead, send_end, &actualcnt );

}

/******************************************************************
 * Function name:  ReadByte
 *   Parameters:   char *buffer which is a pointer to the array to
 *                   store the read bytes
```

```
 *                  unsigned long BytesToRead which indicates the
 *                    maximum number of bytes to read
 *   Return value: integer which indicates the actual number of bytes
 *                  read
 *   Description: This routine inputs strings from the scope device
 *                 session using SICL commands.
*****************************************************************/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
   unsigned long BytesRead;
   int reason;

   BytesRead = BytesToRead;

   iread( scope, buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}

/*****************************************************************
 * Function name:  ReadWord
 *   Parameters:   short *buffer which is a pointer to the word array
 *                     to store the bytes read
 *                  int reason which is the reason that the read
 *                     terminated
 *                  unsigned long BytesToRead which indicates the
 *                     maximum number of bytes to read
 *   Return value: integer which indicates the actual number of
 *                     bytes read
 *   Description: This routine inputs an array of short values from
 *                 the oscilloscope device session using SICL commands.
******************************************************************/

unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead )
{
   long BytesRead;

   iread( scope, buffer, BytesToRead, reason, &BytesRead );

   return BytesRead;
}

/*****************************************************************
 * Function name:  ReadDouble
 *   Parameters:   double *buffer which is a pointer to the float
 *                     value to read
 *   Return value: none
 *   Description: This routine inputs a float value from the
 *                 oscilloscope device session using SICL commands.
******************************************************************/

void ReadDouble( double *buffer )
{
   iscanf( scope, "%lf", buffer );
}
```

```
/*****************************************************************
 * Function name: close_IO
 *   Parameters:   none
 *   Return value: none
 *   Description:  This routine closes device and interface sessions
 *                 for the SICL environment, and calls the routine
 *                 _siclcleanup which de-allocates resources
 *                 used by the SICL environment.
*****************************************************************/

void CloseIO( void )
{

   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

}

/*****************************************************************
 * Function name:  AcquireData
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine acquires data using the current
 *                oscilloscope settings.
*****************************************************************/

void AcquireData( void )
{
   /*
    * The root level :DIGitize command is recommended for
      * acquiring new waveform data. It initializes the
      * oscilloscope's data buffers, acquires new data,
      * and ensures that acquisition criteria are met before the
      * acquisition is stopped.  Note that the display is
      * automatically turned off when you use this form of the
      * :DIGitize command and must be turned on to view the
      * captured data on screen.
   */

   WriteIO(":DIGitize CHANnel1");
   WriteIO(":CHANnel1:DISPlay ON");

}

/*****************************************************************
 * Function name: GetVoltageConversionFactors
 *   Parameters:  double yInc which is the voltage difference
 *                   represented by adjacent waveform data digital codes
.
 *                double yOrg which is the voltage value of digital
 *                  code 0.
 *   Return value: none
 *   Description: This routine reads the conversion factors used to
 *                convert waveform data to volts.
```

```
*****************************************************************/

void GetVoltageConversionFactors( double *yInc, double *yOrg )
{

   /* Read values which are used to convert data to voltage values */

   WriteIO(":WAVeform:YINCrement?");
   ReadDouble( yInc );

   WriteIO(":WAVeform:YORigin?");
   ReadDouble( yOrg );

}

/*****************************************************************
 * Function name:  SetupDataTransfer
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine sets up the waveform data transfer and
 *                removes the # and 0 characters.
 *****************************************************************/

void SetupDataTransfer( void )
{
   char cData;

   WriteIO( ":WAVeform:DATA?" );   /* Request waveform data */

   /* Find the # character */

   do
   {
      ReadByte( &cData, 1L );
   } while ( cData != '#' );

   /* Find the 0 character */

   do
   {
      ReadByte( &cData, 1L );
   } while ( cData != '0' );
}

/*****************************************************************
 * Function name:  GetTimeConversionFactors
 *   Parameters:   double xInc which is the time between consecutive
 *                    sample points.
 *                 double xOrg which is the time value of the first
 *                    data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
 *****************************************************************/

void GetTimeConversionFactors( double *xInc, double *xOrg )
{
```

```
   /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( xOrg );

}

/*******************************************************************
 * Function name:  WriteCsvToFile
 *   Parameters:   unsigned long BytesToRead which is the number of
 *                   data points to read
 *   Return value: none
 *   Description: This routine stores the time and voltage
 *                information about the waveform as time and
 *                voltage separated by commas to a file.
********************************************************************/

void WriteCsvToFile( unsigned long BytesToRead )
{
   FILE *fp;
   int done = FALSE;
   int reason = 0;
   unsigned long i;
   unsigned long j = 0;
   unsigned long BytesRead = 0L;
   double Time;
   double Volts;
   short *buff;

   fp = fopen( "pairs.csv", "wb" ); /* Open file in binary mode - clear
                                       file if it already exists */

   if (fp != NULL)
   {
      while( !done )
      {
         BytesRead = ReadWord( buffer, &reason, BytesToRead );

         switch( reason )
         {
         case I_TERM_MAXCNT:
            done = FALSE;
            break;
         case I_TERM_END:
            done = TRUE;
            break;
         case I_TERM_CHR:
            done = TRUE;
            break;
         default:
            done = TRUE;
            break;
         };
```

```
         buff = (short *) buffer;

         for( i = 0; i < ((BytesRead - 1)/2); i++)
         {
            Time = (j * xInc) + xOrg; /* calculate time */
            j = j + 1;

            Volts = (buff[i] * yInc) + yOrg;/* calculate voltage */

            fprintf( fp, "%e,%f\n", Time, Volts );
         }
      }
      fclose( fp );

   }
   else
   {
      printf("Unable to open file 'pairs.csv'\n");
   }
}
```

DATA? Example for Digital Channels

The following C example shows how to transfer both BYTE and WORD formatted waveform data for digital channels to a computer. There is a file on the Infiniium Oscilloscope Example Programs disk called readdig.c in the c directory that contains this program.

```
/* readdig. c */

/* Reading Byte and Word format Example.  This program demonstrates the
order of
 commands suggested for operation of the Infiniium oscilloscope by LAN o
r GPIB.
 This program initializes the scope, acquires data, transfers data in bo
th the
 BYTE and WORD formats, converts the data into hex, octal, binary and ti
me values,
 and stores the data in a file as comma-separated values.  This format i
s useful
 for spreadsheet applications.
*/

#include <stdio.h>       /* location of: printf() */
#include <stdlib.h>      /* location of: atof(), atoi() */
#include <string.h>      /* location of: strlen() */
#include "sicl.h"

/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( short *buffer, unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetTimeConversionFactors( void );
```

```
void CreateTimeData( unsigned long AcquiredLength,
                     double *TimeValues );
void WriteCsvToFile( double *TimeValues, unsigned short *wordData,
                     unsigned char *byteData, unsigned long AcquiredLeng
th);
unsigned long SetupDataTransfer( double lTime, double rTime );
int Round( double number );

/* Defines */
#define MAX_LENGTH 8192000

#define LAN

#ifdef LAN
   #define INTERFACE "lan[130.29.71.202]:hpib7,7"
#else
   #define INTERFACE "hpib7"
#endif

#define DEVICE_ADDR "hpib7,7"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Globals */
INST bus;
INST scope;
double TimeValues[MAX_LENGTH];        /* Time value of data */
unsigned short wordData[MAX_LENGTH/2];/* Buffer for reading word format
data */
unsigned char byteData[MAX_LENGTH];   /* Buffer for reading byte format
data */
double xOrg, xInc;                   /* Values necessary to create time
data */

int Start;

void main( void )
{
   char Term;
   unsigned long BytesToRead;

   if ( !InitIO() ) {
      exit( 1 );
   }

   AcquireData();

   WriteIO( ":SYSTem:HEADer OFF" );
   WriteIO( ":SYSTem:LONGform OFF" );
   WriteIO( ":WAVeform:BYTEorder LSBFirst" ); /* Setup byte order */
   WriteIO( ":WAVeform:FORMat WORD" );        /* Setup transfer format *
/
   WriteIO( ":WAVeform:SOURce POD1" );      /* Waveform data source pod
1 */

   GetTimeConversionFactors( );
```

```
   BytesToRead = SetupDataTransfer( -25E-6, 25E-6);
   ReadWord( wordData, BytesToRead );
   ReadByte( &Term, 1L );               /* Read termination character *
/

   WriteIO(":WAVeform:FORMat BYTE");     /* Setup transfer format */

   BytesToRead = SetupDataTransfer( -25E-6, 25E-6);
   ReadByte( byteData, BytesToRead );
   ReadByte( &Term, 1L );               /* Read termination character *
/

   CreateTimeData( BytesToRead, TimeValues );

   WriteCsvToFile( TimeValues, wordData, byteData, BytesToRead );

   CloseIO( );

}

/***********************************************************************
********
*   Function name: InitIO
*   Parameters:    none
*   Return value:  none
*   Description:   This routine initializes the SICL environment.  It se
ts up
*                  error handling, opens both an interface and device se
ssion,
*                  sets timeout values, clears the GPIB interface card,
and
*                  clears the oscilloscope's GPIB card by performing a
*                  Selected Device Clear.
************************************************************************
*******/

int InitIO( void )
{

   ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

   bus = iopen( INTERFACE );   /* open interface session */
   if ( bus == 0 ) {
      printf( "Bus session invalid\n" );
      return FALSE;
   }

   itimeout( bus, IO_TIMEOUT );       /* set bus timeout */
   iclear( bus );                     /* clear the interface */

#ifdef LAN
   scope = bus;
#else
   scope = iopen( DEVICE_ADDR );      /* open the scope device session *
/
   if ( scope == 0) {
```

```
      printf( "Scope session invalid\n" );
      iclose( bus );     /* close interface session */
      _siclcleanup();    /* required for 16-bit applications */
      return FALSE;
   }

   itimeout( scope, IO_TIMEOUT );     /* set device timeout */
   iclear( scope );      /* perform Selected Device Clear on oscilloscop
e */
#endif

   return TRUE;
}

/************************************************************************
********
*   Function name: WriteIO
*   Parameters:    char *buffer which is a pointer to the character stri
ng to
*                  be output
*   Return value:  none
*   Description: This routine outputs strings to the oscilloscope device
*                session using SICL commands.
************************************************************************
*******/

void WriteIO( char *buffer )
{
   unsigned long actualcnt;
   unsigned long BytesToWrite;
   int send_end = 1;

   BytesToWrite = strlen( buffer );

   iwrite( scope, buffer, BytesToWrite, send_end, &actualcnt );

}


/************************************************************************
*******
*   Function name: ReadByte
*   Parameters:    char *buffer which is a pointer to the array to store
*                  the read bytes unsigned long BytesToRead which indica
tes
*                  the maximum number of bytes to read
*   Return value:  integer which indicates the actual number of bytes re
ad
*   Description: This routine inputs strings from the scope device sessi
on
*                using SICL commands.
************************************************************************
******/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
   unsigned long BytesRead=0L;
```

```
   int reason;

   BytesRead = BytesToRead;
    iread( scope, buffer, BytesToRead, &reason, &BytesRead );

    return BytesRead;
}

/*********************************************************************
*******
*   Function name: ReadWord
*   Parameters:    short *buffer which is a pointer to the word array to
 store
*                  the bytes read unsigned long BytesToRead which indica
tes
*                  the maximum number of bytes to read
*   Return value:  integer which indicates the actual number of bytes re
ad
*   Description: This routine inputs an array of short values from the
*                oscilloscope device session using SICL commands.
*********************************************************************
******/

unsigned long ReadWord( short *buffer, unsigned long BytesToRead )
{
   long BytesRead=0L;
   int reason;

   BytesRead = BytesToRead;
   iread( scope, (char *) buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}

/*********************************************************************
*******
*   Function name: ReadDouble
*   Parameters:    double *buffer which is a pointer to the float value
to read
*   Return value:  none
*   Description:   This routine inputs a float value from the oscillosco
pe
*                   device session using SICL commands.
*********************************************************************
*******/

void ReadDouble( double *buffer )
{
   int error;
    error = iscanf( scope, "%lf", buffer );
}

/*********************************************************************
*******
*  Function name: close_IO
*   Parameters:   none
*   Return value: none
```

```
*   Description:  This routine closes device and interface sessions for
*                 the SICL environment, and calls the routine _siclclean
up
*                 which de-allocates resources used by the SICL environm
ent.
************************************************************************
******/

void CloseIO( void )
{

   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

}

/***********************************************************************
*******
*   Function name:   AcquireData
*   Parameters:     none
*   Return value:   none
*   Description:   This routine acquires data using the current
*                  oscilloscope settings.
************************************************************************
******/

void AcquireData( void )
{
   /*
    *   The root level :DIGitize command is recommended for acquiring ne
w
    *   waveform data. It initializes the oscilloscope's data buffers,
    *   acquires new data, and ensures that acquisition criteria are met
    *   before the acquisition is stopped.  Note that the display is
    *   automatically turned off when you use this form of the
    *   :DIGitize command and must be turned on to view the captured dat
a
    *   on screen.
   */

   WriteIO(":DIGitize POD1");
   WriteIO(":POD1:DISPlay ON");

}

/***********************************************************************
*******
*   Function name: SetupDataTransfer
*   Parameters:    double lTime which is the time value of the first
*                  waveform memory location of data.
*                  double rTime which is the time value of the last
*                  waveform memory location of data.
*   Return value:  Number of bytes of waveform data to read.
*   Description: This routine sets up the waveform data transfer and get
s
```

```
*                 the number of bytes to be read.  The beginning of data
*                 starts with the # character followed by a number which
*                 tells how many bytes to read for the integer which is t
he
*                 total number of data bytes that are being transferred.
*                 Following this is the waveform data.  For example, if 1
024
*                 bytes of waveform data is being transferred then this
*                 information will be as follows:
*                              #41024 <1024 data bytes>
***********************************************************************
******/

unsigned long SetupDataTransfer( double lTime, double rTime )
{
   unsigned long BytesToRead;
   char header_str[8];
   char cData;
   unsigned long BytesRead;
   int Size;
   char Range[100];

   /* Find the index value of the first data memory location */

   Start = Round((lTime - xOrg)/xInc);
   if (Start < 1) {
      Start = 1;
   }

   /* Find the number of data bytes that you want */

   Size = Round((rTime - lTime)/xInc);

   sprintf( Range, ":WAVeform:DATA? %d,%d", Start, Size);
   WriteIO( Range );                /* Request waveform data */

   /* Find the # character */

   do {
         ReadByte( &cData, 1L );
   } while ( cData != '#' );

   /* Read the next byte which tells how many bytes to read for the numb
er
   * of waveform data bytes to transfer value.
   */

   ReadByte( &cData, 1L );
   BytesToRead = cData - '0';  /* Convert to a number */

   /* Reads the number of data bytes that will be transferred */

   BytesRead = ReadByte( header_str, BytesToRead );
   header_str[BytesRead] = '\0';
   BytesToRead = atoi( header_str );

   return BytesToRead;
```

```
}

/**********************************************************************
*******
*   Function name: GetTimeConversionFactors
*   Parameters:    none
*   Return value:  none
*   Description:   This routine transfers the waveform conversion
*                  factors for the time values.
**********************************************************************
*****/

Void GetTimeConversionFactors( void )
{

   /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( &xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( &xOrg );

}

/**********************************************************************
*******
*   Function name: CreateTimeData
*   Parameters:    unsigned long AcquiredLength which is the number of d
ata
*                  points
*                  double TimeValues is a pointer to the array where tim
e
*                  values are stored
*   Return value:  none
*   Description: This routine converts the data to time values using
*                the values that describe the waveform.  These values ar
e stored
*                in global variables.
**********************************************************************
*****/

void CreateTimeData( unsigned long AcquiredLength, double *TimeValues )
{
   unsigned long i;

   for (i = 0; i < AcquiredLength; i++) {
      TimeValues[i] =((Start + i) * xInc) + xOrg;   /* calculate time va
lues */
   }

}

/**********************************************************************
*******
*   Function name: WriteCsvToFile
```

```
*   Parameters:    double *TimeValues which is a pointer to an array of
*                  calculated time values
*                  unsigned short *wordData which is a pointer to an arr
ay of
*                  word format digital values
*                  unsigned char *byteData which is a pointer to an arra
y of
*                  byte format digital values
*                  unsigned long AcquiredLength which is the number of d
ata
*                  points read
*   Return value:  none
*   Description:   This routine stores the time and digital information
about
*                  the waveform as time, word format, and byte format
*                  separated by commas to a file.
*********************************************************************
******/

void WriteCsvToFile( double *TimeValues, unsigned short *wordData,
                unsigned char *byteData, unsigned long AcquiredLength )
{
  FILE *fp;
  char Binary[9];
  unsigned long i;
  int j;
  int k;

  fp = fopen( "digital.csv", "wb" ); /* Open file in binary mode - clear
 file
                                        if it already exists */

  if (fp != NULL) {

     fprintf( fp, "Time,Decimal Word Data,Hex Word Data,Hex Byte Data,Bi
nary Byte Data\n");
     Binary[8] = '\0';

     for ( i = 0; i < AcquiredLength; i++ ) {

        // Create the binary formatted byte data
        for ( j = 7, k = 0; j >= 0; j--, k++ ) {
               Binary[k] = ((byteData[i] & ( 1 << j )) >> j) + '0';
          }

        fprintf( fp, "%e,%d,%04X,%02X,%s\
n", TimeValues[i], wordData[i], wordData[i],
                                            byteData[i], Binary );
     }

       fclose( fp );
  }
  else {
       printf("Unable to open file 'digital.csv'\n");
  }

}
```

```
/***********************************************************************
*******
*   Function name: Round
*   Parameters:    double number which is a floating point number
*                  to be converted.
*   Return value:  The rounded integer value for the number parameter.
*   Description:   This routine takes a floating point number and create
s an
*                  integer.
************************************************************************
*****/

int Round( double number )
{
   if (number < 0.0f ) {
      return ( (int) (number - 0.5f) );
   }
   else {
      return ( (int) (number + 0.5f) );
   }
}
```

Understanding WORD and BYTE Formats

Before you can understand how the WORD and BYTE downloads work, it is necessary to understand how Infiniium creates waveform data.

Analog-to-digital Conversion Basics

The input channel of every digital sampling oscilloscope contains an analog-to-digital converter (ADC) as shown in Figure 7. The 8-bit ADC in some Infiniium oscilloscope models consists of 256 voltage comparators. Each comparator has two inputs. One input is connected to a reference dc voltage level and the other input is connected to the channel input. When the voltage of the waveform on the channel input is greater than the dc level, then the comparator output is a 1 otherwise the output is a 0. Each of the comparators has a different reference dc voltage. The output of the comparators is converted into an 8-bit integer by the encoder.

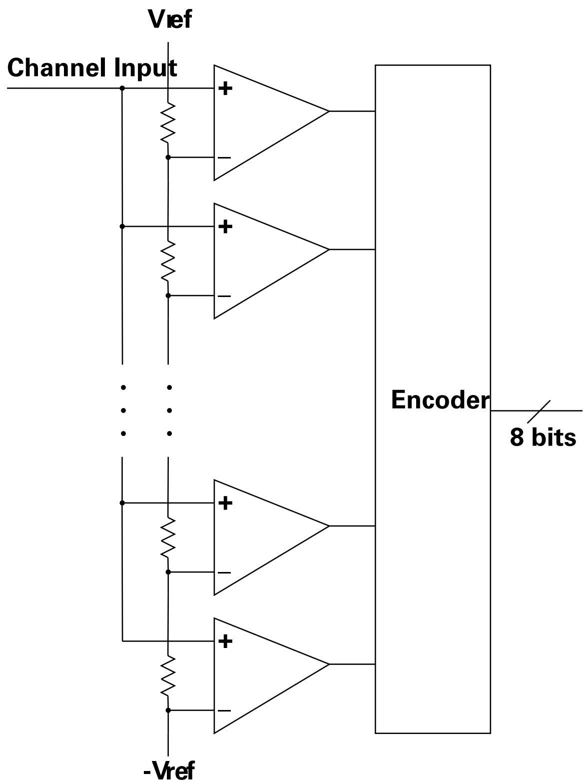


Figure 7 Block Diagram of an ADC

All ADCs have non-linearity errors which, if not corrected, can give less accurate vertical measurement results. For example, the non-linearity error for a 3-bit ADC is shown in the following figure.

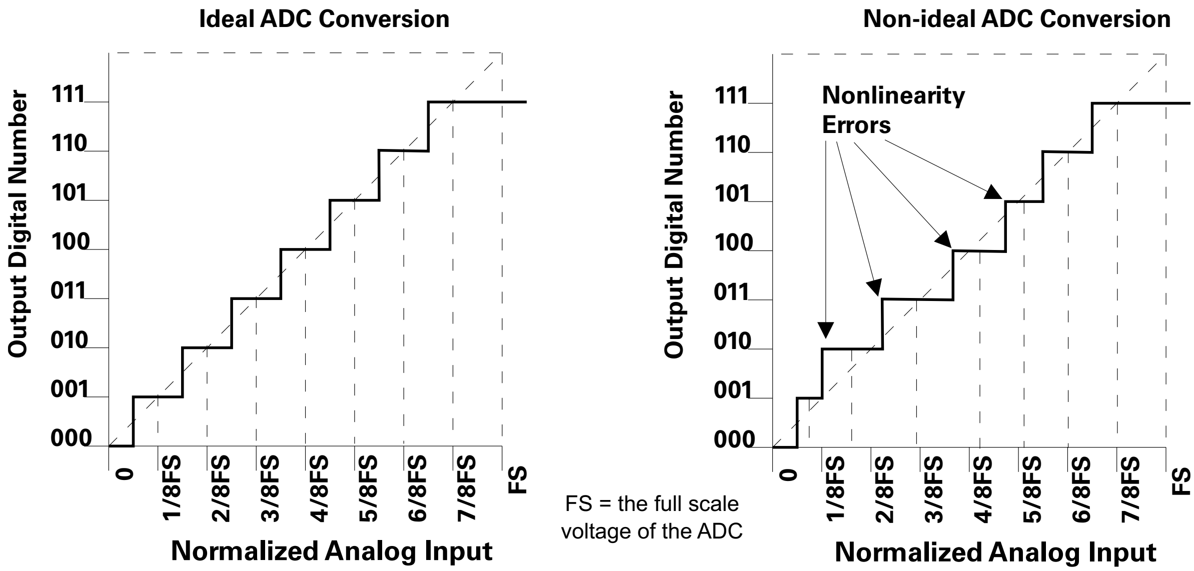


Figure 8 ADC Non-linearity Errors for a 3-bit ADC

The graph on the left shows an ADC which has no non-linearity errors. All of the voltage levels are evenly spaced producing output codes that represent evenly spaced voltages. In the graph on the right, the voltages are not evenly spaced with some being wider and some being narrower than the others.

When you calibrate your Infiniium, the input to each channel, in turn, is connected to the Aux Out connector. The Aux Out is connected to a 16-bit digital-to-analog converter (DAC) whose input is controlled by Infiniium's CPU. There are 65,536 dc voltage levels that are produced by the 16-bit DAC at the Aux Out. At each dc voltage value, the output of the ADC is checked to see if a new digital code is produced. When this happens, a 16-bit correction factor is calculated for that digital code and this correction factor is stored in a Calibration Look-up Table.

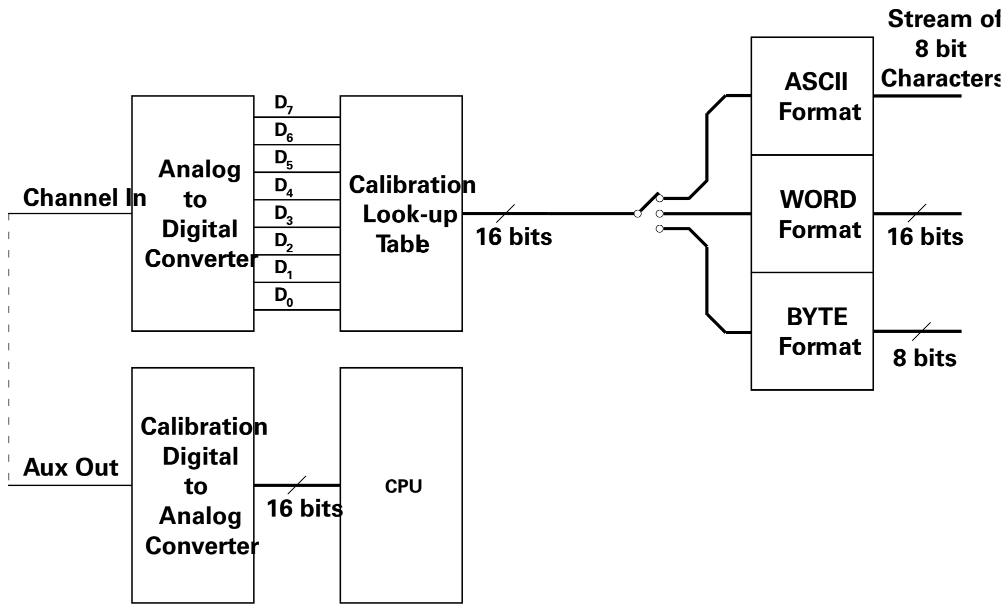


Figure 9 Data Flow in Infiniium

This process continues until all 256 digital codes are calibrated. The calibration process removes most of the non-linearity error of the ADC which yields more accurate vertical voltage values.

During normal operation of the oscilloscope, the output of the ADC is used as an address to the Calibration Look-up Table which produces 16-bit data for the oscilloscope to process and display. The output of the ADC is a signed 8-bit integer and the output of the Calibration Look-up Table is a signed 16-bit integer. If the amplitude of the input waveform is larger than the maximum dc reference level of the ADC, the ADC will output the maximum 8-bit value that it can (255). This condition is called ADC clipping. When the 255 digital code is applied to the Calibration Look-up Table, a 16-bit value, such as 32640 could be produced which represents an ADC clipped value.

Data values for clipped portions of waveforms are the maximum and minimum Q (quantization) values. For 16-bit waveform data, the maximum Q value is 32640 and the minimum Q value is -32704.

WORD and BYTE Data Formats

When downloading the waveform data in WORD format, the 16-bit signed integer value for each data point is sent in two consecutive 8-bit bytes over the remote interface. Whether the least significant byte (LSB) or the most significant byte (MSB) is sent first depends on the byte order determined by the BYTeorder command.

Before downloading the waveform data in BYTE format, each 16-bit signed integer is converted into an 8-bit signed integer. Because there are more possible 16-bit integers than there are 8-bit integers, a range of 16-bit integers is converted into single 8-bit numbers. For example, the following 16-bit numbers are all converted into one 8-bit number.

16-Bit Integers			8-Bit Integer	
Decimal	Hex		Hex	Decimal
26,240	0x6680	Truncated to >>	0x66	102
26,200	0x6658			
26,160	0x6630			
26,120	0x6608			

This conversion is what makes the BYTE download format less accurate than the WORD format.

See Also

- ":WAVeform:SOURce" on page 1712
- ":WAVeform:XINCrement?" on page 1719
- ":WAVeform:FORMat" on page 1698
- ":WAVeform:BYTeorder" on page 1669
- ":ANALyze:SIGNal:TYPE" on page 404
- ":WAVeform:STReaming" on page 1713

History

Legacy command (existed before version 3.10).

Version 10.12: When SPECtral is selected as the signal type, this query returns ASCII format data in complex IQ pairs in the following order: <Real>,<Imaginary>,<Real>,<Imaginary>,...

:WAVeform:FORMat

Command `:WAVeform:FORMat {ASCii | BINary | BYTE | WORD | FLOat}`

The :WAVeform:FORMat command sets the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope, and pertains to all waveforms.

The default format is ASCii.

Table 18 Selecting a Format

Type	Advantages	Disadvantages
ASCii	▪ Data is returned as voltage values and does not need to be converted. ▪ Is as accurate as WORD format. ▪ Supports HISTogram SOURce.	▪ Very slow data download rate.
BYTE	▪ Data download rate is twice as fast as the WORD format.	▪ Data is less accurate than the WORD format for analog channels. ▪ Not compatible with digital bus and pod data.
WORD	▪ Data is the most accurate for analog channels.	▪ Data download rate takes twice as long as the BYTE format.
BINary	▪ Supports HISTogram SOURce. ▪ Can be used for analog channels. ▪ Can be used for color grade waveform views.	▪ Data download rate takes twice as long as the BYTE format for analog channels.
FLOat	▪ Supports color grade waveform view only (:WAVeform:VIEW CGRade). ▪ Supports larger pixel count numbers than BINary with fewer data bytes.	▪ None.

ASCii ASCii-formatted data consists of waveform data values converted to the currently selected units, such as volts, and are output as a string of ASCII characters with each value separated from the next value by a comma. The values are formatted in floating point engineering notation. For example:

```
8.0836E+2,8.1090E+2,...,-3.1245E-3
```

NOTE **The ASCii format does not send out the header information indicating the number of bytes being downloaded.**

In ASCii format:

- The value "99.999E+36" represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

BYTE BYTE-formatted data is formatted as *signed* 8-bit integers. Depending on your programming language and IO library, you may need to create a function to convert these signed bytes to signed integers. In BYTE format:

- The value 125 represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

The waveform data values are converted from 16-bit integers to 8-bit integers before being downloaded to the computer. For more information, see "Understanding WORD and BYTE Formats" on page 1693.

WORD WORD-formatted data is transferred as *signed* 16-bit integers in two bytes. If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each word is sent first. In WORD format:

- The value 32672 represents a hole level. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

For more information, see "Understanding WORD and BYTE Formats" on page 1693.

BINary BINary-formatted data can be used with any SOURce. When a source is any valid source except for histogram, the data is returned in WORD format.

When the source is set to HISTogram, the data is transferred as signed 64-bit integers in 8 bytes. The are no hole values in the histogram data.

If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each long word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each long word is sent first.

FLOat When the color grade waveform view is selected (:WAVeform:VIEW CGRade), the color grade (pixel) database count values can be queried using these formats only:

- BINary — the :WAVeform:DATA? query will return a binary block of (8-byte) uint64 values.
- FLOat — the :WAVeform:DATA? query will return a binary block of (4-byte) single-precision floating-point values.

Example This example selects the WORD format for waveform data transmission.

```
myScope.WriteString ":WAVeform:FORMat WORD"
```

Query

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat? query returns the current output format for transferring waveform data.

Returned Format

```
[:WAVeform:FORMat] {ASCii | BINary | BYTE | WORD}<NL>
```

Example This example places the current output format for data transmission in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":WAVeform:FORMat?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also

- ":WAVeform:VIEW" on page 1715
- ":WAVeform:DATA" on page 1675
- ":WAVeform:CGRade:WIDTh?" on page 1671
- ":WAVeform:CGRade:HEIGht?" on page 1670

History Legacy command (existed before version 3.10).

Version 6.00: Added the FLOat option for getting the color grade (pixel) database data as single-precision floating-point values.

:WAVeform:PNOise:FREQuency

Query `:WAVeform:PNOise:FREQuency?`

The :WAVeform:PNOise:FREQuency? query returns the horizontal frequency axis values for the phase noise analysis results waveform. The corresponding vertical values for the waveform are returned by the :WAVeform:DATA? query when the :WAVeform:SOURce is set to PNOise.

With the phase noise analysis results waveform, the :WAVeform:FORMat must be set to ASCii or FLOat.

Returned Format

```
<freq_axis_values><NL>

<freq_axis_values> ::= {<comma-separated-ascii> (with ASCii format)
   | <definite-length_block_of_32-bit_floats> (with FLOat format)}
```

See Also

- ":WAVeform:DATA" on page 1675
- ":WAVeform:SOURce" on page 1712
- ":WAVeform:FORMat" on page 1698
- ":MEASure:PN:CORRelations" on page 1057
- ":MEASure:PN:DESKew" on page 1058
- ":MEASure:PN:EDGE" on page 1059
- ":MEASure:PN:HORizontal:STARt" on page 1060
- ":MEASure:PN:HORizontal:STOP" on page 1061
- ":MEASure:PN:RSSC" on page 1063
- ":MEASure:PN:SOURce" on page 1064
- ":MEASure:PN:SPURs" on page 1066
- ":MEASure:PN:SSENsitivity" on page 1067
- ":MEASure:PN:STATe" on page 1068
- ":MEASure:PN:VERTical:REFerence" on page 1069
- ":MEASure:PN:VERTical:SCALe" on page 1070
- ":MEASure:PN:WINDow" on page 1071

History New in version 10.25.

:WAVeform:POINts?

Query `:WAVeform:POINts?`

The :WAVeform:POINts? query returns the points value in the current waveform preamble. The points value is the number of time buckets contained in the waveform selected with the :WAVeform:SOURce command. If the Sin(x)/x interpolation filter is enabled, the number of points can be larger than the oscilloscope's memory depth setting because the waveform includes the interpolated points.

NOTE When an acquisition is made on multiple channels, the data for each channel has the same X origin and the same number of points.

With ":WAVeform:VIEW CGRade", the :WAVeform:POINts? query returns the number of count values in the color grade (pixel) database. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Returned Format `[:WAVeform:POINts] <points><NL>`

<points> An integer. See the :ACQuire:POINts command for a table of possible values.

Example This example places the current acquisition length in the numeric variable, varLength, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:POINts?"
varLength = myScope.ReadNumber
Debug.Print FormatNumber(varLength, 0)
```

NOTE

Turn Headers Off

When you are receiving numeric data into numeric variables, you should turn the headers off. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":ACQuire:POINts[:ANALog] — Memory depth" on page 310
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:PREamble?

Query

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble? query outputs a waveform preamble to the computer from the waveform source, which can be a waveform memory or channel buffer.

Returned Format

```
[:WAVeform:PREamble] <preamble_data><NL>
```

The preamble can be used to translate raw data into time and voltage values. The following lists the elements in the preamble.

<preamble_ data>

<format>, <type>, <points>, <count> , <X increment>, <X origin>, <X reference>, <Y increment>, <Y origin>, <Y reference>, <coupling>, <X display range>, <X display origin>, <Y display range>, <Y display origin>, <date>, <time>, <frame model #>, <acquisition mode>, <completion>, <X units>, <Y units>, <max bandwidth limit>, <min bandwidth limit>[,<segment count>]

Table 19 Waveform Preamble Elements

Element	Description
<format>	Returned format values can be: ▪ 0 – ASCii format ▪ 1 – BYTE format ▪ 2 – WORD format ▪ 3 – LONG format ▪ 4 – LONGLONG format ▪ 5 – FLOat format The format value describes the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope. (See ":WAVeform:FORMat" on page 1698.)
<type>	Returned type values can be: ▪ 1 – RAW ▪ 2 – AVERage ▪ 3 – VHIStogram ▪ 4 – HHIStogram ▪ 5 – not used ▪ 6 – INTerpolate ▪ 7 – not used ▪ 8 – not used ▪ 9 – DIGITAL ▪ 10 – PDETect This value describes how the waveform was acquired. (See also the ":WAVeform:TYPE?" on page 1714 query.)

Table 19 Waveform Preamble Elements (continued)

Element	Description
<points>	The number of data points or data pairs contained in the waveform data. (See ":ACQuire:POINts[:ANALog] — Memory depth" on page 310.)
<count>	For the AVERAGE waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value requested with the :ACQuire:AVERage:COUNt command. For RAW and INTerpolate waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble. (See ":WAVeform:TYPE?" on page 1714 and ":ACQuire[:AVERage]:COUNt" on page 295.)
<X increment>	The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired. (See the ":WAVeform:XINCrement?" on page 1719 query.)
<X origin>	The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double-precision 64-bit floating-point number. If the value is zero, then no data has been acquired. (See the ":WAVeform:XORigin?" on page 1720 query.)
<X reference>	The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the ":WAVeform:XREFerence?" on page 1722 query.)
<Y increment>	The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero, then no data has been acquired. (See the ":WAVeform:YINCrement?" on page 1725 query.)
<Y origin>	The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero, then no data has been acquired. (See the ":WAVeform:YORigin?" on page 1726 query.)
<Y reference>	The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero. (See the ":WAVeform:YREFerence?" on page 1728 query.)
<coupling>	Returned coupling values can be: ▪ 0 – AC coupling ▪ 1 – DC coupling ▪ 2 – DCFIFTY coupling ▪ 3 – LFREJECT coupling The input coupling of the waveform. The coupling value is ignored when sent to the oscilloscope in the preamble. (See the ":WAVeform:COUPling?" on page 1674 query.)

Table 19 Waveform Preamble Elements (continued)

Element	Description
<X display range>	The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero, then no data has been acquired. (See the ":WAVeform:XRANge?" on page 1721 query.)
<X display origin>	The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating-point number. If the value is zero, then no data has been acquired. (See the ":WAVeform:XDISplay?" on page 1718 query.)
<Y display range>	The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero, then no data has been acquired. (See the ":WAVeform:YRANge?" on page 1727 query.)
<Y display origin>	The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero, then no data has been acquired. (See the ":WAVeform:YDISplay?" on page 1724 query.)
<date>	A string containing the date in the format DD MMM YYYY, where DD is the day, 1 to 31; MMM is the month; and YYYY is the year.
<time>	A string containing the time in the format HH:MM:SS:TT, where HH is the hour, 0 to 23, MM is the minutes, 0 to 59, SS is the seconds, 0 to 59, and TT is the hundreds of seconds, 0 to 99.
<frame model #>	A string containing the model number and serial number of the oscilloscope in the format of MODEL#:SERIAL#. The frame model number is ignored when it is sent to an oscilloscope in the preamble.
<acquisition mode>	Returned acquisition mode values can be: ▪ 0 – RTIMe or HRESolution mode ▪ 1 – ETIMe mode ▪ 2 – SEGMented or SEGHres mode ▪ 3 – PDETect or SEGPdetect mode The acquisition sampling mode of the waveform. (See ":ACQuire:MODE" on page 308.)
<completion>	The completion value is the percent of time buckets that are complete. The completion value is ignored when it is sent to the oscilloscope in the preamble. (See the ":WAVeform:COMPlete?" on page 1672 query.)

Table 19 Waveform Preamble Elements (continued)

Element	Description
<X units> <Y units>	Returned type values can be: ▪ 0 – UNKNOWN units ▪ 1 – VOLT units ▪ 2 – SECOND units ▪ 3 – CONSTANT units ▪ 4 – AMP units ▪ 5 – DECIBEL units The X-axis and Y-axis units of the waveform. (See the ":WAVeform:XUNits?" on page 1723 query and the ":WAVeform:YUNits?" on page 1729 query.)
<max bandwidth limit > <min bandwidth limit >	The band pass consists of two values that are an estimation of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the ":WAVeform:BANDpass?" on page 1668 query.)
<segment count>	When segmented memory acquisitions are turned on, the ":WAVeform:SEGMented:ALL ON" command has been sent, and the waveform source is a channel or pod and has segmented acquisitions, this additional preamble value is returned. It specifies the number of segments in the returned waveform data. (See the ":WAVeform:SEGMented:COUNt?" on page 1708 query.)

With ":WAVeform:VIEW CGRade", the X increment, X origin, Y increment, and Y origin information returned by the :WAVeform:PREamble? query have different meanings for the color grade (pixel) database count values. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Example This example outputs the current waveform preamble for the selected source to the string variable, strPreamble.

```
Dim strPreamble As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:PREamble?"
strPreamble = myScope.ReadString
```

See Also

- ":WAVeform:DATA" on page 1675
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:SEGMented:ALL

Command `:WAVeform:SEGMented:ALL {{ON | 1} | {OFF | 0}}`

The :WAVeform:SEGmented:ALL command configures the DATA query for rapidly downloading all segments in one query.

The <start> and <size> optional parameters for the DATA query are still supported and represent the start and size of the data for each segment.

Powering on the oscilloscope or performing a Default Setup sets this command to OFF.

There is complete backwards compatibility when this command is set to OFF.

The ON setting applies when channel or pod sources have segmented memory acquisitions. For other sources, such as functions, the DATA query returns the data that corresponds to the current segment.

In segmented acquisition mode, with this command set to ON, the number of segments is appended to end of the waveform preamble.

Example This example turns on this command.

```
myScope.WriteString ":WAVeform:SEGMented:ALL ON"
```

Query `:WAVeform:SEGMented:ALL?`

This query returns the status of this command.

See Also

- ":WAVeform:SEGMented:POINts" on page 1709

History Legacy command (existed before version 3.10).

:WAVeform:SEGMented:COUNt?

Query `:WAVeform:SEGMented:COUNt?`

The :WAVeform:SEGMented:COUNt? query returns the index number of the last captured segment. A return value of zero indicates that the :ACQuire:MODE is not set to SEGMented.

The signal that is queried for the count is the signal set by the :WAVeform:SOURce command.

If you query the COUNt while the segmented acquisition is occurring, it will return the number of segments acquired so far.

<index_number> An integer number representing the index value of the last segment.

Returned Format [:WAVeform:SEGMented:COUNt] <index_number><NL>

Example This example returns the number of the last segment that was captured in the variable varIndex and prints it to the computer screen.

```
myScope.WriteString ":WAVeform:SEGMented:COUNt?"
varIndex = myScope.ReadNumber
Debug.Print FormatNumber(varIndex, 0)
```

See Also

- ":WAVeform:SOURce" on page 1712
- ":ACQuire:SEGMented:INDex" on page 319
- ":WAVeform:SEGMented:TTAG?" on page 1710
- ":WAVeform:SEGMented:ALL" on page 1707
- ":WAVeform:SEGMented:XLISt?" on page 1711

History Legacy command (existed before version 3.10).

Version 6.20: The signal that is queried for the count is the signal set by the :WAVeform:SOURce command. If you query the COUNt while the segmented acquisition is occurring, it will return the number of segments acquired so far.

:WAVeform:SEGMented:POINts

Query `:WAVeform:SEGMented:POINts?`

The :WAVeform:SEGMented:POINts? query returns the number of points in the segmented memory data.

If all segments are returned in one query (:WAVeform:SEGMented:POINts ON), the :WAVeform:SEGMented:POINts? query returns the number of points in all segments.

If individual segments are returned one at a time (:WAVeform:SEGMented:POINts OFF), the :WAVeform:SEGMented:POINts? query returns the number of points in the current segment.

Returned Format `<points><NL>`

`<points> ::= number of points in NR1 format`

See Also

- ":WAVeform:SEGMented:ALL" on page 1707

History New in version 11.10.

:WAVeform:SEGMented:TTAG?

Query `:WAVeform:SEGMented:TTAG?`

The :WAVeform:SEGMented:TTAG? query returns the time difference between the first segment's trigger point and the trigger point of the currently displayed segment.

The signal that is queried for the time tag is the signal set by the :WAVeform:SOURce command.

<delta_time> A real number in exponential format representing the time value difference between the first segment's trigger point and the currently displayed segment.

Returned Format [:WAVeform:SEGMented:TTAG] <delta_time><NL>

Example This example returns the time from the first segment's trigger point and the currently displayed segment's trigger point in the variable varDtime and prints it to the computer screen.

```
myScope.WriteString ":WAVeform:SEGMented:TTAG?"
varDtime = myScope.ReadNumber
Debug.Print FormatNumber(varDtime, 0)
```

See Also

- ":WAVeform:SOURce" on page 1712
- ":ACQuire:SEGMented:INDex" on page 319
- ":WAVeform:SEGMented:COUNt?" on page 1708
- ":WAVeform:SEGMented:ALL" on page 1707
- ":WAVeform:SEGMented:XLISt?" on page 1711

History Legacy command (existed before version 3.10).

Version 6.20: The signal that is queried for the time tag is the signal set by the :WAVeform:SOURce command.

:WAVeform:SEGMented:XLISt?

Query

```
:WAVeform:SEGMented:XLISt? {RELXorigin | ABSXorigin | TTAG | OFFSet}
```

The :WAVeform:SEGMented:XLISt? query rapidly downloads x-parameter values for all segments:

RELXorigin = relative X origin for each segment.

ABSXorigin = relative origin + time tag for each segment

TTAG = time tag for each segment

OFFSet = offset for each segment (within the returned data) when :WAVeform:SEGMented:ALL is ON. When :WAVeform:SEGMented:ALL is OFF, this query returns 0.

This query uses the DATA query format for the returned data and supports all waveform command options including: BYTeorder, FORmat (only ASCii or BINary (float64 with 8 bytes per value)), SOURce (only CHANnel<N> or POD<N>), STReaming, VIEW.

See Also

- ":WAVeform:SEGMented:ALL" on page 1707

History

Legacy command (existed before version 3.10).

Version 11.10: Added the OFFSet option for getting the offset for each segment (within the returned data) when :WAVeform:SEGMented:ALL is ON.

:WAVeform:SOURce

Command

```
:WAVeform:SOURce <source>
```

The :WAVeform:SOURce command selects a channel, function, waveform memory, or histogram as the waveform source.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | HISTogram | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR | BUS<B> | POD1 | POD2 | PODALL}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example selects channel 1 as the waveform source.

```
myScope.WriteString ":WAVeform:SOURce CHANnel1"
```

Query

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected waveform source.

Returned Format

```
[:WAVeform:SOURce] {CHAN<N> | DIFF<D> | COMM<C> | FUNC<F> | HIST
   |  WMEM<R> | CLOC | MTR | MSP | EQU<L> | XT<X> | PNO | INPut
   | CORRected | ERRor | LFPR | BUS<B> | POD1 | POD2 | PODALL}<NL>
```

Example This example places the current selection for the waveform source in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":WAVeform:SOURce?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

History Legacy command (existed before version 3.10).

:WAVeform:STReaming

Command `:WAVeform:STReaming {{ON | 1} | {OFF | 0}}`

When enabled, :WAVeform:STReaming allows more than 999,999,999 bytes of data to be transferred from the Infiniium oscilloscope to a PC when using the :WAVeform:DATA? query. See the :WAVeform:DATA? query for information on receiving this much data.

Example This example turns on the streaming feature.

```
myScope.WriteString ":WAVeform:STReaming ON"
```

Query `:WAVeform:STReaming?`

The :WAVeform:STReaming? query returns the status of the streaming feature.

Returned Format `[:WAVeform:STReaming] {1 | 0}<NL>`

History Legacy command (existed before version 3.10).

:WAVeform:TYPE?

Query `:WAVeform:TYPE?`

The :WAVeform:TYPE? query returns the current acquisition data type for the currently selected source. The type returned describes how the waveform was acquired. The waveform type may be:

- RAW — data consists of one data point in each time bucket with no interpolation.
- AVERage — data consists of the average of the first n hits in a time bucket, where n is the value in the count portion of the preamble. Time buckets that have fewer than n hits return the average of the data they contain. If the :ACQuire:COMPlete parameter is set to 100%, then each time bucket must contain the number of data hits specified with the :ACQuire:AVERage:COUNt command.
- VHIStogram — data is a vertical histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.
- HHIStogram — data is a horizontal histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.
- INTerpolate — In the INTerpolate acquisition type, the last data point in each time bucket is stored, and additional data points between the acquired data points are filled by interpolation.
- DIGITAL — data consists of digital pod or bus values for each time bucket.
- PDETect — data consists of two data points in each time bucket: the minimum values and the maximum values.

Returned Format `[:WAVeform:TYPE] {RAW | AVER | VHIS | HHIS | INT | DIGITAL | PDET}<NL>`

Example This example places the current acquisition data type in the string variable, strType, then prints the contents of the variable to the computer's screen.

```
Dim strType As String   ' Dimension variable.
myScope.WriteString ":WAVeform:TYPE?"
strType = myScope.ReadString
Debug.Print strType
```

History Legacy command (existed before version 3.10).

:WAVeform:VIEW

Command

```
:WAVeform:VIEW {ALL | MAIN | WINDow | CGRade}
```

The :WAVeform:VIEW command selects the view of the waveform that is selected for data and preamble queries. You can set the command to ALL, MAIN, WINDow, or CGRade.

The view has different meanings depending upon the waveform source selected.

The default setting for this command is ALL.

The following table summarizes the parameters for this command for each source.

Table 20 Waveform View Parameters

Source/Parameter	ALL	MAIN	WINDow	CGRade
CHANnel<N>	All data	Main time base	Zoom	Color grade (pixel) database
WMEMory<R>	All data	Memory time base	Memory time base	Color grade (pixel) database
FUNCtion<F>	All data	All data	All data	Color grade (pixel) database

Channels For channels, you may select ALL, MAIN, or WINDow views. If you select ALL, all of the data in the waveform record is referenced. If you select MAIN, only the data in the main time base range is referenced. The first value corresponds to the first time bucket in the main time base range, and the last value corresponds to the last time bucket in the main time base range. If WINDow is selected, only data in the delayed view is referenced. The first value corresponds to the first time bucket in the delayed view and the last value corresponds to the last time bucket in the delayed view.

Memories For memories, if you specify ALL, all the data in the waveform record is referenced. WINDow and MAIN refer to the data contained in the memory time base range for the particular memory. The first value corresponds to the first time bucket in the memory time base range, and the last value corresponds to the last time bucket in the memory time base range.

Functions For functions, ALL, MAIN, and WINDow refer to all of the data in the waveform record.

Example This example sets up the oscilloscope to view all of the data.

```
myScope.WriteString ":WAVeform:VIEW ALL"
```

Getting Color Grade (Pixel) Database Count Values

Before you can select the CGRade waveform view, you must enable color grade persistence, color grade view, or a real-time eye for the source waveform.

NOTE

Getting color grade (pixel) database count values is not supported when segmented memory acquisitions are enabled.

After you select the CGRade waveform view, color grade (pixel) database information is available from the following queries:

Command	Description
:WAVeform:CGRade:HEIGht? (see page 1670)	Returns the color grade (pixel) database data height.
:WAVeform:CGRade:WIDTh? (see page 1671)	Returns the color grade (pixel) database data width.
:WAVeform:POINts? (see page 1702)	The number of count values in the database (should be the width times the height).
:WAVeform:XINCrement? (see page 1719)	The time per column of the color grade database.
:WAVeform:XORigin? (see page 1720)	The time at column 0 of the color grade database.
:WAVeform:YINCrement? (see page 1725)	The volts per row of the color grade database.
:WAVeform:YORigin? (see page 1726)	The volts at row 0 of the color grade database.
:WAVeform:PREamble? (see page 1703)	Returns the same points, X increment, X origin, Y increment, and Y origin information as the individual queries.

To get the color grade (pixel) database count values:

1 Use the :WAVeform:FORMat command to specify the format you in which want the database count values returned:

   - Use the ":WAVeform:FORMat BINary" command to get (8-byte) uint64 values.
   - Use the ":WAVeform:FORMat FLOat" command to get (4-byte) single-precision floating-point values.

   When getting color grade database values, the only valid formats are BINary and FLOat.

2 Send the :WAVeform:DATA? query.

   A binary block of values in the selected format is returned.

The order of the returned values is:

- From the row at the top of the display to the bottom of the display (with "height" number of rows).
    - Within a row, values are returned from the left of the display to the right of the display (with "width" number of columns).

Query `:WAVeform:VIEW?`

The :WAVeform:VIEW? query returns the currently selected view.

Returned Format `[:WAVeform:VIEW] {ALL | MAIN | WIND | CGR}<NL>`

Example This example returns the current view setting to the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":WAVeform:VIEW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":WAVeform:FORMat" on page 1698
- ":WAVeform:DATA" on page 1675
- ":WAVeform:CGRade:WIDTh?" on page 1671
- ":WAVeform:CGRade:HEIGht?" on page 1670
- ":WAVeform:POINts?" on page 1702
- ":WAVeform:XINCrement?" on page 1719
- ":WAVeform:XORigin?" on page 1720
- ":WAVeform:YINCrement?" on page 1725
- ":WAVeform:YORigin?" on page 1726
- ":WAVeform:PREamble?" on page 1703

History Legacy command (existed before version 3.10).

Version 6.00: Added the CGRade option for getting the color grade (pixel) database data.

:WAVeform:XDISplay?

Query `:WAVeform:XDISplay?`

The :WAVeform:XDISplay? query returns the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. For VERSus type waveforms, it is the value at the center of the X-axis of the display. This value is treated as a double precision 64-bit floating point number.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XDISplay] <value><NL>`

<value> A real number representing the X-axis value at the left edge of the display.

Example This example returns the X-axis value at the left edge of the display to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:XDISplay"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:WAVeform:XINCrement?

Query `:WAVeform:XINCrement?`

The :WAVeform:XINCrement? query returns the duration between consecutive data points for the currently specified waveform source. For time domain waveforms, this is the time difference between consecutive data points. For VERSus type waveforms, this is the duration between levels on the X axis. For voltage waveforms, this is the voltage corresponding to one level.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

With ":WAVeform:VIEW CGRade", the :WAVeform:XINCrement? query returns the time per column of the color grade (pixel) database. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Returned Format `[:WAVeform:XINCrement] <value><NL>`

<value> A real number representing the duration between data points on the X axis.

Example This example places the current X-increment value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:XINCrement?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- You can also obtain the X-increment value through the :WAVeform:PREamble? query. See ":WAVeform:PREamble?" on page 1703.
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:XORigin?

Query `:WAVeform:XORigin?`

The :WAVeform:XORigin? query returns the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. For VERSus type waveforms, it is the X-axis value at level zero. For voltage waveforms, it is the voltage at level zero. The value returned by this query is treated as a double precision 64-bit floating point number.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

NOTE When an acquisition is made on multiple channels, the data for each channel has the same X origin and the same number of points.

With ":WAVeform:VIEW CGRade", the :WAVeform:XORigin? query returns the time at column 0 of the color grade (pixel) database. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Returned Format `[:WAVeform:XORigin] <value><NL>`

<value> A real number representing the X-axis value of the first data point in the data record.

Example This example places the current X-origin value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:XORigin?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- You can also obtain the X-origin value through the :WAVeform:PREamble? query. See ":WAVeform:PREamble?" on page 1703.
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:XRANge?

Query `:WAVeform:XRANge?`

The :WAVeform:XRANge? query returns the X-axis duration of the displayed waveform. For time domain waveforms, it is the duration of the time across the display. For VERSus type waveforms, it is the duration of the waveform that is displayed on the X axis.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XRANge] <value><NL>`

<value> A real number representing the X-axis duration of the displayed waveform.

Example This example returns the X-axis duration of the displayed waveform to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:XRANge?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:WAVeform:XREFerence?

Query `:WAVeform:XREFerence?`

The :WAVeform:XREFerence? query returns the data point or level associated with the X-origin data value. It is at this data point or level that the X origin is defined. In this oscilloscope, the value is always zero.

Returned Format `[:WAVeform:XREFerence] 0<NL>`

Example This example places the current X-reference value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:XREFerence?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also You can obtain the X-reference value through the :WAVeform:PREamble? query.

History Legacy command (existed before version 3.10).

:WAVeform:XUNits?

Query `:WAVeform:XUNits?`

The :WAVeform:XUNits? query returns the X-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format `[:WAVeform:XUNits] {UNKNown | VOLT | SECond | CONStant | AMP | DECibels | HERTz | WATT}<NL>`

Example This example returns the X-axis units of the currently selected waveform source to the string variable, strUnit, then prints the contents of the variable to the computer's screen.

```
Dim strUnit As String   ' Dimension variable.
myScope.WriteString ":WAVeform:XUNits?"
strUnit = myScope.ReadString
Debug.Print strUnit
```

History Legacy command (existed before version 3.10).

:WAVeform:YDISplay?

Query `:WAVeform:YDISplay?`

The :WAVeform:YDISplay? query returns the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display.

NOTE **A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.**

Returned Format `[:WAVeform:YDISplay] <value><NL>`

<value> A real number representing the Y-axis value at the center of the display.

Example This example returns the current Y-display value to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":":WAVeform:YDISplay?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:WAVeform:YINCrement?

Query `:WAVeform:YINCrement?`

The :WAVeform:YINCrement? query returns the y-increment voltage value for the currently specified source. This voltage value is the voltage difference between two adjacent waveform data digital codes. Adjacent digital codes are codes that differ by one least significant bit. For example, the digital codes 24680 and 24681 vary by one least significant bit.

- For BYTE and WORD data, and voltage waveforms, it is the voltage corresponding to one least significant bit change.
- For ASCii data format, the YINCrement is the full scale voltage range covered by the A/D converter.

NOTE **A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.**

With ":WAVeform:VIEW CGRade", the :WAVeform:YINCrement? query returns the volts per row of the color grade (pixel) database. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Returned Format `[:WAVeform:YINCrement] <real_value><NL>`

<real_value> A real number in exponential format.

Example This example places the current Y-increment value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:YINCrement?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1693.
- You can also obtain the Y-increment value through the :WAVeform:PREamble? query. See ":WAVeform:PREamble?" on page 1703.
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:YORigin?

Query

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin voltage value for the currently specified source. The voltage value returned is the voltage value represented by the waveform data digital code 00000.

- For BYTE and WORD data, and voltage waveforms, it is the voltage at digital code zero.
- For ASCii data format, the YORigin is the Y-axis value at the center of the data range. Data range is returned in the Y increment.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

With ":WAVeform:VIEW CGRade", the :WAVeform:YORigin? query returns the volts at row 0 of the color grade (pixel) database. See "Getting Color Grade (Pixel) Database Count Values" on page 1716.

Returned Format

```
[:WAVeform:YORigin] <real_value><NL>
```

<real_value> A real number in exponential format.

Example This example places the current Y-origin value in the numeric variable, varCenter, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:YORigin?"
varCenter = myScope.ReadNumber
Debug.Print FormatNumber(varCenter, 0)
```

See Also

- For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1693.
- You can also obtain the Y-origin value through the :WAVeform:PREamble? query. See ":WAVeform:PREamble?" on page 1703.
- ":WAVeform:VIEW" on page 1715

History Legacy command (existed before version 3.10).

:WAVeform:YRANge?

Query `:WAVeform:YRANge?`

The :WAVeform:YRANge? query returns the Y-axis duration of the displayed waveform. For voltage waveforms, it is the voltage across the entire display.

NOTE **A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.**

Returned Format `[:WAVeform:YRANge] <value><NL>`

<value> A real number representing the Y-axis duration of the displayed waveform.

Example This example returns the current Y-range value to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:YRANge?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:WAVeform:YREFerence?

Query `:WAVeform:YREFerence?`

The :WAVeform:YREFerence? query returns the y-reference voltage value for the currently specified source. It is at this level that the Y origin is defined. In this oscilloscope, the value is always zero.

Returned Format `[:WAVeform:YREFerence] 0<NL>`

Example This example places the current Y-reference value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":WAVeform:YREFerence?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1693.

You can obtain the Y-reference value through the :WAVeform:PREamble? query.

History Legacy command (existed before version 3.10).

:WAVeform:YUNits?

Query `:WAVeform:YUNits?`

The :WAVeform:YUNits? query returns the Y-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format `[:WAVeform:YUNits] {UNKNown | VOLT | SECond | HITS | DECibels | CONStant | AMP}<NL>`

Example This example returns the Y-axis units of the currently selected waveform source to the string variable, strUnit, then prints the contents of the variable to the computer's screen.

```
Dim strUnit As String   ' Dimension variable.
myScope.WriteString ":WAVeform:YUNits?"
strUnit = myScope.ReadString
Debug.Print strUnit
```

History Legacy command (existed before version 3.10).

46 :WGEN (Waveform Generator) Commands

KEYSIGHT TECHNOLOGIES

You can use the built-in waveform generator to output sine, square, ramp, pulse, DC, noise, sine cardinal, exponential rise, exponential fall, cardiac, and gaussian pulse waveforms. The :WGEN commands are used to select the waveform function and parameters.

:WGEN:ARBitrary:IMPort

Command

```
:WGEN:ARBitrary:IMPort <file_name>

<file_name> ::= quoted ASCII full path name
```

The :WGEN:ARBitrary:IMPort command imports arbitrary waveform points from a previously-saved comma-separated value (CSV) arbitrary waveform points file.

See Also

- ":WGEN:ARBitrary:INTerpolate" on page 1734
- ":WGEN:FUNCtion" on page 1736

History New in version 11.05.

:WGEN:ARBitrary:INTerpolate

Command

```
:WGEN:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}}
```

The :WGEN:ARBitrary:INTerpolate command specifies how lines are drawn between arbitrary waveform points:

- ON — Lines are drawn between points in the waveform editor. Voltage levels change linearly between one point and the next.
- OFF — All line segments in the waveform editor are horizontal. The voltage level of one point remains until the next point.

Query

```
:WGEN:ARBitrary:INTerpolate?
```

The :WGEN:ARBitrary:INTerpolate? query returns the arbitrary waveform interpolate setting.

Returned Format

```
<setting><NL>

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

See Also

- ":WGEN:ARBitrary:IMPort" on page 1733
- ":WGEN:FUNCtion" on page 1736

History New in version 11.05.

:WGEN:FREQuency

Command

```
:WGEN:FREQuency <frequency>
```

```
<frequency> ::= frequency in Hz in NR3 format
```

For all waveforms except Noise and DC, the :WGEN:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN:PERiod command.

Query

```
:WGEN:FREQuency?
```

The :WGEN:FREQuency? query returns the currently set waveform generator frequency.

Returned Format

```
<frequency><NL>
```

```
<frequency> ::= frequency in Hz in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:PERiod" on page 1760

History New in version 11.00.

:WGEN:FUNCtion

Command

```
:WGEN:FUNCtion <signal>

<signal> ::= {SINusoid | SQUare | RAMP | PULSe | DC | NOISe | SINC
   | EXPRise | EXPFall | CARDiac | GAUSsian | PRBS | DEMO | ARBitrary}
```

The :WGEN:FUNCtion command selects the type of waveform:

Waveform Type	Characteristics	Frequency Range	Max Amplitude[2] (High-Z) [1]	Pct of Wfm Above/Below Offset[2]
ARBitrary	Use these commands to set the arbitrary signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	12.5 mHz to 25 MHz	2 mVpp to 10 Vpp	50/50
SINusoid	Use these commands to set the sine signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	12.5 mHz to 50 MHz	2 mVpp to 10 Vpp	50/50
SQUare	Use these commands to set the square wave signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765 ▪ ":WGEN:FUNCtion:SQUare:DCYCle" on page 1742 The duty cycle can be adjusted from 20% to 80%.	1.5 mHz to 20 MHz	2 mVpp to 10 Vpp	50/50

Waveform Type	Characteristics	Frequency Range	Max Amplitude[2] (High-Z) [1]	Pct of Wfm Above/Below Offset[2]
RAMP	Use these commands to set the ramp signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765 ▪ ":WGEN:FUNCtion:RAMP:SYMMetry" on page 1741 Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.	12.5 mHz to 200 kHz	2 mVpp to 10 Vpp	50/50
PULSe	Use these commands to set the pulse signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765 ▪ ":WGEN:FUNCtion:PULSe:WIDTh" on page 1740 The pulse width can be adjusted from 10 ns to the period minus 10 ns.	1.5 mHz to 20 MHz	2 mVpp to 10 Vpp	50/50
DC	Use this command to set the DC level: ▪ ":WGEN:VOLTage:OFFSet" on page 1766	n/a	n/a	50/50
NOISe	Use these commands to set the noise signal parameters: ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	n/a	2 mVpp to 10 Vpp	50/50

Waveform Type	Characteristics	Frequency Range	Max Amplitude[2] (High-Z) [1]	Pct of Wfm Above/Below Offset[2]
SINC	Use these commands to set the sine cardinal signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766	12.5 mHz to 1 MHz	2 mVpp to 9 Vpp	80/20
EXPRise	Use these commands to set the exponential rise signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	12.5 mHz to 10 MHz	2 mVpp to 10 Vpp	50/50
EXPFall	Use these commands to set the exponential fall signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	12.5 mHz to 10 MHz	2 mVpp to 10 Vpp	50/50
CARDiac	Use these commands to set the cardiac signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766	12.5 mHz to 200 kHz	2 mVpp to 10 Vpp	80/20
GAUSsian	Use these commands to set the gaussian pulse signal parameters: ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766	50 nHz to 5 MHz	2 mVpp to 8 Vpp	100/0

Waveform Type	Characteristics	Frequency Range	Max Amplitude[2] (High-Z) [1]	Pct of Wfm Above/Below Offset[2]
PRBS	Use these commands to set the sine signal parameters: ▪ ":WGEN:PRBS:LENGth" on page 1761 ▪ ":WGEN:FREQuency" on page 1735 ▪ ":WGEN:PERiod" on page 1760 ▪ ":WGEN:VOLTage" on page 1763 ▪ ":WGEN:VOLTage:OFFSet" on page 1766 ▪ ":WGEN:VOLTage:HIGH" on page 1764 ▪ ":WGEN:VOLTage:LOW" on page 1765	100 Hz to 40 MHz (actually Bps)	2 mVpp to 10 Vpp	50/50
DEMO	Demo signals are intended to be used with the front panel graphical user interface only. There are no SCPI commands for selecting or returning the specific type of demo signal. For a description of the demo signals available, see the online help.			
[1]When the output load is 50 Ω, these values are halved. [2]The offset plus the percent of the waveform amplitude above or below the offset must be within the ±8.00 V Max/Min V range. For example, for a sine waveform where 50% of the waveform is above and below the offset, when the amplitude is 10 Vpp, the offset must be between ±3 V; when the amplitude is 10 mVpp, the offset must be between ±7.995 V. For amplitudes less than 10 mV, the Max/Min V range becomes ±395 mV.				

Query `:WGEN:FUNCtion?`

The :WGEN:FUNCtion? query returns the currently selected signal type.

The DEMO signal type is returned when the **Demo** type has been selected in the front panel graphical user interface. Demo signals are intended to be used with the front panel graphical user interface only. There are no SCPI commands for selecting or returning the specific type of demo signal.

Returned Format

```
<signal><NL>

<signal> ::= {SIN | SQU | RAMP | PULS | DC | NOIS | SINC | EXPR | EXPF
            | CARD | GAUS | PRBS | DEMO | ARB}
```

See Also

- ":WGEN:MODulation:NOISe" on page 1751

History New in version 11.00.

Version 11.05: ARBitrary has been added as a waveform generator function.

:WGEN:FUNCtion:PULSe:WIDTh

Command

```
:WGEN:FUNCtion:PULSe:WIDTh <width>
```

```
<width> ::= pulse width in seconds in NR3 format
```

For Pulse waveforms, the :WGEN: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

```
:WGEN:FUNCtion:PULSe:WIDTh?
```

The :WGEN:FUNCtion:PULSe:WIDTh? query returns the currently set pulse width.

Returned Format

```
<width><NL>
```

```
<width> ::= pulse width in seconds in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736

History

New in version 11.00.

:WGEN:FUNCtion:RAMP:SYMMetry

Command

```
:WGEN:FUNCtion:RAMP:SYMMetry <percent>
```

```
<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

For Ramp waveforms, the :WGEN: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

```
:WGEN:FUNCtion:RAMP:SYMMetry?
```

The :WGEN:FUNCtion:RAMP:SYMMetry? query returns the currently set ramp symmetry.

Returned Format

```
<percent><NL>
```

```
<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN:FUNCtion" on page 1736

History

New in version 11.00.

:WGEN:FUNCtion:SQUare:DCYCle

Command

```
:WGEN:FUNCtion:SQUare:DCYCle <percent>
```

```
<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

For Square waveforms, the :WGEN:FUNCtion:SQUare:DCYCle command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query

```
:WGEN:FUNCtion:SQUare:DCYCle?
```

The :WGEN:FUNCtion:SQUare:DCYCle? query returns the currently set square wave duty cycle.

Returned Format

```
<percent><NL>
```

```
<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

See Also

- ":WGEN:FUNCtion" on page 1736

History New in version 11.00.

:WGEN:MODulation:AM:DEPTh

Command

```
:WGEN:MODulation:AM:DEPTh <percent>
```

```
<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

The :WGEN: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

```
:WGEN:MODulation:AM:DEPTh?
```

The :WGEN:MODulation:AM:DEPTh? query returns the AM depth percentage setting.

Returned Format

```
<percent><NL>
```

```
<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:AM:FREQuency

Command

```
:WGEN:MODulation:AM:FREQuency <frequency>
```

```
<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN:MODulation:AM:FREQuency command specifies the frequency of the modulating signal.

Query

```
:WGEN:MODulation:AM:FREQuency?
```

The :WGEN:MODulation:AM:FREQuency? query returns the frequency of the modulating signal.

Returned Format

```
<frequency><NL>
```

```
<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History

New in version 11.00.

:WGEN:MODulation:FM:DEViation

Command

```
:WGEN:MODulation:FM:DEViation <frequency>
```

```
<frequency> ::= frequency deviation in Hz in NR3 format
```

The :WGEN: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

```
:WGEN:MODulation:FM:DEViation?
```

The :WGEN:MODulation:FM:DEViation? query returns the frequency deviation setting.

Returned Format

```
<frequency><NL>
```

```
<frequency> ::= frequency deviation in Hz in NR3 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:FM:FREQuency

Command

```
:WGEN:MODulation:FM:FREQuency <frequency>
```

```
<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN:MODulation:FM:FREQuency command specifies the frequency of the modulating signal.

Query

```
:WGEN:MODulation:FM:FREQuency?
```

The :WGEN:MODulation:FM:FREQuency? query returns the frequency of the modulating signal.

Returned Format

```
<frequency><NL>
```

```
<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:FSKey:FREQuency

Command

```
:WGEN:MODulation:FSKey:FREQuency <frequency>

<frequency> ::= hop frequency in Hz in NR3 format
```

The :WGEN:MODulation:FSKey:FREQuency command specifies the "hop frequency".

The output frequency "shifts" between the original carrier frequency and this "hop frequency".

Query

```
:WGEN:MODulation:FSKey:FREQuency?
```

The :WGEN:MODulation:FSKey:FREQuency? query returns the "hop frequency" setting.

Returned Format

```
<frequency><NL>

<frequency> ::= hop frequency in Hz in NR3 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:FSKey:RATE

Command

```
:WGEN:MODulation:FSKey:RATE <rate>
```

```
<rate> ::= FSK modulation rate in Hz in NR3 format
```

The :WGEN:MODulation:FSKey:RATE command specifies the rate at which the output frequency "shifts".

The FSK rate specifies a digital square wave modulating signal.

Query

```
:WGEN:MODulation:FSKey:RATE?
```

The :WGEN:MODulation:FSKey:RATE? query returns the FSK rate setting.

Returned Format

```
<rate><NL>
```

```
<rate> ::= FSK modulation rate in Hz in NR3 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History

New in version 11.00.

:WGEN:MODulation:FUNCtion

Command

```
:WGEN:MODulation:FUNCtion <shape>

<shape> ::= {SINusoid | SQUare| RAMP}
```

The :WGEN:MODulation:FUNCtion command specifies the shape of the modulating signal.

When the RAMP shape is selected, you can specify the amount of time per cycle that the ramp waveform is rising with the :WGEN:MODulation:FUNCtion:RAMP:SYMMetry command.

This command applies to AM and FM modulation. (The FSK modulation signal is a square wave shape.)

Query

```
:WGEN:MODulation:FUNCtion?
```

The :WGEN:MODulation:FUNCtion? query returns the specified modulating signal shape.

Returned Format

```
<shape><NL>

<shape> ::= {SIN | SQU| RAMP}
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:FUNCtion:RAMP:SYMMetry

Command

```
:WGEN:MODulation:FUNCtion:RAMP:SYMMetry <percent>
```

```
<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

The :WGEN:MODulation:FUNCtion:RAMP:SYMMetry command specifies the amount of time per cycle that the ramp waveform is rising. The ramp modulating waveform shape is specified with the :WGEN:MODulation:FUNCtion command.

Query

```
:WGEN:MODulation:FUNCtion:RAMP:SYMMetry?
```

The :WGEN:MODulation:FUNCtion:RAMP:SYMMetry? query returns ramp symmetry percentage setting.

Returned Format

```
<percent><NL>
```

```
<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:STATe" on page 1752
- ":WGEN:MODulation:TYPE" on page 1753

History

New in version 11.00.

:WGEN:MODulation:NOISe

Command

```
:WGEN:MODulation:NOISe <percent>
```

```
<percent> ::= 0 to 100
```

The :WGEN: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 TRIG OUT). This is because the trigger comparator is located after the noise source.

Query

```
:WGEN:MODulation:NOISe?
```

The :WGEN:MODulation:NOISe query returns the percent of added noise.

Returned Format

```
<percent><NL>
```

```
<percent> ::= 0 to 100
```

See Also

- ":WGEN:FUNCtion" on page 1736

History New in version 11.00.

:WGEN:MODulation:STATe

Command `:WGEN:MODulation:STATe <setting>`

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

The :WGEN: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 `:WGEN:MODulation:STATe?`

The :WGEN:MODulation:STATe? query returns whether the modulated waveform generator output is enabled of disabled.

Returned Format `<setting><NL>`

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

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749
- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:TYPE" on page 1753

History New in version 11.00.

:WGEN:MODulation:TYPE

Command

```
:WGEN:MODulation:TYPE <type>
```

```
<type> ::= {AM | FM | FSK}
```

The :WGEN: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:MODulation:AM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN: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:MODulation:FM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN:MODulation:FM:DEViation command to specify the frequency deviation from the original carrier signal frequency.
- FSK (frequency-shift keying modulation) — the output frequency "shifts" between the original carrier frequency and a "hop frequency" at the specified FSK rate.

  The FSK rate specifies a digital square wave modulating signal.

  Use the :WGEN:MODulation:FSKey:FREQuency command to specify the "hop frequency".

  Use the :WGEN:MODulation:FSKey:RATE command to specify the rate at which the output frequency "shifts".

Query

```
:WGEN:MODulation:TYPE?
```

The :WGEN:MODulation:TYPE? query returns the selected modulation type.

Returned Format

```
<type><NL>
```

```
<type> ::= {AM | FM | FSK}
```

See Also

- ":WGEN:MODulation:AM:DEPTh" on page 1743
- ":WGEN:MODulation:AM:FREQuency" on page 1744
- ":WGEN:MODulation:FM:DEViation" on page 1745
- ":WGEN:MODulation:FM:FREQuency" on page 1746
- ":WGEN:MODulation:FSKey:FREQuency" on page 1747
- ":WGEN:MODulation:FSKey:RATE" on page 1748
- ":WGEN:MODulation:FUNCtion" on page 1749

- ":WGEN:MODulation:FUNCtion:RAMP:SYMMetry" on page 1750
- ":WGEN:MODulation:STATe" on page 1752

History New in version 11.00.

:WGEN:OUTPut

Command

```
:WGEN:OUTPut <on_off>
```

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

The :WGEN:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query

```
:WGEN:OUTPut?
```

The :WGEN:OUTPut? query returns the current state of the waveform generator output setting.

Returned Format

```
<on_off><NL>
```

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

See Also

- ":WGEN:OUTPut:LOAD" on page 1756

History New in version 11.00.

:WGEN:OUTPut:LOAD

Command

```
:WGEN:OUTPut:LOAD <impedance>
```

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

The :WGEN: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

```
:WGEN:OUTPut:LOAD?
```

The :WGEN:OUTPut:LOAD? query returns the current expected output load impedance.

Returned Format

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

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

See Also

- ":WGEN:OUTPut" on page 1755

History

New in version 11.00.

:WGEN:OUTPut:MODE

Command

```
:WGEN:OUTPut:MODE <mode>

<mode> ::= {NORMal | SINGle}
```

The :WGEN:OUTPut:MODE command specifies whether the defined waveform is output continuously or as a single cycle (single-shot):

- NORMal — the defined waveform is output continuously.
- SINGle — one cycle of the defined waveform is output when you send the :WGEN:OUTPut:SINGle command.

  Not all waveform types allow single-shot output.

Query

```
:WGEN:OUTPut:MODE?
```

The :WGEN:OUTPut:MODE? query returns the output mode setting.

Returned Format

```
<mode><NL>

<mode> ::= {NORMal | SINGle}
```

See Also

- ":WGEN:OUTPut:SINGle" on page 1759

History

New in version 11.00.

:WGEN:OUTPut:POLarity

Command

```
:WGEN:OUTPut:POLarity <polarity>

<polarity> ::= {NORMal | INVerted}
```

The :WGEN:OUTPut:POLarity command specifies whether the waveform generator output is inverted.

Query

```
:WGEN:OUTPut:POLarity?
```

The :WGEN:OUTPut:POLarity? query returns the specified output polarity.

Returned Format

```
<polarity><NL>

<polarity> ::= {NORM | INV}
```

See Also

- ":WGEN:OUTPut" on page 1755

History New in version 11.00.

:WGEN:OUTPut:SINGle

Command `:WGEN:OUTPut:SINGle`

When the single-shot output mode is selected (by the :WGEN:OUTPut:MODE command), the :WGEN:OUTPut:SINGle command causes a single cycle of the defined waveform to be output.

Sending this command multiple times will interrupt a slow signal output before the cycle is completed.

See Also

- ":WGEN:OUTPut:MODE" on page 1757

History New in version 11.00.

:WGEN:PERiod

Command

```
:WGEN:PERiod <period>
```

```
<period> ::= period in seconds in NR3 format
```

For all waveforms except Noise and DC, the :WGEN:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN:FREQuency command.

Query

```
:WGEN:PERiod?
```

The :WGEN:PERiod? query returns the currently set waveform generator period.

Returned Format

```
<period><NL>
```

```
<period> ::= period in seconds in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:FREQuency" on page 1735

History New in version 11.00.

:WGEN:PRBS:LENGth

Command

```
:WGEN:PRBS:LENGth {P7M1 | P15M1 | P23M1 | P31M1}
```

When the PRBS waveform is selected (:WGEN:FUNCtion PRBS), the :WGEN:PRBS:LENGth command specifies the bit length of the PRBS pattern. The PRBS length can be: 2^7-1, 2^15-1, 2^23-1, or 2^31-1 bits.

Query

```
:WGEN:PRBS:LENGth?
```

The :WGEN:PRBS:LENGth? query returns the specified PRBS waveform pattern bit length.

Returned Format

```
<length><NL>

<length> ::= {P7M1 | P15M1 | P23M1 | P31M1}
```

See Also

- ":WGEN:FUNCtion" on page 1736

History

New in version 11.10.

Version 11:15: The options for this command have changed to P7M1, P15M1, P23M1, and P31M1 for PRBS pattern lengths 2^7-1, 2^15-1, 2^23-1, and 2^31-1, respectively.

:WGEN:RST

Command `:WGEN:RST`

The :WGEN:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:FREQuency" on page 1735

History New in version 11.00.

:WGEN:VOLTage

Command

```
:WGEN:VOLTage <amplitude>
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage command specifies the waveform's 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, when 50% of the waveform is above the offset and 50% is below (which is true of most waveform types), an amplitude of 5 V and an offset of 0 V is the same as a high-level voltage of 2.5 V and a low-level voltage of -2.5 V. (Cardiac and Sine Cardinal waveforms are 80% above the offset and 20% below, and Gaussian waveforms are 100% above the offset.)

Query

```
:WGEN:VOLTage?
```

The :WGEN:VOLTage? query returns the currently specified waveform amplitude.

Returned Format

```
<amplitude><NL>
```

```
<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:VOLTage:OFFSet" on page 1766
- ":WGEN:VOLTage:HIGH" on page 1764
- ":WGEN:VOLTage:LOW" on page 1765

History New in version 11.00.

:WGEN:VOLTage:HIGH

Command

```
:WGEN:VOLTage:HIGH <high>

<high> ::= high-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, when 50% of the waveform is above the offset and 50% is below (which is true of most waveform types), a high-level voltage of 2.5 V and a low-level voltage of -2.5 V is the same as an amplitude of 5 V and an offset of 0 V. (Cardiac and Sine Cardinal waveforms are 80% above the offset and 20% below, and Gaussian waveforms are 100% above the offset.)

Query

```
:WGEN:VOLTage:HIGH?
```

The :WGEN:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

Returned Format

```
<high><NL>

<high> ::= high-level voltage in volts, in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:VOLTage:LOW" on page 1765
- ":WGEN:VOLTage" on page 1763
- ":WGEN:VOLTage:OFFSet" on page 1766

History New in version 11.00.

:WGEN:VOLTage:LOW

Command

```
:WGEN:VOLTage:LOW <low>
```

```
<low> ::= low-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, when 50% of the waveform is above the offset and 50% is below (which is true of most waveform types), a high-level voltage of 2.5 V and a low-level voltage of -2.5 V is the same as an amplitude of 5 V and an offset of 0 V. (Cardiac and Sine Cardinal waveforms are 80% above the offset and 20% below, and Gaussian waveforms are 100% above the offset.)

Query

```
:WGEN:VOLTage:LOW?
```

The :WGEN:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

Returned Format

```
<low><NL>
```

```
<low> ::= low-level voltage in volts, in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:VOLTage:HIGH" on page 1764
- ":WGEN:VOLTage" on page 1763
- ":WGEN:VOLTage:OFFSet" on page 1766

History New in version 11.00.

:WGEN:VOLTage:OFFSet

Command

```
:WGEN:VOLTage:OFFSet <offset>
```

```
<offset> ::= offset in volts in NR3 format
```

The :WGEN:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, when 50% of the waveform is above the offset and 50% is below (which is true of most waveform types), an amplitude of 5 V and an offset of 0 V is the same as a high-level voltage of 2.5 V and a low-level voltage of -2.5 V. (Cardiac and Sine Cardinal waveforms are 80% above the offset and 20% below, and Gaussian waveforms are 100% above the offset.)

Query

```
:WGEN:VOLTage:OFFSet?
```

The :WGEN:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

Returned Format

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

```
<offset> ::= offset in volts in NR3 format
```

See Also

- ":WGEN:FUNCtion" on page 1736
- ":WGEN:VOLTage" on page 1763
- ":WGEN:VOLTage:HIGH" on page 1764
- ":WGEN:VOLTage:LOW" on page 1765

History New in version 11.00.

47 :WMEMory (Waveform Memory) Commands

The Waveform Memory Subsystem commands let you save and display waveforms, memories, and functions.

NOTE

<N> in WMEMory<R> Indicates the Waveform Memory Number

In Waveform Memory commands, the <N> in WMEMory<R> represents the waveform memory number (1-8).

:WMEMory:TIETimebase

Command `:WMEMory:TIETimebase {{ON | 1} | {OFF | 0}}`

The :WMEMory:TIETimebase command specifies whether the waveform memory horizontal scale is tied to the main horizontal time/div setting or can be adjusted separately.

Example This example ties the waveform memory horizontal scale to the main horizontal time/div setting.

```
myScope.WriteString ":WMEMory:TIETimebase ON"
```

Query `:WMEMory:TIETimebase?`

The :WMEMory:TIETimebase? query returns the state of the "tie to timebase" control.

Returned Format `[:WMEMory:TIETimebase] {1 | 0}<NL>`

History Legacy command (existed before version 3.10).

:WMEMory<R>:CLEar

Command

```
:WMEMory<R>:CLEar
```

The :WMEMory<R>:CLEar clears the associated wave memory.

<R> An integer, 1 to the number of waveform memories.

Example This example clears the waveform memory 1.

```
myScope.WriteString ":WMEMory1:CLEar"
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:DISPlay

Command

```
:WMEMory<R>:DISPlay {{ON | 1} | {OFF | 0}}
```

The :WMEMory<R>:DISPlay command enables or disables the viewing of the selected waveform memory.

<R> An integer, 1 to the number of waveform memories.

Example This example turns on the waveform memory 1 display.

```
myScope.WriteString ":WMEMory1:DISPlay ON"
```

Query

```
:WMEMory<R>:DISPlay?
```

The :WMEMory<R>:DISPlay? query returns the state of the selected waveform memory.

Returned Format

```
[:WMEMory<R>:DISPlay] {1 | 0}<NL>
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:FFT:HSCale

Command

```
:WMEMory<R>:FFT:HSCale {LINear | LOG}
```

For a FFT waveform memory, the :WMEMory<R>:FFT:HSCale command specifies whether the horizontal scale is linear or logarithmic.

Query

```
:WMEMory<R>:FFT:HSCale?
```

The :WMEMory<R>:FFT:HSCale? query returns the horizontal scale setting.

Returned Format

```
<type><NL>
```

```
<type> ::= {LIN | LOG}
```

See Also

- ":FUNCtion<F>:FFT:HSCale" on page 627

History

New in version 10.10.

:WMEMory<R>:LABel

Command

```
:WMEMory<R>:LABel <quoted_string>
```

The :WMEMory<R>:LABel command sets the waveform memory label to the quoted string.

Labels can be enabled with the :DISPlay:LABel command.

<R> An integer, 1 to the number of waveform memories.

<quoted_string> A series of 16 or fewer characters as a quoted ASCII string.

Query

```
:WMEMory<R>:LABel?
```

The :WMEMory<R>:LABel? query returns the label of the specified waveform memory.

Returned Format

```
[:WMEMory<R>:LABel] <quoted_string><NL>
```

See Also

- ":DISPlay:LABel" on page 584
- ":CHANnel<N>:LABel" on page 467
- ":FUNCtion<F>:LABel" on page 659

History New in version 11.15.

:WMEMory<R>:LOAD

Command

```
:WMEMory<R>:LOAD <file_name>
```

The :WMEMory<R>:LOAD command loads an oscilloscope waveform memory location with a waveform from a file that has an internal waveform format (extension .wfm), comma separated xypairs, (extension .csv), tab separated xypairs (extension .tsv), and yvalues text (extension .txt). You can load the file from either the c: or a: drive, or any lan connected drive. See the examples below.

The oscilloscope assumes that the default path for waveforms is C:\Users\Public\Documents\Infiniium. To use a different path, specify the path and file name completely.

<R> An integer, 1 to the number of waveform memories.

<file_name> A quoted string which specifies the file to load, and has a .wfm, .csv, .tsv, or .txt extension.

Examples This example loads waveform memory 4 with a file.

```
myScope.WriteString _
":WMEMory4:LOAD ""C:\Users\Public\Documents\Infiniium\waveform.wfm"""
```

This example loads waveform memory 3 with a file that has the internal waveform format and is stored on drive U:.

```
myScope.WriteString ":WMEMory3:LOAD ""U:\waveform.wfm"""
```

Related Commands

```
:DISK:LOAD
:DISK:STORe
```

See Also

- ":DISK:LOAD" on page 532
- ":DISK:SAVE:SETup" on page 545
- ":DISK:SAVE:WAVeform" on page 546

History Legacy command (existed before version 3.10).

:WMEMory<R>:SAVE

Command

```
:WMEMory<R>:SAVE <source>
```

The :WMEMory<R>:SAVE command stores the specified channel, waveform memory, or function to the waveform memory. You can save waveforms to waveform memories regardless of whether the waveform memory is displayed or not.

The :WAVeform:VIEW command determines the view of the data being saved.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | CLOCk | FUNCtion<F> | EQUalized<L> | MTRend | MSPectrum | WMEMory<R> | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR}

For more information on <source> parameters, see Chapter 50, “Waveform Sources,” starting on page 2011.

Example This example saves channel 1 to waveform memory 4.

```
myScope.WriteString ":WMEMory4:SAVE CHANnel1"
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:SEGMented:COUNt?

Query `:WMEMory<R>:SEGMented:COUNt?`

When segmented memory acquisitions are saved to waveform memory, the :WMEMory<R>:SEGMented:COUNt? query returns the number of segments in the waveform memory.

Returned Format `<#segments><NL>`

`<#segments> ::= integer in NR1 format`

See Also

- ":WMEMory<R>:SEGMented:COUNt?" on page 1775
- ":WMEMory<R>:SEGMented:INDex" on page 1776
- ":WMEMory<R>:SEGMented:PLAY" on page 1777

History New in version 6.00.

:WMEMory<R>:SEGMented:INDex

Command

```
:WMEMory<R>:SEGMented:INDex <number>
```

When segmented memory acquisitions are saved to waveform memory, the :WMEMory<R>:SEGMented:INDex command displays the waveform segment at the specified index.

<number> Segment number in NR1 format.

Query

```
:WMEMory<R>:SEGMented:INDex?
```

The :WMEMory<R>:SEGMented:INDex? query returns the index of the waveform segment that is currently being displayed.

Returned Format

```
<number><NL>
```

```
<number> ::= segment number in NR1 format
```

See Also

- ":WMEMory<R>:SEGMented:COUNt?" on page 1775
- ":WMEMory<R>:SEGMented:INDex" on page 1776
- ":WMEMory<R>:SEGMented:PLAY" on page 1777

History New in version 6.00.

:WMEMory<R>:SEGMented:PLAY

Command

```
:WMEMory<R>:SEGMented:PLAY {{0 | OFF} | {1 | ON}}
```

When segmented memory acquisitions are saved to waveform memory, the ":WMEMory<R>:SEGMented:PLAY" command plays (or stops) acquired segments.

- ON — is the similar to clicking the Play button in the graphical user interface, except that the display is not updated while segments are played.
- OFF — is the same as clicking the Stop button in the graphical user interface.

Playing waveform memory segments can take a while depending on the analysis taking place. You can query to determine when playing is complete.

The play rate is the same rate set by :ACQuire:SEGMented:PRATe command.

Query

```
:WMEMory<R>:SEGMented:PLAY?
```

The :WMEMory<R>:SEGMented:PLAY? query returns whether segments are currently being played (1) or are stopped (0).

NOTE

The :WMEMory<R>:SEGMented:PLAY? query is different than the :ACQuire:SEGMented:PLAY? query in that it waits until all segments are played before it returns.

Returned Format

```
[:WMEMory<R>:SEGMented:PLAY] <status><NL>

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

See Also

- ":ACQuire:SEGMented:PRATe" on page 321
- ":WMEMory<R>:SEGMented:COUNt?" on page 1775
- ":WMEMory<R>:SEGMented:INDex" on page 1776

History New in version 6.00.

:WMEMory<R>:XOFFset

Command

```
:WMEMory<R>:XOFFset <offset_value>
```

The :WMEMory<R>:XOFFset command sets thex-axis, horizontal position for the selected waveform memory's display scale. The position is referenced to center screen.

<R> An integer, 1 to the number of waveform memories.

<offset_value> A real number for the horizontal offset (position) value.

Example This example sets the X-axis, horizontal position for waveform memory 3 to 0.1 seconds (100 ms).

```
myScope.WriteString ":WMEMory3:XOFFset 0.1"
```

Query

```
:WMEMory<R>:XOFFset?
```

The :WMEMory<R>:XOFFset? query returns the current X-axis, horizontal position for the selected waveform memory.

Returned Format

```
[:WMEMory<R>:XOFFset] <offset_value><NL>
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:XRANge

Command

```
:WMEMory<R>:XRANge <range_value>
```

The :WMEMory<R>:XRANge command sets theX-axis, horizontal range for the selected waveform memory's display scale. The horizontal scale is the horizontal range divided by 10.

<R> An integer, 1 to the number of waveform memories.

<range_value> A real number for the horizontal range value.

Example This example sets the X-axis, horizontal range of waveform memory 2 to 435 microseconds.

```
myScope.WriteString ":WMEMory2:XRANge 435E-6"
```

Query

```
:WMEMory<R>:XRANge?
```

The :WMEMory<R>:XRANge? query returns the current X-axis, horizontal range for the selected waveform memory.

Returned Format

```
[:WMEMory<R>:XRANge] <range_value><NL>
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:YOFFset

Command

```
:WMEMory<R>:YOFFset <offset_value>
```

The :WMEMory<R>:YOFFset command sets the Y-axis (vertical axis) offset for the selected waveform memory.

<R> An integer, 1 to the number of waveform memories.

<offset_value> A real number for the vertical offset value.

Example This example sets the Y-axis (vertical) offset of waveform memory 2 to 0.2V.

```
myScope.WriteString ":WMEMory2:YOFFset 0.2"
```

Query

```
:WMEMory<R>:YOFFset?
```

The :WMEMory<R>:YOFFset? query returns the current Y-axis (vertical) offset for the selected waveform memory.

Returned Format

```
[:WMEMory<R>:YOFFset] <offset_value><NL>
```

History Legacy command (existed before version 3.10).

:WMEMory<R>:YRANge

Command

```
:WMEMory<R>:YRANge <range_value>
```

The :WMEMory<R>:YRANge command sets the Y-axis, vertical range for the selected memory. The vertical scale is the vertical range divided by 8.

<R> An integer, 1 to the number of waveform memories.

<range_value> A real number for the vertical range value.

Example This example sets the Y-axis (vertical) range of waveform memory 3 to 0.2 volts.

```
myScope.WriteString ":WMEMory3:YRANge 0.2"
```

Query

```
:WMEMory<R>:YRANge?
```

The :WMEMory<R>:YRANge? query returns the Y-axis, vertical range for the selected memory.

Returned Format

```
[:WMEMory<R>:YRANge]<range_value><NL>
```

History Legacy command (existed before version 3.10).

48 :XTALk (Crosstalk Analysis) Commands

The XTALk commands and queries control the Crosstalk Analysis application. This application helps you troubleshoot and characterize crosstalk on up to four simultaneously-acquired signals.

:XTALk:ENABle

Command

```
:XTALk:ENABle {{0 | OFF} | {1 | ON}}
```

The :XTALk:ENABle command enables or disables crosstalk analysis.

Query

```
:XTALk:ENABle?
```

The :XTALk:ENABle? query returns whether crosstalk analysis is enabled or disabled.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk:RESults?" on page 1799
- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAISi" on page 1803
- ":XTALk:SAIFilter" on page 1802
- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAXFilter" on page 1805
- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAISi" on page 1789
- ":XTALk:PAIFilter" on page 1788
- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAXFilter" on page 1791
- ":XTALk:PJADeskew" on page 1793

- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJISi" on page 1795
- ":XTALk:PJIFilter" on page 1794
- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJXFilter" on page 1797

History New in version 5.70.

:XTALk:PAADeskew

Command `:XTALk:PAADeskew {{0 | OFF} | {1 | ON}}`

The :XTALk:PAADeskew command is an advanced configuration option for power supply aggressors (amplitude) that specifies whether auto deskew is enabled or disabled.

When auto deskew is enabled, the Crosstalk Analysis application uses a proprietary cross-correlation method for aligning the waveforms in time. The algorithm searches over a finite range of delays to find the best possible alignment. The search range corresponds to about 1 m of difference in cable length, which is sufficient for most situations.

When auto deskew is disabled, the deskew time limit is specified by the :XTALk:PASLimit command.

Query `:XTALk:PAADeskew?`

The :XTALk:PAADeskew? query returns the "auto deskew" setting for power supply aggressors (amplitude).

Returned Format `<setting><NL>`

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

See Also

- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAISi" on page 1789
- ":XTALk:PAIFilter" on page 1788
- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAXFilter" on page 1791

History New in version 5.70.

:XTALk:PAIFilter

Command

```
:XTALk:PAIFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:PAIFilter command is an advanced configuration option for power supply aggressors (amplitude) that, when auto limit ISI filter time is disabled (:XTALk:PAISi OFF), lets you specify the ISI filter time span.

Query

```
:XTALk:PAIFilter?
```

The :XTALk:PAIFilter? query returns the specified ISI filter time span.

Returned Format

```
<time_span><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:PAISi" on page 1789
- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAXFilter" on page 1791

History

New in version 5.70.

:XTALk:PAISi

Command

```
:XTALk:PAISi {{0 | OFF} | {1 | ON}}
```

The :XTALk:PAISi command is an advanced configuration option for power supply aggressors (amplitude) that specifies whether auto limit ISI filter time is enabled or disabled.

The ISI filter represents the channel model and can therefore convert an ideal input waveform into the one that is band-limited and containing reflections.

- When auto limit ISI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal ISI filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit ISI filter time is disabled, the ISI filter time span is specified by the :XTALk:PAIFilter command.

Query

```
:XTALk:PAISi?
```

The :XTALk:PAISi? query returns the "auto limit ISI filter time" setting for power supply aggressors (amplitude).

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk:PAIFilter" on page 1788
- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAXFilter" on page 1791

History New in version 5.70.

:XTALk:PASLimit

Command

```
:XTALk:PASLimit <time_limit>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

The :XTALk:PASLimit command is an advanced configuration option for power supply aggressors (amplitude) that, when auto deskew is disabled (:XTALk:PAADeskew OFF), lets you specify the deskew time limit (search range).

Notice that the default deskew time limit value is larger for a power supply than for serial data transmission lines. This is because buffers and other circuit components can create delays much larger than the propagation distance. These delays may vary greatly from one circuit to another, so keep in mind that changing the default value may produce better results.

Query

```
:XTALk:PASLimit?
```

The :XTALk:PASLimit? query returns the specified deskew time limit.

Returned Format

```
<time_limit><NL>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

See Also

- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PAISi" on page 1789
- ":XTALk:PAIFilter" on page 1788
- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAXFilter" on page 1791

History New in version 5.70.

:XTALk:PAXFilter

Command

```
:XTALk:PAXFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:PAXFilter command is an advanced configuration option for power supply aggressors (amplitude) that, when auto limit XSI filter time is disabled (:XTALk:PAXSi OFF), lets you specify the XSI (crosstalk) filter time span.

Query

```
:XTALk:PAXFilter?
```

The :XTALk:PAXFilter? query returns the specified XSI (crosstalk) filter time span.

Returned Format

```
<time_span><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:PAXSi" on page 1792
- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAISi" on page 1789
- ":XTALk:PAIFilter" on page 1788

History

New in version 5.70.

:XTALk:PAXSi

Command `:XTALk:PAXSi {{0 | OFF} | {1 | ON}}`

The :XTALk:PAXSi command is an advanced configuration option for power supply aggressors (amplitude) that specifies whether auto limit XSI filter time is enabled or disabled

Crosstalk filters describe how an aggressor signal is transformed into a crosstalk signal (such as NEXT or FEXT), and what magnitude it will have.

- When auto limit XSI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal crosstalk filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit XSI filter time is disabled, the XSI (crosstalk) filter time span is specified by the :XTALk:PAXFilter command.

  A FEXT filter length should be at least as long as an edge rise time, and a NEXT filter length should be at least twice the propagation delay of the channel.

Query `:XTALk:PAXSi?`

The :XTALk:PAXSi? query returns the "auto limit XSI filter time" setting for power supply aggressors (amplitude).

Returned Format `<setting><NL>`

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

See Also

- ":XTALk:PAXFilter" on page 1791
- ":XTALk:PAADeskew" on page 1787
- ":XTALk:PASLimit" on page 1790
- ":XTALk:PAISi" on page 1789
- ":XTALk:PAIFilter" on page 1788

History New in version 5.70.

:XTALk:PJADeskew

Command

```
:XTALk:PJADeskew {{0 | OFF} | {1 | ON}}
```

The :XTALk:PJADeskew command is an advanced configuration option for power supply aggressors (jitter) that specifies whether auto deskew is enabled or disabled.

When auto deskew is enabled, the Crosstalk Analysis application uses a proprietary cross-correlation method for aligning the waveforms in time. The algorithm searches over a finite range of delays to find the best possible alignment. The search range corresponds to about 1 m of difference in cable length, which is sufficient for most situations.

When auto deskew is disabled, the deskew time limit is specified by the :XTALk:PJSLimit command.

Query

```
:XTALk:PJADeskew?
```

The :XTALk:PJADeskew? query returns the "auto deskew" setting for power supply aggressors (jitter).

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJISi" on page 1795
- ":XTALk:PJIFilter" on page 1794
- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJXFilter" on page 1797

History New in version 5.70.

:XTALk:PJIFilter

Command

```
:XTALk:PJIFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:PJIFilter command is an advanced configuration option for power supply aggressors (amplitude) that, when auto limit ISI filter time is disabled (:XTALk:PJISi OFF), lets you specify the ISI filter time span.

Query

```
:XTALk:PJIFilter?
```

The :XTALk:PJIFilter? query returns the specified ISI filter time span.

Returned Format

```
<time_span><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:PJISi" on page 1795
- ":XTALk:PJADeskew" on page 1793
- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJXFilter" on page 1797

History

New in version 5.70.

:XTALk:PJISi

Command `:XTALk:PJISi {{0 | OFF} | {1 | ON}}`

The :XTALk:PJISi command is an advanced configuration option for power supply aggressors (jitter) that specifies whether auto limit ISI filter time is enabled or disabled.

The ISI filter represents the channel model and can therefore convert an ideal input waveform into the one that is band-limited and containing reflections.

- When auto limit ISI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal ISI filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit ISI filter time is disabled, the ISI filter time span is specified by the :XTALk:PJIFilter command.

Query `:XTALk:PJISi?`

The :XTALk:PJISi? query returns the "auto limit ISI filter time" setting for power supply aggressors (jitter).

Returned Format `<setting><NL>`

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

See Also

- ":XTALk:PJIFilter" on page 1794
- ":XTALk:PJADeskew" on page 1793
- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJXFilter" on page 1797

History New in version 5.70.

:XTALk:PJSLimit

Command

```
:XTALk:PJSLimit <time_limit>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

The :XTALk:PJSLimit command is an advanced configuration option for power supply aggressors (jitter) that, when auto deskew is disabled (:XTALk:PJADeskew OFF), lets you specify the deskew time limit (search range).

Notice that the default deskew time limit value is larger for a power supply than for serial data transmission lines. This is because buffers and other circuit components can create delays much larger than the propagation distance. These delays may vary greatly from one circuit to another, so keep in mind that changing the default value may produce better results.

Query

```
:XTALk:PJSLimit?
```

The :XTALk:PJSLimit? query returns the specified deskew time limit.

Returned Format

```
<time_limit><NL>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

See Also

- ":XTALk:PJADeskew" on page 1793
- ":XTALk:PJISi" on page 1795
- ":XTALk:PJIFilter" on page 1794
- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJXFilter" on page 1797

History New in version 5.70.

:XTALk:PJXFilter

Command

```
:XTALk:PJXFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:PJXFilter command is an advanced configuration option for power supply aggressors (amplitude) that, when auto limit XSI filter time is disabled (:XTALk:PJXSi OFF), lets you specify the XSI (crosstalk) filter time span.

Query

```
:XTALk:PJXFilter?
```

The :XTALk:PJXFilter? query returns the specified XSI (crosstalk) filter time span.

Returned Format

```
<opt><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:PJXSi" on page 1798
- ":XTALk:PJADeskew" on page 1793
- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJISi" on page 1795
- ":XTALk:PJIFilter" on page 1794

History

New in version 5.70.

:XTALk:PJXSi

Command

```
:XTALk:PJXSi {{0 | OFF} | {1 | ON}}
```

The :XTALk:PJXSi command is an advanced configuration option for power supply aggressors (jitter) that specifies whether auto limit XSI filter time is enabled or disabled

Crosstalk filters describe how an aggressor signal is transformed into a crosstalk signal (such as NEXT or FEXT), and what magnitude it will have.

- When auto limit XSI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal crosstalk filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit XSI filter time is disabled, the XSI (crosstalk) filter time span is specified by the :XTALk:PJXFilter command.

  A FEXT filter length should be at least as long as an edge rise time, and a NEXT filter length should be at least twice the propagation delay of the channel.

Query

```
:XTALk:PJXSi?
```

The :XTALk:PJXSi? query returns the "auto limit XSI filter time" setting for power supply aggressors (jitter).

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk:PJXFilter" on page 1797
- ":XTALk:PJADeskew" on page 1793
- ":XTALk:PJSLimit" on page 1796
- ":XTALk:PJISi" on page 1795
- ":XTALk:PJIFilter" on page 1794

History

New in version 5.70.

:XTALk:RESults?

Query `:XTALk:RESults?`

The :XTALk:RESults? query returns the crosstalk analysis results in a comma-separated list of values. The values returned for each victim:aggressor pair in the results are:

Value	Format	Example
Label:	string	c1:c2
Amplitude skew:	floating-point number string in scientific notation	0.0E+00
Jitter skew:	floating-point number string in scientific notation	0.0E+00
Volt, Error(rms), for the non victim:victim lines:	floating-point number string in scientific notation	0.0E+00
Volt, Error(p-p), for the non victim:victim lines:	floating-point number string in scientific notation	0.0E+00
Time, Error(rms), for the non victim:victim lines:	floating-point number string in scientific notation	0.0E+00
Time Error(p-p), for the non victim:victim lines:	floating-point number string in scientific notation	0.0E+00
V high, Error(rms):	floating-point number string in scientific notation	0.0E+00
V high, Error(p-p):	floating-point number string in scientific notation	0.0E+00
V low, Error(rms):	floating-point number string in scientific notation	0.0E+00
V low, Error(p-p):	floating-point number string in scientific notation	0.0E+00
Volt, Error(rms), for the victim:victim line:	floating-point number string in scientific notation	0.0E+00
Volt, Error(p-p), for the victim:victim line:	floating-point number string in scientific notation	0.0E+00
Time, Error(rms), for the victim:victim line:	floating-point number string in scientific notation	0.0E+00
Time, Error(p-p), for the victim:victim line:	floating-point number string in scientific notation	0.0E+00

Returned Format `<results_list><NL>`

```
<results_list> ::= comma-delimited list of values.
```

See Also

- ":XTALk:ENABle" on page 1785

History New in version 5.70.

:XTALk:SAADeskew

Command

```
:XTALk:SAADeskew {{0 | OFF} | {1 | ON}}
```

The :XTALk:SAADeskew command is an advanced configuration option for serial data aggressors that specifies whether auto deskew is enabled or disabled.

When auto deskew is enabled, the Crosstalk Analysis application uses a proprietary cross-correlation method for aligning the waveforms in time. The algorithm searches over a finite range of delays to find the best possible alignment. The search range corresponds to about 1 m of difference in cable length, which is sufficient for most situations.

When auto deskew is disabled, the deskew time limit is specified by the :XTALk:SASLimit command.

Query

```
:XTALk:SAADeskew?
```

The :XTALk:SAADeskew? query returns the "auto deskew" setting for serial data aggressors.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAISi" on page 1803
- ":XTALk:SAIFilter" on page 1802
- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAXFilter" on page 1805

History New in version 5.70.

:XTALk:SAIFilter

Command

```
:XTALk:SAIFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:SAIFilter command is an advanced configuration option for serial data aggressors that, when auto limit ISI filter time is disabled (:XTALk:SAISi OFF), lets you specify the ISI filter time span.

Query

```
:XTALk:SAIFilter?
```

The :XTALk:SAIFilter? query returns the specified ISI filter time span.

Returned Format

```
<time_span><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:SAISi" on page 1803
- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAXFilter" on page 1805

History New in version 5.70.

:XTALk:SAISi

Command

```
:XTALk:SAISi {{0 | OFF} | {1 | ON}}
```

The :XTALk:SAISi command is an advanced configuration option for serial data aggressors that specifies whether auto limit ISI filter time is enabled or disabled.

The ISI filter represents the channel model and can therefore convert an ideal input waveform into the one that is band-limited and containing reflections.

- When auto limit ISI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal ISI filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit ISI filter time is disabled, the ISI filter time span is specified by the :XTALk:SAIFilter command.

Query

```
:XTALk:SAISi?
```

The :XTALk:SAISi? query returns the "auto limit ISI filter time" setting for serial data aggressors.

Returned Format

```
<setting><NL>
```

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

See Also

- ":XTALk:SAIFilter" on page 1802
- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAXFilter" on page 1805

History

New in version 5.70.

:XTALk:SASLimit

Command

```
:XTALk:SASLimit <time_limit>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

The :XTALk:SASLimit command is an advanced configuration option for serial data aggressors that, when auto deskew is disabled (:XTALk:SAADeskew OFF), lets you specify the deskew time limit (search range).

Notice that the default deskew time limit value is larger for a power supply than for serial data transmission lines. This is because buffers and other circuit components can create delays much larger than the propagation distance. These delays may vary greatly from one circuit to another, so keep in mind that changing the default value may produce better results.

Query

```
:XTALk:SASLimit?
```

The :XTALk:SASLimit? query returns the specified deskew time limit.

Returned Format

```
<time_limit><NL>
```

```
<time_limit> ::= time limit in seconds in NR3 format.
```

See Also

- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SAISi" on page 1803
- ":XTALk:SAIFilter" on page 1802
- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAXFilter" on page 1805

History

New in version 5.70.

:XTALk:SAXFilter

Command

```
:XTALk:SAXFilter <time_span>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

The :XTALk:SAXFilter command is an advanced configuration option for serial data aggressors that, when auto limit XSI filter time is disabled (:XTALk:SAXSi OFF), lets you specify the XSI (crosstalk) filter time span.

Query

```
:XTALk:SAXFilter?
```

The :XTALk:SAXFilter? query returns the specified XSI (crosstalk) filter time span.

Returned Format

```
<time_span><NL>
```

```
<time_span> ::= time span in seconds in NR3 format.
```

See Also

- ":XTALk:SAXSi" on page 1806
- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAISi" on page 1803
- ":XTALk:SAIFilter" on page 1802

History

New in version 5.70.

:XTALk:SAXSi

Command

```
:XTALk:SAXSi {{0 | OFF} | {1 | ON}}
```

The :XTALk:SAXSi command is an advanced configuration option for serial data aggressors that specifies whether auto limit XSI filter time is enabled or disabled

Crosstalk filters describe how an aggressor signal is transformed into a crosstalk signal (such as NEXT or FEXT), and what magnitude it will have.

- When auto limit XSI filter time is enabled, the Crosstalk Analysis application uses a proprietary algorithm to automatically determine an ideal crosstalk filter length for a given situation. The application further tries to optimize the shape of the filter by adaptively placing more taps in areas that are needed.
- When auto limit XSI filter time is disabled, the XSI (crosstalk) filter time span is specified by the :XTALk:SAXFilter command.

  A FEXT filter length should be at least as long as an edge rise time, and a NEXT filter length should be at least twice the propagation delay of the channel.

Query

```
:XTALk:SAXSi?
```

The :XTALk:SAXSi? query returns the "auto limit XSI filter time" setting for serial data aggressors.

Returned Format

```
<setting><NL>
```

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

See Also

- ":XTALk:SAXFilter" on page 1805
- ":XTALk:SAADeskew" on page 1801
- ":XTALk:SASLimit" on page 1804
- ":XTALk:SAISi" on page 1803
- ":XTALk:SAIFilter" on page 1802

History

New in version 5.70.

:XTALk<X>:AENable<X>

Command

```
:XTALk<X>:AENable<X> {{0 | OFF} | {1 | ON}}
```

When the crosstalk analysis signal is a victim (:XTALk<X>:IVICtim ON), the :XTALk<X>:AENable<X> command specifies whether to remove the crosstalk from another signal in the crosstalk analysis.

<X> An integer from 1-4.

Example For example if crosstalk signal1 is a victim, you can specify to remove the crosstalk from the signal3 aggressor with the command:

```
myScope.WriteString ":XTALk1:AENable3 ON"
```

See Also

- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:ENABle

Command

```
:XTALk<X>:ENABle {{0 | OFF} | {1 | ON}}
```

The :XTALk<X>:ENABle command adds or removes a signal from the crosstalk analysis.

Crosstalk analysis can be performed on up to four simultaneously acquired signals.

<X> An integer from 1-4.

Query

```
:XTALk<X>:ENABle?
```

The :XTALk<X>:ENABle? query returns whether the signal has been added to the crosstalk analysis.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:IAGGressor

Command

```
:XTALk<X>:IAGGressor {{0 | OFF} | {1 | ON}}
```

The :XTALk<X>:IAGGressor command specifies whether the signal is an aggressor.

<X> An integer from 1-4.

Query

```
:XTALk<X>:IAGGressor?
```

The :XTALk<X>:IAGGressor? query returns the "is aggressor" setting.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:IVICtim

Command

```
:XTALk<X>:IVICtim {{0 | OFF} | {1 | ON}}
```

The :XTALk<X>:IVICtim command specifies whether the signal is a victim.

<X> An integer from 1-4.

Query

```
:XTALk<X>:IVICtim?
```

The :XTALk<X>:IVICtim? query returns the "is victim" setting.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:PAUTo

Command

```
:XTALk<X>:PAUTo {{0 | OFF} | {1 | ON}}
```

When the crosstalk analysis signal type is DIGital (serial data), the :XTALk<X>:PAUTo command specifies whether the pattern length is automatically determined.

<X> An integer from 1-4.

Query

```
:XTALk<X>:PAUTo?
```

The :XTALk<X>:PAUTo? query returns the auto length setting.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:PLENgth

Command

```
:XTALk<X>:PLENgth <number_of_bits>
```

```
<number_of_bits> ::= integer from 2-1024.
```

When the crosstalk analysis signal type is DIGital (serial data) and the pattern length is not automatically determined (:XTALk<X>:PAUTo OFF), the :XTALk<X>:PLENgth command specifies the pattern length.

<X> An integer from 1-4.

Query

```
:XTALk<X>:PLENgth?
```

The :XTALk<X>:PLENgth? query returns the pattern length setting.

Returned Format

```
<number_of_bits><NL>
```

```
<number_of_bits> ::= integer from 2-1024.
```

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:PTYPe

Command

```
:XTALk<X>:PTYPe <pattern_type>
```

```
<pattern_type> ::= {PERiodic | ARBitrary}
```

When the crosstalk analysis signal type is DIGital (serial data), the :XTALk<X>:PTYPe command specifies whether the pattern is periodic or arbitrary.

<X> An integer from 1-4.

Query

```
:XTALk<X>:PTYPe?
```

The :XTALk<X>:PTYPe? query returns the pattern type setting.

Returned Format

```
<pattern_type><NL>
```

```
<pattern_type> ::= {PERiodic | ARBitrary}
```

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:RIDeal

Command

```
:XTALk<X>:RIDeal {{0 | OFF} | {1 | ON}}
```

When the crosstalk analysis signal is a victim (:XTALk<X>:IVICtim ON), the :XTALk<X>:RIDeal command specifies whether to remove the signal's ideal waveform contribution should be removed from the crosstalk-removed waveform.

<X> An integer from 1-4.

Query

```
:XTALk<X>:RIDeal?
```

The :XTALk<X>:RIDeal? query returns the "remove ideal waveform" setting.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:RISI

Command

```
:XTALk<X>:RISI {{0 | OFF} | {1 | ON}}
```

When the crosstalk analysis signal is a victim (:XTALk<X>:IVICtim ON), the :XTALk<X>:RISI command specifies whether to remove the signal's ISI (inter-symbol interference) contribution should be removed from the crosstalk-removed waveform.

<X> An integer from 1-4.

Query

```
:XTALk<X>:RISI?
```

The :XTALk<X>:RISI? query returns the "remove ISI" setting.

Returned Format

```
<setting><NL>

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:ROTHer

Command

```
:XTALk<X>:ROTHer {{0 | OFF} | {1 | ON}}
```

When the crosstalk analysis signal is a victim (:XTALk<X>:IVICtim ON), the :XTALk<X>:ROTHer command specifies whether to remove the signal's unknown crosstalk and noise contribution should be removed from the crosstalk-removed waveform.

<X> An integer from 1-4.

Query

```
:XTALk<X>:ROTHer?
```

The :XTALk<X>:ROTHer? query returns the "remove unknown crosstalk and noise" setting.

Returned Format

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

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:SOURce

Command

```
:XTALk<X>:SOURce <source>

<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F>
   | WMEMory<N> | EQUalized<L> | MTRend}
```

The :XTALk<X>:SOURce command specifies the source of the crosstalk signal.

<X> An integer from 1-4.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer from 1-16.

<L> An integer from 1-4.

Query

```
:XTALk<X>:SOURce?
```

The :XTALk<X>:SOURce? query returns the specified source of the crosstalk signal.

Returned Format

```
<source><NL>

<source> ::= {CHAN<N> | FUNC<F> | WMEM<N> | EQU<L> | MTR}
```

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813
- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:STYPe" on page 1819
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

:XTALk<X>:STYPe

Command

```
:XTALk<X>:STYPe <signal_type>
```

```
<signal_type> ::= {POWer | ANALog | DIGital}
```

The :XTALk<X>:STYPe command specifies the crosstalk analysis signal type:

- POWer — Tells the application to use the specialized algorithms for power supply analysis.

NOTE You cannot have both a power supply victim and a power supply aggressor in the same setup because these require different algorithms.

- ANALog — For increased accuracy, it is recommended to use clock recovery (DIGital) for all data waveforms whenever possible; however, aggressors that are too noisy for clock recovery can be specified as ANALog, which tells the algorithm to skip clock recovery.

  If a victim signal is too noisy for clock recovery, another option is to perform equalization on the waveform before sending it to the Crosstalk Analysis application. This can be done directly on the oscilloscope using the Infiniium Equalization application and then selecting FFE as the signal source for crosstalk analysis.

- DIGital — Tells the application that the input signal represents a digital bit stream (serial data), and that it needs to do a clock recovery.

  When the signal type is data DIGital, you need to specify the type of pattern (periodic or arbitrary) and whether the pattern length should be automatically determined or specified with :XTALk<X>:PLENgth.

<X> An integer from 1-4.

Query

```
:XTALk<X>:STYPe?
```

The :XTALk<X>:STYPe? query returns the specified signal type.

Returned Format

```
<signal_type><NL>
```

```
<signal_type> ::= {POW | ANAL | DIG}
```

See Also

- ":XTALk<X>:AENable<X>" on page 1807
- ":XTALk<X>:ENABle" on page 1808
- ":XTALk<X>:IAGGressor" on page 1809
- ":XTALk<X>:IVICtim" on page 1810
- ":XTALk<X>:PAUTo" on page 1811
- ":XTALk<X>:PLENgth" on page 1812
- ":XTALk<X>:PTYPe" on page 1813

- ":XTALk<X>:RIDeal" on page 1814
- ":XTALk<X>:RISI" on page 1815
- ":XTALk<X>:ROTHer" on page 1816
- ":XTALk<X>:SOURce" on page 1817
- ":XTALk:ENABle" on page 1785
- ":XTALk:RESults?" on page 1799

History New in version 5.70.

49 Obsolete and Discontinued Commands

Obsolete commands are deprecated, older forms of commands that still work but have been replaced by newer commands.

Obsolete Command	Current Command Equivalent	Behavior Differences
:ACQuire:HRESolution (see page 1835)	:ACQuire:ADCRes (see page 292)	The :ACQuire:HRESolution is for backwards compatibility. It does not provide access to all the High Resolution Oscilloscope (HRO) settings.
:ANALyze:CLOCk:METHod:PAM:B03 (see page 1838)	None	The command works and as before, but changing its default (ON) is not necessary.
:ANALyze:CLOCk:METHod:PAM:B12 (see page 1840)	None	The command works and as before, but changing its default (ON) is not necessary.
:ANALyze:CLOCk:METHod:PAM:NONSymmetric (see page 1842)	None	The command works and as before, but changing its default (OFF) is not necessary.
:DISPlay:COLumn (see page 1845)	:DISPlay:BOOKmark<N>:XPOSition (see page 555)	Bookmarks are now the method used to place text strings or annotations on screen.
:DISPlay:LINE (see page 1846)	:DISPlay:BOOKmark<N>:SET (see page 553)	
:DISPlay:ROW (see page 1847)	:DISPlay:BOOKmark<N>:YPOSition (see page 556)	
:DISPlay:STRing (see page 1848)	:DISPlay:BOOKmark<N>:SET (see page 553)	
:DISPlay:TAB (see page 1849)	None	This command is not supported in version 5.00 or higher. The query now returns only NONE.

Obsolete Command	Current Command Equivalent	Behavior Differences
:DISPlay:TEXT (see page 1850)	:DISPlay:BOOKmark<N>:DELete (see page 552)	Bookmarks are now the method used to place text strings or annotations on screen.
:MEASure:CLOCk:METHod (see page 1865)	▪ :MEASure:CLOCk:METHod (see page 1863) ▪ :MEASure:CLOCk:METHod:JTF (see page 1871) ▪ :MEASure:CLOCk:METHod:OJTF (see page 1873)	The command options for specifying clock recovery PLL options have been moved to the new commands :MEASure:CLOCk:METHod:JTF and :MEASure:CLOCk:METHod:OJTF.
:MEASure:CLOCk (see page 1862)	:ANALyze:CLOCk (see page 333)	There are no differences in behavior. This is just a remapping of commands into a new subsystem.
:MEASure:CLOCk:METHod (see page 1863)	:ANALyze:CLOCk:METHod (see page 334)	
:MEASure:CLOCk:METHod:ALIGn (see page 1867)	:ANALyze:CLOCk:METHod:ALIGn (see page 338)	
:MEASure:CLOCk:METHod:DEEMphasis (see page 1868)	:ANALyze:CLOCk:METHod:DEEMphasis (see page 339)	
:MEASure:CLOCk:METHod:EDGE (see page 1869)	:ANALyze:CLOCk:METHod:EDGE (see page 340)	
:MEASure:CLOCk:METHod:JTF (see page 1871)	:ANALyze:CLOCk:METHod:JTF (see page 343)	
:MEASure:CLOCk:METHod:OJTF (see page 1873)	:ANALyze:CLOCk:METHod:OJTF (see page 345)	
:MEASure:CLOCk:METHod:PLLTrack (see page 1875)	:ANALyze:CLOCk:METHod:PLLTrack (see page 348)	
:MEASure:CLOCk:METHod:SOURce (see page 1876)	:ANALyze:CLOCk:METHod:SOURce (see page 351)	
:MEASure:CLOCk:VERTical (see page 1877)	:ANALyze:CLOCk:VERTical (see page 352)	
:MEASure:CLOCk:VERTical:OFFSet (see page 1878)	:ANALyze:CLOCk:VERTical:OFFSet (see page 353)	
:MEASure:CLOCk:VERTical:RANGe (see page 1879)	:ANALyze:CLOCk:VERTical:RANGe (see page 354)	
:MEASure:DDPWS (see page 1880)	:MEASure:RJDJ:ALL? (see page 1087)	The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements.

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:FFT:PEAK1 (see page 1882)	:MEASure:FFT:DFRequency (see page 935) :MEASure:FFT:DMAGnitude (see page 937)	Peak numbers and threshold levels are now specified in the :MEASure:FFT:DFRequency and :MEASure:FFT:DMAGnitude command/query parameters.
:MEASure:FFT:PEAK2 (see page 1883)		
:MEASure:FFT:THReshold (see page 1884)		
:MEASure:JITTer:STATistics (see page 1885)	:ANALyze:AEDGes (see page 332)	The :ANALyze:AEDGes command maps to the "Measure All Edges" control in the user interface's Measurement Setup dialog box only. It does not affect jitter modes or statistics.
:MEASure:TIEData (see page 1886)	:MEASure:TIEData2 (see page 1176)	Clock recovery options have been removed (clock recovery as specified with the :ANALyze:CLOCk:METHod is used). When the signal type is PAM-4, an additional <threshold> parameter is used to specify the threshold at which to make the TIE measurements.
:MTESt:AVERage (see page 1854)		
:MTESt:AVERage:COUNt (see page 1855)		
:MTESt:FOLDing:COUNt? (see page 1856)	:MTESt:FOLDing:COUNt:UI? (see page 1253) :MTESt:FOLDing:COUNt:WAVeforms? (see page 1254)	The :MTESt:FOLDing:COUNt? query returns two values for UI count and waveform count. Now, there are separate queries that return those values individually. The UI count returned by :MTESt:FOLDing:COUNt? is now a floating-point value instead of an integer value.
:MTESt:STIMe (see page 1858)		
:MTESt<N>:ALIGn (see page 1859)		
:MTESt<N>:AUTO (see page 1860)		

Obsolete Command	Current Command Equivalent	Behavior Differences
:SPRocessing:CTLequalizer:ACGain (see page 1890)	:LANE1:EQUalizer:CTLE:ACGain (see page 761)	All :SPRocessing:CTLequalizer commands apply to Lane 1.
:SPRocessing:CTLequalizer:DCGain (see page 1891)	:LANE1:EQUalizer:CTLE:DCGain (see page 765)	
:SPRocessing:CTLequalizer:DISPlay (see page 1892)	:LANE1:EQUalizer:CTLE:STATe (see page 777)	The ":SPRocessing:CTLequalizer:DISPlay ON" command now: (1) turns CTLE on in Lane 1, (2) turns FFE off in Lane 1, and (3) turns on Lane 1.
:SPRocessing:CTLequalizer:FDISplay (see page 1893)	:LANE1:EQUalizer:LOCation (see page 815)	All :SPRocessing:CTLequalizer commands apply to Lane 1.
:SPRocessing:CTLequalizer:NUMPoles (see page 1894)	:LANE1:EQUalizer:CTLE:NUMPoles (see page 768)	
:SPRocessing:CTLequalizer:P1 (see page 1895)	:LANE1:EQUalizer:CTLE:P1 (see page 770)	
:SPRocessing:CTLequalizer:P2 (see page 1896)	:LANE1:EQUalizer:CTLE:P2 (see page 771)	
:SPRocessing:CTLequalizer:P3 (see page 1897)	:LANE1:EQUalizer:CTLE:P3 (see page 772)	
:SPRocessing:CTLequalizer:P4 (see page 1898)	:LANE1:EQUalizer:CTLE:P4 (see page 773)	
:SPRocessing:CTLequalizer:RATe (see page 1899)	:LANE1:EQUalizer:CTLE:RATE (see page 776)	
:SPRocessing:CTLequalizer:SOURce (see page 1900)	:LANE1:SOURce (see page 816)	Selected source applies to the entire lane.
:SPRocessing:CTLequalizer:VERTical (see page 1901)	:LANE1:VERTical (see page 818)	All :SPRocessing:CTLequalizer commands apply to Lane 1.
:SPRocessing:CTLequalizer:VERTical:OFFSet (see page 1902)	:LANE1:VERTical:OFFSet (see page 819)	
:SPRocessing:CTLequalizer:VERTical:RANGe (see page 1903)	:LANE1:VERTical:RANGe (see page 820)	
:SPRocessing:CTLequalizer:Z1 (see page 1904)	:LANE1:EQUalizer:CTLE:Z1 (see page 778)	
:SPRocessing:CTLequalizer:Z2 (see page 1905)	:LANE1:EQUalizer:CTLE:Z2 (see page 779)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:SPRocessing:CTLequalizer:ZERo (see page 1906)	▪ :SPRocessing:CTLequalizer:Z1 (see page 1904) ▪ :SPRocessing:CTLequalizer:Z2 (see page 1905)	Now that you can specify up to two zeros for a 3-pole CTLE, this command has been replaced by two new commands.
:SPRocessing:DFEQualizer:NTAPs (see page 1907)	:LANE2:EQUalizer:DFE:NTAPs (see page 780)	All :SPRocessing:DFEQualizer commands apply to Lane 2.
:SPRocessing:DFEQualizer:SOURce (see page 1908)	:LANE2:SOURce (see page 816)	Selected source applies to the entire lane.
:SPRocessing:DFEQualizer:STATe (see page 1909)	:LANE2:EQUalizer:DFE:STATe (see page 781)	The ":SPRocessing:DFEQualizer:STATe ON" command now: (1) turns on DFE in Lane 2, (2) set Lane 2's location to "in-place", and (3) turns on Lane 2.

Obsolete Command	Current Command Equivalent	Behavior Differences
:SPRocessing:DFEQualizer:TAP (see page 1910)	:LANE2:EQUalizer:DFE:TAP (see page 782)	All :SPRocessing:DFEQualizer commands apply to Lane 2.
:SPRocessing:DFEQualizer:TAP:AUTomatic (see page 1911)	:LANE2:EQUalizer:DFE:TAP:AUTomatic (see page 784)	
:SPRocessing:DFEQualizer:TAP:DELay (see page 1912)	:LANE2:EQUalizer:DFE:TAP:DELay (see page 785)	
:SPRocessing:DFEQualizer:TAP:DELay:AUTomatic (see page 1913)	:LANE2:EQUalizer:DFE:TAP:DELay:AUTomatic (see page 786)	
:SPRocessing:DFEQualizer:TAP:GAIN (see page 1914)	:LANE2:EQUalizer:DFE:TAP:GAIN (see page 787)	
:SPRocessing:DFEQualizer:TAP:LTARget (see page 1915)	:LANE2:EQUalizer:DFE:TAP:LTARget (see page 790)	
:SPRocessing:DFEQualizer:TAP:MAX (see page 1916)	:LANE2:EQUalizer:DFE:TAP:MAX (see page 791)	
:SPRocessing:DFEQualizer:TAP:MAXV (see page 1917)	:LANE2:EQUalizer:DFE:TAP:MAXV (see page 792)	
:SPRocessing:DFEQualizer:TAP:MIN (see page 1918)	:LANE2:EQUalizer:DFE:TAP:MIN (see page 793)	
:SPRocessing:DFEQualizer:TAP:MINV (see page 1919)	:LANE2:EQUalizer:DFE:TAP:MINV (see page 794)	
:SPRocessing:DFEQualizer:TAP:NORMalize (see page 1920)	:LANE2:EQUalizer:DFE:TAP:NORMalize (see page 795)	
:SPRocessing:DFEQualizer:TAP:UTARget (see page 1921)	:LANE2:EQUalizer:DFE:TAP:UTARget (see page 798)	
:SPRocessing:DFEQualizer:TAP:WIDTh (see page 1922)	:LANE2:EQUalizer:DFE:TAP:WIDTh (see page 799)	
:SPRocessing:EQUalizer:FDCouple (see page 1923)		
:SPRocessing:FFEQualizer:BANDwidth (see page 1924)	:LANE1:EQUalizer:FFE:BANDwidth (see page 803)	All :SPRocessing:FFEQualizer commands apply to Lane 1.
:SPRocessing:FFEQualizer:BWMode (see page 1925)	:LANE1:EQUalizer:FFE:BWMode (see page 804)	
:SPRocessing:FFEQualizer:DISPlay (see page 1926)	:LANE1:EQUalizer:FFE:STATe (see page 808)	The ":SPRocessing:FFEQualizer:DISPlay ON" command now: (1) turns FFE on in Lane 1, (2) turns CTLE off in Lane 1, and (3) turns on Lane 1.

Obsolete Command	Current Command Equivalent	Behavior Differences
:SPRocessing:FFEQualizer:FDISplay (see page 1927)	:LANE1:EQUalizer:LOCation (see page 815)	All :SPRocessing:FFEQualizer commands apply to Lane 1.
:SPRocessing:FFEQualizer:NPRecursor (see page 1928)	:LANE1:EQUalizer:FFE:NPRecursor (see page 805)	
:SPRocessing:FFEQualizer:NTAPs (see page 1929)	:LANE1:EQUalizer:FFE:NTAPs (see page 806)	
:SPRocessing:FFEQualizer:RATe (see page 1930)	:LANE1:EQUalizer:FFE:RATE (see page 807)	
:SPRocessing:FFEQualizer:SOURce (see page 1931)	:LANE1:SOURce (see page 816)	Selected source applies to the entire lane.
:SPRocessing:FFEQualizer:TAP (see page 1932)	:LANE1:EQUalizer:FFE:TAP (see page 809)	All :SPRocessing:FFEQualizer commands apply to Lane 1.
:SPRocessing:FFEQualizer:TAP:AUTomatic (see page 1933)	:LANE1:EQUalizer:FFE:TAP:AUTomatic (see page 810)	
:SPRocessing:FFEQualizer:TAP:DELay (see page 1934)	:LANE1:EQUalizer:FFE:TAP:DELay (see page 811)	
:SPRocessing:FFEQualizer:TAP:WIDTh (see page 1935)	:LANE1:EQUalizer:FFE:TAP:WIDTh (see page 812)	
:SPRocessing:FFEQualizer:TDELay (see page 1936)	:LANE1:EQUalizer:FFE:TDELay (see page 813)	
:SPRocessing:FFEQualizer:TDMode (see page 1937)	:LANE1:EQUalizer:FFE:TDMode (see page 814)	
:SPRocessing:FFEQualizer:VERTical (see page 1938)	:LANE1:VERTical (see page 818)	
:SPRocessing:FFEQualizer:VERTical:OFFSet (see page 1939)	:LANE1:VERTical:OFFSet (see page 819)	
:SPRocessing:FFEQualizer:VERTical:RANGe (see page 1940)	:LANE1:VERTical:RANGe (see page 820)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe (see page 1957)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:ARM:SLOPe (see page 1576)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:DELay:EDLY Commands" on page 1955.
:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce (see page 1958)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:ARM:SOURce (see page 1577)	
:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay (see page 1959)	:TRIGger:DELay:EDELay:COUNt (see page 1578)	
:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe (see page 1960)	:TRIGger:DELay:EDELay:SLOPe (see page 1579)	
:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce (see page 1961)	:TRIGger:DELay:EDELay:SOURce (see page 1580)	
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe (see page 1962)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:TRIGger:SLOPe (see page 1584)	
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce (see page 1963)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:TRIGger:SOURce (see page 1585)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe (see page 1966)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:ARM:SLOPe (see page 1576)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:DELay:TDLY Commands" on page 1964.
:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce (see page 1967)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:ARM:SOURce (see page 1577)	
:TRIGger:ADVanced:DELay:TDLY:DELay (see page 1968)	:TRIGger:DELay:TDELay:TIME (see page 1582)	
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe (see page 1969)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:TRIGger:SLOPe (see page 1584)	
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce (see page 1970)	:TRIGger:DELay:MODE (see page 1581) :TRIGger:DELay:TRIGger:SOURce (see page 1585)	
:TRIGger:ADVanced:PATTern:CONDition (see page 1946)	:TRIGger:PATTern:CONDition (see page 1613)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:PATTern Commands" on page 1944.
:TRIGger:ADVanced:PATTern:LOGic (see page 1947)	:TRIGger:PATTern:LOGic (see page 1614)	
:TRIGger:ADVanced:PATTern:THReshold:LEVel (see page 1948)	:TRIGger:LEVel (see page 1569)	
:TRIGger:ADVanced:STATe:CLOCk (see page 1950)	:TRIGger:STATe:CLOCk (see page 1645)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:STATe Commands" on page 1949.
:TRIGger:ADVanced:STATe:LOGic (see page 1951)	:TRIGger:STATe:LOGic (see page 1646)	
:TRIGger:ADVanced:STATe:LTYPe (see page 1952)	:TRIGger:STATe:LTYPe (see page 1647)	
:TRIGger:ADVanced:STATe:SLOPe (see page 1953)	:TRIGger:STATe:SLOPe (see page 1648)	
:TRIGger:ADVanced:STATe:THReshold:LEVel (see page 1954)	:TRIGger:LEVel (see page 1569)	
:TRIGger:ADVanced:VIOLation:MODE (see page 1972)	:TRIGger:MODE (see page 1572)	There are minimal differences in behavior. See also "Obsolete Advanced Violation Trigger Modes" on page 1971.

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:ADVanced:VIOLation:PWIDth:DIRection (see page 1975)	:TRIGger:PWIDth:MODE (see page 1616)	There are minimal differences in behavior. See also ":Obsolete :TRIGger:ADVanced:VIOLation:PWIDth Commands" on page 1973.
:TRIGger:ADVanced:VIOLation:PWIDth:POLarity (see page 1976)	:TRIGger:PWIDth:POLarity (see page 1617)	
:TRIGger:ADVanced:VIOLation:PWIDth:SOURce (see page 1977)	:TRIGger:PWIDth:SOURce (see page 1619)	
:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh (see page 1978)	:TRIGger:PWIDth:WIDTh (see page 1621)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce (see page 1982)	:TRIGger:SHOLd:CSOurce (see page 1638)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:VIOLation:SETup Commands" on page 1979.
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE (see page 1983)	:TRIGger:SHOLd:CSOurce:EDGE (see page 1639)	
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel (see page 1984)	:TRIGger:LEVel (see page 1569)	
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce (see page 1985)	:TRIGger:SHOLd:DSOurce (see page 1640)	
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold (see page 1986)	:TRIGger:HTHReshold (see page 1567)	
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold (see page 1987)	:TRIGger:LTHReshold (see page 1571)	
:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME (see page 1988)	:TRIGger:SHOLd:HoldTIMe (see page 1641)	
:TRIGger:ADVanced:VIOLation:SETup:MODE (see page 1989)	:TRIGger:SHOLd:MODE (see page 1642)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce (see page 1990)	:TRIGger:SHOLd:CSOurce (see page 1638)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE (see page 1991)	:TRIGger:SHOLd:CSOurce:EDGE (see page 1639)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel (see page 1992)	:TRIGger:LEVel (see page 1569)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce (see page 1993)	:TRIGger:SHOLd:DSOurce (see page 1640)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold (see page 1994)	:TRIGger:HTHReshold (see page 1567)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold (see page 1995)	:TRIGger:LTHReshold (see page 1571)	
:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME (see page 1996)	:TRIGger:SHOLd:SetupTIMe (see page 1643)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:ADVanced:VIOLation:TRANsition (see page 2006)	:TRIGger:TRANsition:MODE (see page 1654) :TRIGger:TRANsition:TIME (see page 1657)	There are minimal differences in behavior. See also "Obsolete :TRIGger:ADVanced:VIOLation:TRANsition Commands" on page 2005.
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce (see page 2007)	:TRIGger:TRANsition:SOURce (see page 1656)	
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold (see page 2008)	:TRIGger:HTHReshold (see page 1567)	
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold (see page 2009)	:TRIGger:LTHReshold (see page 1571)	
:TRIGger:ADVanced:VIOLation:TRANsition:TYPE (see page 2010)	:TRIGger:TRANsition:TYPE (see page 1658)	
:TRIGger:PWIDth:DIRection (see page 1942)	:TRIGger:PWIDth:MODE (see page 1616)	In addition to the "greater than" or "less than" modes, the :TRIGger:PWIDth:MODE command adds "inside range" and "outside range" modes.
:TRIGger:TRANsition:DIRection (see page 1943)	:TRIGger:TRANsition:MODE (see page 1654)	In addition to the "greater than" or "less than" modes, the :TRIGger:TRANsition:MODE command adds "inside range" and "outside range" modes.

Advanced Trigger Mode and Commands are Obsolete (Deprecated)

To place the instrument in the advanced triggering mode you select:

```
:TRIGger:MODE ADVanced
```

The advanced triggering mode allows backward compatibility access to the DELay, PATTern, STATe, and VIOLation modes. When this mode is selected, use the :TRIGger:ADVanced:MODE command to select the advanced trigger mode.

```
:TRIGger:ADVanced:MODE <advanced_trigger_mode>
```

Table 21 :TRIGger:ADVanced:MODE Settings

Mode	Definition
DELay	Delay by Events mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. Delay by Time mode lets you view pulses in your waveform that occur a long time after a specified waveform edge.
PATTern	Pattern triggering lets you trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.
STATe	State triggering lets you set the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.
VIOLation	Trigger violation modes: Pulse WIDth, SETup, TRANsition. When this mode is selected, use the :TRIGger:ADVanced:VIOLation:MODE command to select the advanced trigger violation mode.

Each mode is described with its command set in this chapter.

Discontinued Commands

Discontinued commands are commands that were supported in previous versions of the Infiniium oscilloscope software, but are not supported by this version of the Infiniium oscilloscope software. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
:DISK:STORe	":DISK:SAVE:SETup" on page 545 ":DISK:SAVE:WAVeform" on page 546	For saving setups and waveforms to disk.
:DISPlay:GRATicule:SIZE	None	Graticule sizing is different in the version 5.00 next-generation Infiniium user interface software.

Obsolete Acquire Commands

:ACQuire:HRESolution

Command

NOTE This command is for backwards compatibility; however, it does not provide access to all the High Resolution Oscilloscope (HRO) settings. Use the ":ACQuire:ADCRes" on page 292 command instead.

```
:ACQuire:HRESolution {BITS10 | {BITS11 | BITF11} | {BITS12 | BITF12}
    | BITF13 | BITF14 | BITF15 | BITF16}
```

In earlier Infiniium oscilloscopes, the High Resolution acquisition mode uses uncorrected data (that is, no response or interleave correction) and a boxcar average is performed to produce high-resolution output of each acquisition.

In the MXR/EXR-Series oscilloscopes, the analog-to-digital converter (ADC) can perform both response and interleave correction on the ADC input stream while bandwidth-limiting the data to produce corrected, high-resolution data with an effective bits performance that is superior to the earlier high-resolution method.

The starting sample rate for High Resolution Oscilloscope (HRO) is 6.4 GSa/s and goes down to 50 MSa/s as shown the following table.

ADC Resolution	HRO GSa/s	Bandwidth	Effective Bits
BITF11	6.4	2.56 GHz	10.7
BITF12	3.2	1.28 GHz	11.2
BITF13	1.6	640 MHz	11.7
BITF14	0.800	320 MHz	12.2
BITF15	0.400	160 MHz	12.7
BITF16 (80 MHz)	0.200	80 MHz	13.2
16 Bits (40 MHz)*	0.100	40 MHz	13.7
16 Bits (20 MHz)*	0.050	20 MHz	14.2
* These settings can be made with the ":ACQuire:ADCRes" on page 292 command.			

NOTE Selecting an ADC Resolution of 11 Bits or greater locks in the ADC sample rate and bandwidth according to the above table.

If the time/div and memory depth settings are such that there are more samples than can fit into memory, simple decimation (or peak detect in that acquisition mode) is used, not additional high-resolution decimation (where samples in a time bin are averaged).

In earlier Infiniium oscilloscopes, the High Resolution and Peak Detect acquisition modes were mutually exclusive. In the MXR/EXR-Series oscilloscopes, you can still select between the **Normal** and **Peak Detect** acquisition modes when high-resolution options (between 11 and 16 bits) are selected.

For more information on the ADC Resolution settings, see the MXR/EXR-Series oscilloscopes online help.

Example This example sets the bit resolution setting to a minimum of 14 bits.

```
myScope.WriteString ":ACQuire:HRESolution BITF14"
```

Query

```
:ACQuire:HRESolution?
```

The :ACQuire:HRESolution? query returns the bit resolution setting.

Returned Format

```
[:ACQuire:HRESolution] {BITS10 | BITF11 | BITF12 | BITF13 | BITF14
    | BITF15 | BITF16}<NL>
```

Example This example places the current bit resolution setting in the string variable, strBitRes, then prints the contents of the variable to the computer's screen.

```
Dim strBitRes As String   ' Dimension variable.
myScope.WriteString ":ACQuire:HRESolution?"
strBitRes = myScope.ReadString
Debug.Print strBitRes
```

See Also

- ":ACQuire:MODE" on page 308
- ":ACQuire:SRATe[:ANALog] — Analog Sample Rate" on page 323
- ":TIMebase:SCALe" on page 1549
- ":TIMebase:RANGe" on page 1543

History Legacy command (existed before version 3.10).

Version 10.00: Because the Keysight UXR Series oscilloscopes have a 10-bit analog-to-digital converter (ADC), the BITS10 and BITS9 parameters are no longer applicable.

Version 11.00: The MXR/EXR-Series oscilloscopes provide High Resolution Oscilloscope (HRO) selections that work differently than the High Resolution acquisition mode in earlier Infiniium oscilloscopes.

Obsolete Analyze Commands

:ANALyze:CLOCk:METHod:PAM:B03

Command

```
:ANALyze:CLOCk:METHod:PAM:B03 {{0 | OFF} | {1 | ON}}
```

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the :ANALyze:CLOCk:METHod:PAM:B03 command specifies whether edges from the 0 level to the 3 level and from the 3 level to the 0 level are are included in the clock recovery.

Remember, with PAM-4 signals, clock recovery is performed individually for each signal source; therefore, this setting applies to the source specified with the :ANALyze:CLOCk:METHod:SOURce command.

Query

```
:ANALyze:CLOCk:METHod:PAM:B03?
```

The :ANALyze:CLOCk:METHod:PAM:B03? query returns whether 03,30 edges are included in the clock recovery.

Returned Format

```
[:ANALyze:CLOCk:METHod:PAM:B03] <setting><NL>
```

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

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod:PAM:B12" on page 1840
- ":ANALyze:CLOCk:METHod:PAM:NONSymmetric" on page 1842
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

:ANALyze:CLOCk:METHod:PAM:B12

Command

```
:ANALyze:CLOCk:METHod:PAM:B12 {{0 | OFF} | {1 | ON}}
```

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the :ANALyze:CLOCk:METHod:PAM:B12 command specifies whether edges from the 1 level to the 2 level and from the 2 level to the 1 level are are included in the clock recovery.

Remember, with PAM-4 signals, clock recovery is performed individually for each signal source; therefore, this setting applies to the source specified with the :ANALyze:CLOCk:METHod:SOURce command.

Query

```
:ANALyze:CLOCk:METHod:PAM:B12?
```

The :ANALyze:CLOCk:METHod:PAM:B12? query returns whether 12,21 edges are included in the clock recovery.

Returned Format

```
[:ANALyze:CLOCk:METHod:PAM:B12] <setting><NL>
```

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

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod:PAM:B03" on page 1838
- ":ANALyze:CLOCk:METHod:PAM:NONSymmetric" on page 1842
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154
- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

:ANALyze:CLOCk:METHod:PAM:NONSymmetric

Command

```
:ANALyze:CLOCk:METHod:PAM:NONSymmetric {{0 | OFF} | {1 | ON}}
```

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), the :ANALyze:CLOCk:METHod:PAM:NONSymmetric command specifies whether edges that are non-symmetric about the middle threshold (for example, from the 1 level to the 3 level or from the 2 level to the 0 level) are included in the clock recovery.

Remember, with PAM-4 signals, clock recovery is performed individually for each signal source; therefore, this setting applies to the source specified with the :ANALyze:CLOCk:METHod:SOURce command.

Query

```
:ANALyze:CLOCk:METHod:PAM:NONSymmetric?
```

The :ANALyze:CLOCk:METHod:PAM:NONSymmetric? query returns whether non-symmetric edges are included in the clock recovery.

Returned Format

```
[:ANALyze:CLOCk:METHod:PAM:NONSymmetric] <setting><NL>
```

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

See Also

- ":ANALyze:CLOCk:METHod:SOURce" on page 351
- ":ANALyze:CLOCk:METHod:PAM:B03" on page 1838
- ":ANALyze:CLOCk:METHod:PAM:B12" on page 1840
- ":ANALyze:SIGNal:DATarate" on page 387
- ":ANALyze:SIGNal:SYMBolrate" on page 402
- ":ANALyze:SIGNal:TYPE" on page 404
- ":MEASure:CGRade:EWIDth" on page 892
- ":MEASure:CGRade:EHEight" on page 889
- ":MEASure:FALLtime" on page 932
- ":MEASure:PAM:ELEVel" on page 1014
- ":MEASure:PAM:ESKew" on page 1016
- ":MEASure:PAM:LEVel" on page 1027
- ":MEASure:PAM:LRMS" on page 1029
- ":MEASure:PAM:LTHickness" on page 1031
- ":MEASure:RISetime" on page 1085
- ":MEASure:THResholds:GENeral:METHod" on page 1135
- ":MEASure:THResholds:GENeral:PAMCustom" on page 1137
- ":MEASure:THResholds:GENeral:PAMAutomatic" on page 1139
- ":MEASure:THResholds:RFALl:METHod" on page 1152
- ":MEASure:THResholds:RFALl:PAMAutomatic" on page 1154

- ":MEASure:TIEData2" on page 1176

History New in version 5.50.

Obsolete Display Commands

:DISPlay:COLumn

Command

NOTE

This command is deprecated. It is accepted but ignored. Bookmarks are now the method used to place text strings or annotations on screen. The closest command equivalent is ":DISPlay:BOOKmark<N>:XPOSition" on page 555.

```
:DISPlay:COLumn <column_number>
```

The :DISPlay:COLumn command specifies the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands.

<column _number> An integer representing the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 0 to 90.

Example This example sets the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands to column 10.

```
myScope.WriteString ":DISPlay:COLUMN 10"
```

Query

```
:DISPlay:COLumn?
```

The :DISPlay:COLumn? query returns the column where the next :DISPlay:LINE or :DISPlay:STRing starts.

Returned Format

```
[:DISPlay:COLumn] <value><NL>
```

Example This example returns the current column setting to the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:COLUMN?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

:DISPlay:LINE

Command

NOTE This command is deprecated. It is accepted but ignored. Bookmarks are now the method used to place text strings or annotations on screen. The closest command equivalent is ":DISPlay:BOOKmark<N>:SET" on page 553.

```
:DISPlay:LINE "<string_argument>"
```

The :DISPlay:LINE command writes a quoted string to the screen, starting at the location specified by the :DISPlay:ROW and :DISPlay:COLumn commands.

<string _argument> Any series of ASCII characters enclosed in quotation marks.

Example This example writes the message "Infiniium Test" to the screen, starting at the current row and column location.

```
myScope.WriteString ":DISPlay:LINE ""Infiniium Test"""
```

When using the C programming language, quotation marks within a string are escaped using the backslash (\) character as shown in the next example. This example writes the message "Infiniium Test" to the screen.

```
printf("\"Infiniium Test\"");
```

You may write text up to column 94. If the characters in the string do not fill the line, the rest of the line is blanked. If the string is longer than the space available on the current line, the excess characters are discarded.

In any case, the ROW is incremented and the COLumn remains the same. The next :DISPlay:LINE command will write on the next line of the display. After writing the last line in the display area, the ROW is reset to 0.

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

:DISPlay:ROW

Command

NOTE This command is deprecated. It is accepted but ignored. Bookmarks are now the method used to place text strings or annotations on screen. The closest command equivalent is ":DISPlay:BOOKmark<N>:YPOSition" on page 556.

```
:DISPlay:ROW <row_number>
```

The :DISPlay:ROW command specifies the starting row on the screen for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The row number remains constant until another :DISPlay:ROW command is received, or the row is incremented by the :DISPlay:LINE command.

<row_number> An integer representing the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 9 to 23.

Example This example sets the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands to 10.

```
myScope.WriteString ":DISPlay:ROW 10"
```

Query

```
:DISPlay:ROW?
```

The :DISPlay:ROW? query returns the current value of the row.

Returned Format

```
[:DISPlay:ROW] <row_number><NL>
```

Example This example places the current value for row in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:ROW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

:DISPlay:STRing

Command

NOTE

This command is deprecated. It is accepted but ignored. Bookmarks are now the method used to place text strings or annotations on screen. The closest command equivalent is ":DISPlay:BOOKmark<N>:SET" on page 553.

```
:DISPlay:STRing "<string_argument>"
```

The :DISPlay:STRing command writes text to the oscilloscope screen. The text is written starting at the current row and column settings. If the column limit is reached, the excess text is discarded. The :DISPlay:STRing command does not increment the row value, but :DISPlay:LINE does.

<string _argument> Any series of ASCII characters enclosed in quotation marks.

Example This example writes the message "Example 1" to the oscilloscope's display starting at the current row and column settings.

```
myScope.WriteString ":DISPlay:STRING ""Example 1"""
```

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

:DISPlay:TAB

Command

```
:DISPlay:TAB <tab>
```

The :DISPlay:TAB command displays the corresponding tab indicated by the <tab> parameter.

<tab> MEASurement | MARKer | DIGital | LIMittest | JITTer | NOISe | HISTogram | MASKtest | EYE | COLorgrade | NAVigation | STATus | SCALe | BOOKmark | CROSstalk | FAILuretable | FFTPeaks | EQSettings

Example This example sets the Status tab as the displayed one.

```
myScope.WriteString ":DISPlay:TAB STATus"
```

Query

```
:DISPlay:TAB?
```

The :DISPlay:TAB? query returns the tab that is currently displayed.

Returned Format

```
[:DISPlay:TAB] {MEAS | MARK | DIG | LIM | JITT | NOIS | HIST | MASK
    | EYE | COL | NAV | STAT | SCAL | BOOK | CROS | FAIL | FFTP
    | EQS}<NL>
```

Example This example places the currently displayed tab into the string variable, strTab, then prints the contents of the variable to the computer's screen.

```
Dim strTab As String   ' Dimension variable.
myScope.WriteString ":DISPlay:TAB?"
strTab = myScope.ReadString
Debug.Print strTab
```

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

:DISPlay:TEXT

Command

NOTE This command is deprecated. It is accepted but ignored. Bookmarks are now the method used to place text strings or annotations on screen. The closest command equivalent is ":DISPlay:BOOKmark<N>:DELete" on page 552.

```
:DISPlay:TEXT BLANk
```

The :DISPlay:TEXT command blanks the user text area of the screen.

Example This example blanks the user text area of the oscilloscope's screen.

```
myScope.WriteString ":DISPlay:TEXT BLANK"
```

History Legacy command (existed before version 3.10).

Obsoleted in version 5.00.

Obsolete Hosted Commands

:HOSTed:CALibrate:ALIGn (MultiScope)

Command

```
:HOSTed:CALibrate:ALIGn {{0 | OFF} | {1 | ON}}
```

NOTE As of software version 5.60, acquired data is always aligned, and this command has no effect.

The :HOSTed:CALibrate:ALIGn command lets you align acquired data in the MultiScope system. When ON, a unique time-shifting FIR filter is applied to each output waveform and the waveforms are truncated so that all of them have the same X origin and number of points values.

Setting ":HOSTed:CALibrate:ALIGn ON" causes longer acquisition times.

Query

```
:HOSTed:CALibrate:ALIGn?
```

The :HOSTed:CALibrate:ALIGn? query returns the align acquired data setting.

Returned Format

```
[:HOSTed:CALibrate:ALIGn] <setting><NL>
```

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

See Also

- ":HOSTed:CALibrate:CALibrate" on page 709
- ":HOSTed:CALibrate:CHANnel" on page 710
- ":HOSTed:CALibrate:DESKew:FRAMes" on page 712
- ":HOSTed:CALibrate:DESKew:CHANnels" on page 711
- ":HOSTed:CALibrate:DESKew:SIGNals" on page 713
- ":HOSTed:CALibrate:DESKew:ZERO" on page 714
- ":HOSTed:CALibrate:LEVel" on page 715
- ":HOSTed:CALibrate:PROMpt" on page 717
- ":HOSTed:CALibrate:STATus:CHANnels?" on page 718
- ":HOSTed:CALibrate:STATus:FRAMes?" on page 719
- ":HOSTed:CALibrate:STATus:LEVel?" on page 720
- ":HOSTed:CALibrate:STATus:SIGNals?" on page 721
- ":HOSTed:CALibrate:TREF:DETect" on page 722

History

New in version 5.50.

Version 5.60: Acquired data is now always aligned, and this command has no effect.

Obsolete Mask Test Commands

:MTESt:AVERage

Command

```
:MTESt:AVERage {{ON | 1} | {OFF | 0}}
```

The :MTESt:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :MTESt:AVERage:COUNt command described next.

The :ACQuire:AVERage command performs the same function as this command.

Averaging is not available in PDETect mode.

Example

This example turns averaging on.

```
myScope.WriteString ":MTESt:AVERage ON"
```

Query

```
:MTESt:AVERage?
```

The :MTESt:AVERage? query returns the current setting for averaging.

Returned Format

```
[:MTESt:AVERage] {1 | 0} <NL>
```

Example

This example places the current settings for averaging into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":MTESt:AVERage?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History

Legacy command (existed before version 3.10).

:MTESt:AVERage:COUNt

Command

```
:MTESt:AVERage:COUNt <count_value>
```

The :MTESt:AVERage:COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :MTESt:AVERage:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :ACQuire:AVERage:COUNt command performs the same function as this command.

<count_value> An integer, 2 to 65534, specifying the number of data values to be averaged.

Example This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :MTESt:COMPlete command.

```
myScope.WriteString ":MTESt:AVERage:COUNt 16"
```

Query

```
:MTESt:AVERage:COUNt?
```

The :MTESt:AVERage:COUNt? query returns the currently selected count value.

Returned Format

```
[:MTESt:AVERage:COUNt] <value><NL>
```

<value> An integer, 2 to 65534, specifying the number of data values to be averaged.

Example This example checks the currently selected count value and places that value in the string variable, varResult. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTESt:AVERage:COUNt?"
varResult = myScope.ReadNumber
Debug.Print FormatNumber(varResult, 0)
```

History Legacy command (existed before version 3.10).

:MTESt:FOLDing:COUNt?

Query

```
:MTESt:FOLDing:COUNt? [<source>]
```

The :MTESt:FOLDing:COUNt? query returns the number of waveforms and unit intervals in the real time eye.

<source> {CHANnel<N> | DIFF<D> | COMMonmode<C> | FUNCtion<F> | WMEMory<R> | EQUalized<L>}

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

If the <source> is not specified, the :MTESt:FOLDing:COUNt? query returns the results of the first real-time eye that is on. Sources are ordered by channels, memories, and then functions.

<F> An integer, 1-16.

<R> An integer, 1-4.

<L> An integer, 1-4.

Returned Format

```
[:MTESt:FOLDing:COUNt] Real Time Eye<NL>
<N> UI<NL>
<N> Wfm<NL>
```

The UI count returned is a floating-point value. The Wfm count returned is an integer.

Example

```
myScope.WriteString ":MTESt:FOLDing:COUNt? CHANnel1"
strRteCount = myScope.ReadString
Debug.Print strRteCount
```

See Also

- ":MTESt:FOLDing (Clock Recovery software only)" on page 1249
- ":MTESt:FOLDing:BITS" on page 1251
- ":MTESt:FOLDing:POSition" on page 1255
- ":MTESt:FOLDing:TPOSition" on page 1257
- ":MTESt:FOLDing:SCALe" on page 1256
- ":MTESt:FOLDing:TSCale" on page 1258

History New in version 5.00.

Version 5.50: The UI count returned is now a floating-point value instead of an integer value. This command is deprecated, replaced by ":MTESt:FOLDing:COUNt:UI?" on page 1253 and ":MTESt:FOLDing:COUNt:WAVeforms?" on page 1254.

Version 5.52: The <source> parameter is now optional.

:MTESt:STIMe

Command

```
:MTESt:STIMe <timeout>
```

The :MTESt:STIMe command sets the timeout value for the Autoalign feature. If the oscilloscope is unable to align the mask to your waveform within the specified timeout value, it will stop trying to align and will report an alignment failure.

<timeout> An integer from 1 to 120 seconds representing the time between triggers (not the time that it takes to finish the alignment.)

Example This example sets the timeout value for the Autoalign feature to 10 seconds.

```
myScope.WriteString ":MTESt:STIMe 10"
```

Query

```
:MTESt:STIMe?
```

The query returns timeout value for the Autoalign feature.

Returned Format

```
[:MTESt:STIMe] <timeout><NL>
```

Example This example gets the timeout setting and prints the result on the computer display.

```
myScope.WriteString ":MTESt:STIMe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:MTESt<N>:ALIGn

Command

```
:MTESt<N>:ALIGn
```

The :MTESt<N>:ALIGn command automatically aligns and scales the mask to the current waveform on the display. The type of mask alignment performed depends on the current setting of the Use File Setup When Aligning control. See the :MTESt<N>:AUTO command for more information.

<N> An integer, 1-8.

Example This example aligns the current mask to the current waveform.

```
myScope.WriteString ":MTESt1:ALIGn"
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

:MTESt<N>:AUTO

Command

```
:MTESt<N>:AUTO {{ON | 1} | {OFF | 0}}
```

The :MTESt<N>:AUTO command enables (ON) or disables (OFF) the Use File Setup When Aligning control. This determines which type of mask alignment is performed when the :MTESt<N>:ALIGn command is sent. When enabled, the oscilloscope controls are changed to the values which are determined by the loaded mask file. This alignment guarantees that the aligned mask and any subsequent mask tests meet the requirements of the standard.

When disabled, the alignment is performed using the current oscilloscope settings. This may be useful when troubleshooting problems during the design phase of a project.

<N> An integer, 1-8.

Example This example enables the Use File Settings When Aligning control.

```
myScope.WriteString ":MTESt1:AUTO ON"
```

Query

```
:MTESt<N>:AUTO?
```

The :MTESt<N>:AUTO? query returns the current value of the Use File Setup When Aligning control.

Returned Format

```
[:MTESt<N>:AUTO] {1 | 0} <NL>
```

Example

```
myScope.WriteString ":MTESt1:AUTO?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 11.00: Now applies to one of 8 masks. MTESt is equivalent to MTESt1.

Obsolete Measure Commands

:MEASure:CLOCk

Command

```
:MEASure:CLOCk {{{ON|1},CHANnel<N>} | {OFF|0}}
```

The :MEASure:CLOCk command turns the recovered clock display on or off and sets the clock recovery channel source.

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

Example This example turns the recovered clock display on for channel 1.

```
myScope.WriteString ":MEASure:CLOCk ON,CHANnel1"
```

Query

```
:MEASure:CLOCk?
```

The :MEASure :CLOCk? query returns the state of the recovered clock display.

Returned Format

```
[:MEASure:CLOCk] {1 | 0}<NL>
```

Example This example places the current setting of the recovered clock display in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk" on page 333.

:MEASure:CLOCk:METHod

Command

```
:MEASure:CLOCk:METHod
    {FC,{FC1063 | FC2125 | FC425}}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH}[,<multiplier>]}
  | {FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
```

The :MEASure:CLOCk:METHod command sets the clock recovery method to:

- FC (Fibre Channel).
- EXPlicit (Explicit Clock).
- FIXed (Constant Frequency).
- FLEXR (FlexRay Receiver).
- FLEXT (FlexRay Transmitter).

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":MEASure:CLOCk:METHod:OJTF" on page 1873.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":MEASure:CLOCk:METHod:JTF" on page 1871.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<R> An integer, 1-4.

<data_rate> A real number for the base data rate in Hertz.

<multiplier> An integer used as the multiplication factor.

<baud_rate> A real number used for the baud rate.

Example This example sets the explicit clock recovery method on channel 1, rising edge, with a multiplier of 2.

```
myScope.WriteString ":MEASure:CLOCk:METHod EXPlicit,CHANnel1,RISing,2"
```

Query

```
:MEASure:CLOCk:METHod?
```

The :MEASure:CLOCk:METHod? query returns the state of the clock recovery method.

NOTE

You can use the :MEASure:CLOCk:METHod? query when phase-locked loop (PLL) clock recovery methods are set up. The format returned will be that of the :MEASure:CLOCk:METHod:OJTF? query. See ":MEASure:CLOCk:METHod:OJTF" on page 1873.

Returned Format

```
[:MEASure:CLOCk:METHod]
    {FC,{FC1063 | FC2125 | FC425}}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH},<multiplier>}
  | {FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
```

Example

This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History

Legacy command (existed before version 3.10).

Version 4.20: The command options for specifying clock recovery PLL options moved to the new commands :MEASure:CLOCk:METHod:JTF and :MEASure:CLOCk:METHod:OJTF.

Version 5.10: The PCIE clock recovery method has been removed.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod" on page 334.

:MEASure:CLOCk:METHod (deprecated)

Command

NOTE Some of these command options have been deprecated – options for specifying clock recovery PLL options have been moved to the new commands ":MEASure:CLOCk:METHod:JTF" on page 1871 and ":MEASure:CLOCk:METHod:OJTF" on page 1873. See also ":MEASure:CLOCk:METHod" on page 1863.

```
:MEASure:CLOCk:METHod {FOPLL,<data_rate>,<loop_bandwidth>}
    | {EQFOPLL,<data_rate>,<loop_bandwidth>}
    | {SOPLL,<data_rate>,<loop_bandwidth>, <damping_factor>}
    | {EQSOPLL,<data_rate>,<loop_bandwidth>, <damping_factor>}
    | {FC,{FC1063 | FC2125 | FC425}}
    | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
        <multiplier>,<clock_freq>,<loop_bandwidth>}
    | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
        <multiplier>,<clock_freq>,<loop_bandwidth>,<damping_fact>}
    | {EXPlicit,<source>,{RISing | FALLing | BOTH}[,<multiplier>]}
    | {FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}
    | {FLEXR,<baud_rate>}
    | {FLEXT,<baud_rate>}
```

The :MEASure:CLOCk:METHod command sets the clock recovery method to:

- FOPLL (first order phase-locked loop).
- SOPLL (second order phase-locked loop).
- EQFOPLL (equalized first order phase-locked loop).
- EQSOPLL (equalized second order phase-locked loop).
- FC (Fibre Channel).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).
- EXPlicit (Explicit Clock).
- FIXed (Constant Frequency).
- FLEXR (FlexRay Receiver).
- FLEXT (FlexRay Transmitter).

The equalized clock recovery methods are available when the Advanced Signal Integrity Software license is installed.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<R>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

FUNCtion<N> and WMEMory<R> are:

An integer, 1-4, representing the selected function or waveform memory.

<data_rate> A real number for the base data rate in Hertz.

<damping_ factor> A real number for the damping factor of the PLL in bits per second.

<loop_ bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

<track_freq> A real number used for the tracking frequency of the PLL.

<damping_fact> A real number used for the damping factor of the PLL.

<baud_rate> A real number used for the baud rate.

Example This example sets the clock recovery method to phase-locked loop.

```
myScope.WriteString ":MEASure:CLOCK:METHOD FOPLL,2E9,1.19E6"
```

Query

```
:MEASure:CLOCk:METHod?
```

The :MEASure:CLOCk:METHod? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod] {FOPLL,<data_rate>,<loop_bandwidth>}
    | {EQFOPLL,<data_rate>,<loop_bandwidth>}
    | {SOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>}
    | {EQSOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>}
    | {FC,{FC1063 | FC2125 | FC425}}
    | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
        <multiplier>,<clock_freq>,<track_freq>}
    | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
        <multiplier>,<clock_freq>,<track_freq>,<damping_fact>}
    | {EXPlicit,<source>,{RISing | FALLing | BOTH},<multiplier>}
    | {FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
    | {FLEXR,<baud_rate>}
    | {FLEXT,<baud_rate>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK:METHOD?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 5.10: The PCIE clock recovery method has been removed.

:MEASure:CLOCk:METHod:ALIGn

Command

```
:MEASure:CLOCK:METHod:ALIGn {CENTer | EDGE}
```

When using an explicit method of clock recovery, the :MEASure:CLOCk:METHod:ALIGn command specifies how the clock is aligned with data:

- CENTer — Clock edges are aligned with the center of data.
- EDGE — Clock edges are aligned with data edges. In this case, Time Interval Error (TIE) is measured directly from the data edge to the clock edge.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

Example

When using an explicit method of clock recovery, this example specifies that clock edges are aligned with the center of data.

```
myScope.WriteString ":MEASure:CLOCK:METHOD:ALIGn CENTer"
```

Query

```
:MEASure:CLOCK:METHod:ALIGn?
```

The :MEASure:CLOCk:METHod:ALIGn? query returns the clock recovery method's edge alignment setting.

Returned Format

```
[:MEASure:CLOCk:METHod:ALIGn] {CENT | EDGE}
```

Example

This example places the current edge alignment setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK:METHOD:ALIGn?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History

New in version 3.20.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:ALIGn" on page 338.

:MEASure:CLOCk:METHod:DEEMphasis

Command

```
:MEASure:CLOCk:METHod:DEEMphasis {OFF | ON}
```

The :MEASure:CLOCk:METHod:DEEMphasis command turns de-emphasis on or off.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

See the help system for more information on de-emphasis.

Example

This example enables de-emphasis.

```
myScope.WriteString ":MEASure:CLOCk:METHod:DEEMphasis ON"
```

Query

```
:MEASure:CLOCk:METHod:DEEMphasis?
```

The :MEASure:CLOCk:METHod:DEEMphasis? query returns whether or not de-emphasis is turned on.

Returned Format

```
[:MEASure:CLOCk:METHod:DEEMphasis] {OFF | ON}
```

Example

This example places the current setting of the de-emphasis mode in the string variable strDeemph, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK:METHod:DEEMphasis?"
strDeemph = myScope.ReadString
Debug.Print strDeemph
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History

Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:DEEMphasis" on page 339.

:MEASure:CLOCk:METHod:EDGE

Command

```
:MEASure:CLOCk:METHod:EDGE {RISing | FALLing | BOTH}
```

The :MEASure:CLOCk:METHod:EDGE command specifies which edge(s) of the data are used to recover a clock. (In the front panel GUI, this control appears in the Advanced Clock Recovery dialog box.) Normally, both edges are used. However, if you are performing clock recovery on a low duty cycle clock signal, for example, you may want to use just the rising or falling edge.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

This command applies to the following clock recovery methods:

- FIXed (Constant Frequency).
- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EXPlicit (Explicit Clock).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).

To measure jitter on only rising (or falling) edges of a clock, you must also set :MEASure:RJDJ:EDGE to the same RISing or FALLing option, and you must set :MEASure:RJDJ:CLOCk ON to force the pattern to be a clock and set the jitter for edges not examined to zero (0).

Example This example specifies that both rising and falling edges of the data are used to recover a clock.

```
myScope.WriteString ":MEASure:CLOCk:METHod:EDGE BOTH"
```

Query

```
:MEASure:CLOCK:METHod:EDGE?
```

The :MEASure:CLOCk:METHod:EDGE? query returns the clock recovery method's edge setting.

Returned Format

```
[:MEASure:CLOCk:METHod:EDGE] {RIS | FALL | BOTH}
```

Example This example places the current edge setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:EDGE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876

- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:RJDJ:EDGE" on page 1097
- ":MEASure:RJDJ:CLOCk" on page 1095

History New in version 4.30.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:EDGE" on page 340.

:MEASure:CLOCk:METHod:JTF

Command

```
:MEASure:CLOCk:METHod:JTF
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
```

The :MEASure:CLOCk:METHod:JTF command specifies the clock recovery PLL's response in terms of the Jitter Transfer Function's (JTF) 3 dB bandwidth.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).

The equalized clock recovery methods are available when the Advanced Signal Integrity Software license is installed.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":MEASure:CLOCk:METHod:OJTF" on page 1873.

For setting other clock recovery methods, see ":MEASure:CLOCk:METHod" on page 1863.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<R> An integer, 1-4.

<data_rate> A real number for the base data rate in bits per second.

<peaking> The peaking value in dB.

<jtf_loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a peaking value of 1.25 dB.

```
myScope.WriteString ":MEASure:CLOCk:METHod:JTF SOPLL,4E9,3.822E6,1.25"
```

Query

```
:MEASure:CLOCk:METHod:JTF?
```

The :MEASure:CLOCk:METHod:JTF? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod:JTF]
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:JTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History New in version 4.20.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:JTF" on page 343.

:MEASure:CLOCk:METHod:OJTF

Command

```
:MEASure:CLOCk:METHod:OJTF
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_factor>}
```

The :MEASure:CLOCk:METHod:OJTF command specifies the clock recovery PLL's response in terms of the Observed Jitter Transfer Function's (OJTF) 3 dB bandwidth.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).

The equalized clock recovery methods are available when the Advanced Signal Integrity Software license is installed.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":MEASure:CLOCk:METHod:JTF" on page 1871.

For setting other clock recovery methods, see ":MEASure:CLOCk:METHod" on page 1863.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R>}

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<R> An integer, 1-4.

<data_rate> A real number for the base data rate in bits per second.

<damping_ factor> A real number for the damping factor of the PLL.

<ojtf_loop_ bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a damping factor of 1.0.

```
myScope.WriteString ":MEASure:CLOCk:METHod:OJTF SOPLL,4E9,2.4E6,1.0"
```

Query

```
:MEASure:CLOCk:METHod:OJTF?
```

The :MEASure:CLOCk:METHod:OJTF? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod:OJTF]
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_fact>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:OJTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History New in version 4.20.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:OJTF" on page 345.

:MEASure:CLOCk:METHod:PLLTrack

Command

```
:MEASure:CLOCk:METHod:PLLTrack {OFF | ON}
```

The :MEASure:CLOCk:METHod:PLLTrack command turns transition density dependence on or off. See the help system for more information on the Transition Density Dependent setting.

This command applies to the clock recovery method being set up for the waveform source selected by the :MEASure:CLOCk:METHod:SOURce command.

Example

This example enables the Transition Density Dependent setting.

```
myScope.WriteString ":MEASure:CLOCk:METHod:PLLTrack ON"
```

Query

```
:MEASure:CLOCk:METHod:PLLTrack?
```

The :MEASure:CLOCk:METHod:PLLTrack? query returns whether or not the Transition Density Dependent setting is turned on.

Returned Format

```
[:MEASure:CLOCk:METHod:PLLTrack] {OFF | ON}
```

Example

This example places the current setting of the Transition Density Dependent setting in the string variable strTDD, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK:METHod:PLLTrack?"
strTDD = myScope.ReadString
Debug.Print strTDD
```

See Also

- ":MEASure:CLOCk:METHod:SOURce" on page 1876
- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History

New in version 4.20.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:PLLTrack" on page 348.

:MEASure:CLOCk:METHod:SOURce

Command

```
:MEASure:CLOCk:METHod:SOURce {ALL | <source>}
```

```
<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C>
   | FUNCtion<F> | WMEMory<R> | MTRend | MSPectrum | EQUalized}
```

The :MEASure:CLOCk:METHod:SOURce command selects the waveform source (or ALL sources) to which other clock recovery method setup commands apply.

Clock recovery methods can be set up for each waveform source (or for all waveform sources).

Query

```
:MEASure:CLOCk:METHod:SOURce?
```

The :MEASure:CLOCk:METHod:SOURce? query returns the waveform source to which other clock recovery method commands currently apply.

Returned Format

```
[:MEASure:CLOCk:METHod:SOURce] <source><NL>
```

```
<source> ::= {ALL | CHAN<N> | FUNC<F> | WMEM<N> | MTR | MSP | EQU}
```

See Also

- ":MEASure:CLOCk:METHod" on page 1863
- ":MEASure:CLOCk:METHod:OJTF" on page 1873
- ":MEASure:CLOCk:METHod:JTF" on page 1871
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 1868
- ":MEASure:CLOCk:METHod:ALIGn" on page 1867
- ":MEASure:CLOCk:METHod:PLLTrack" on page 1875
- ":MEASure:CLOCk:METHod:EDGE" on page 1869

History

New in version 5.20.

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:METHod:SOURce" on page 351.

:MEASure:CLOCk:VERTical

Command `:MEASure:CLOCk:VERTical {AUTO | MANual}`

The :MEASure:CLOCk:VERTical command sets the recovered clock vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the recovered clock vertical scale mode to automatic.

```
myScope.WriteString ":MEASure:CLOCk:VERTical AUTO"
```

Query `:MEASure:CLOCk:VERTical?`

The :MEASure:CLOCk:VERTical? query returns the current recovered clock vertical scale mode setting.

Returned Format `[:MEASure:CLOCk:VERTical] {AUTO | MANual}`

Example This example places the current setting of the recovered clock vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCK:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:VERTical" on page 352.

:MEASure:CLOCk:VERTical:OFFSet

Command

```
:MEASure:CLOCk:VERTical:OFFSet <offset>
```

The :MEASure:CLOCk:VERTical:OFFSet command sets the recovered clock vertical offset.

<offset> A real number for the recovered clock vertical offset.

Example This example sets the clock recovery vertical offset to 1 volt.

```
myScope.WriteString ":MEASure:CLOCK:VERTICAL:OFFSET 1"
```

Query

```
:MEASure:CLOCk:VERTical:OFFSet?
```

The :MEASure:CLOCk:VERTical:OFFSet? query returns the clock recovery vertical offset setting.

Returned Format

```
[:MEASure:CLOCk:VERTical:OFFSet] <value><NL>
```

<value> The clock recovery vertical offset setting.

Example This example places the current value of recovered clock vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CLOCK:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:VERTical:OFFSet" on page 353.

:MEASure:CLOCk:VERTical:RANGe

Command

```
:MEASure:CLOCk:VERTical:RANGe <range>
```

The :MEASure:CLOCk:VERTical:RANGe command sets the recovered clock vertical range.

<range> A real number for the full-scale recovered clock vertical range.

Example This example sets the recovered clock vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":MEASure:CLOCK:VERTICAL:RANGE 16"
```

Query

```
:MEASure:CLOCk:VERTical:RANGe?
```

The :MEASure:CLOCk:VERTical:RANGe? query returns the recovered clock vertical range setting.

Returned Format

```
[:MEASure:CLOCk:VERTical:RANGe] <value><NL>
```

<value> The recovered clock vertical range setting.

Example This example places the current value of recovered clock vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":MEASure:CLOCK:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:CLOCk:VERTical:RANGe" on page 354.

:MEASure:DDPWS — Data Dependent Pulse Width Shrinkage

Command

NOTE This command is deprecated. In its place, use the query ":MEASure:RJDJ:ALL?" on page 1087 which returns all of the RJDJ jitter measurements.

NOTE This command is available only when the Jitter Analysis Software license is installed.

```
:MEASure:DDPWS <source>
```

The :MEASure:DDPWS command measures the data dependent pulse width shrinkage for the selected source.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized<L>}

The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.

The CLOCk source is available when the recovered clock is displayed.

The EQUalized<L> source is available when the Advanced Signal Integrity Software license is installed and the equalized waveform is displayed as a function.

<N> An integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example measures the data rate of channel 1.

```
myScope.WriteString ":MEASure:DDPWS CHANnel1"
```

Query

```
:MEASure:DDPWS? <source>
```

The :MEASure:DDPWS? query returns the measured data dependent pulse width shrinkage.

Returned Format

```
[:MEASure:DDPWS] <value><NL>
```

<value> Data dependent pulse width shrinkage in seconds for the selected source.

Example This example places the current data dependent pulse width shrinkage value of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:DDPWS? CHANnel1"
```

```
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History New in version 3.10.

Version 4.20: Obsoleted, replaced by the :MEASure:RJDJ:ALL? query which returns all of the RJDJ jitter measurements.

:MEASure:FFT:PEAK1

Command `:MEASure:FFT:PEAK1 <1st_peak_number>`

The :MEASure:FFT:PEAK1command sets the peak number of the first peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<F> or WMEMory<R>.

<1st_peak _number> An integer, 1 to 100 specifying the number of the first peak.

Query `:MEASure:FFT:PEAK1?`

The :MEASure:FFT:PEAK1? query returns the peak number currently set as the first peak.

Returned Format `[:MEASure:FFT:PEAK1] <1st_peak_number><NL>`

See Also

- ":MEASure:SOURce" on page 1124
- ":MEASure:FFT:THReshold" on page 1884

History Legacy command (existed before version 3.10).

:MEASure:FFT:PEAK2

Command `:MEASure:FFT:PEAK2 <2nd_peak_number>`

The :MEASure:FFT:PEAK2 command sets the peak number of the second peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<F> or WMEMory<R>.

<2nd_peak _number> An integer, 1 to 100 specifying the number of the second peak.

Query `:MEASure:FFT:PEAK2?`

The :MEASure:FFT:PEAK2? query returns the peak number currently set as the second peak.

Returned Format `[:MEASure:FFT:PEAK1] <2nd_peak_number><NL>`

See Also

- ":MEASure:SOURce" on page 1124
- ":MEASure:FFT:THReshold" on page 1884

History Legacy command (existed before version 3.10).

:MEASure:FFT:THReshold

Command `:MEASure:FFT:THReshold <threshold_value>`

The :MEASure:FFT:THReshold command sets the peak search threshold value in dB. The dB after the threshold value is optional.

<threshold _value> A real number specifying the threshold for peaks.

Query `:MEASure:FFT:THReshold?`

The :MEASure:FFT:THReshold? query returns the peak search threshold value.

Returned Format `[:MEASure:FFT:THReshold] <threshold_value><NL>`

These :MEASure commands also operate on FFT functions:

Measure Command	Measurement Performed
:TMAX	The frequency of the maximum value in the spectrum.
:TMIN	The frequency of the minimum value in the spectrum.
:VMAX	The maximum value in the spectrum.
:VMIN	The minimum value in the spectrum.
:VPP	The range of values in the spectrum.
:VTIM	The value at a specified frequency.

See Also

- ":MEASure:FFT:PEAK1" on page 1882
- ":MEASure:FFT:PEAK2" on page 1883

History Legacy command (existed before version 3.10).

:MEASure:JITTer:STATistics

Command

```
:MEASure:JITTer:STATistics {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:STATistics command enables or disables jitter mode and allows you to view: measurement histogram (:MEASure:JITTer:HISTogram), measurement trend (:MEASure:JITTer:TRENd), and jitter spectrum (:MEASure:JITTer:SPsECtrum) if they are enabled.

The :MEASure:JITTer:STATistics command also turns on or off the ability to measure all edges in the waveform; not just the first edge on screen.

Example

This example turns the jitter measurement statistics and the "Measure All Edges" mode on.

```
myScope.WriteString ":MEASure:JITTer:STATistics ON"
```

Query

```
:MEASure:JITTer:STATistics?
```

The :MEASure:JITTer:STATistics? query returns the state of jitter statistics.

Returned Format

```
[:MEASure:JITTer:STATistics] {1 | 0}
```

Example

This example places the current setting of the jitter statistics mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:STATistics?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

History

Legacy command (existed before version 3.10).

Version 5.30: This command is deprecated, replaced by ":ANALyze:AEDGes" on page 332.

:MEASure:TIEData

Command

NOTE This command is only available when the E2681A Jitter Analysis Software, Serial Data Analysis, or the N5400A/5401A Software is installed.

```
:MEASure:TIEData <source>,{SECond | UNITinterval}, {AUTO
                     | CUSTOM,<data_rate> | VARiable,<data_rate>,<bandwidth>
                     | CLOCk}
```

The :MEASure:TIEData command measures data time interval error. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval.

If AUTO is selected, the oscilloscope selects the ideal data rate. If CUSTom is selected, you can enter your own ideal constant data rate. If VARiable is selected, a first order PLL clock recovery is used at a given data rate and loop bandwidth. If CLOCk is given, clock recovery as specified with the :MEASure:CLOCk:METHod is used.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<R> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> An integer, 1 to the number of analog input channels.

<F> An integer, 1-16.

<R> An integer, 1-4.

<data_rate> A real number for the ideal data rate for clock recovery.

<bandwidth> A real number for the loop bandwidth of the PLL clock recovery method.

Example This example measures the data time interval error on channel 1, ideal data rate set to automatic, units set to seconds.

```
myScope.WriteString ":MEASure:TIEData CHANnel1,SECond,AUTO"
```

Query

```
:MEASure:TIEData? <source>,(SECond | UNITinterval},
{AUTO | CUSTom,<frequency> | VARiable,<frequency>,<bandwidth> | CLOCk}
```

The :MEASure:TIEData? query returns the current value of the data time interval error.

Returned Format `[:MEASure:TIEData] <value>[,<result_state>]<NL>`

<value> The data time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of the data time interval error in the variable strValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:TIEData? CHANnel1,SECond,CUSTOM,1E9"
strValue = myScope.ReadString
Debug.Print strValue
```

History Legacy command (existed before version 3.10).

Version 5.50: This command is deprecated, replaced by ":MEASure:TIEData2" on page 1176.

Obsolete Serial Data Equalization Commands

:SPRocessing:CTLequalizer:ACGain

Command

```
:SPRocessing:CTLequalizer:ACGain <ac_gain>
```

The :CTLequalizer:ACGain command sets the AC Gain parameter for the Continuous Time Linear Equalization when USB31 is selected for the "# of Poles" option.

<ac_gain> A real number

Example This example sets the CTLE AC Gain parameter to 1.

```
myScope.WriteString ":SPRocessing:CTLequalizer:ACGain 1"
```

Query

```
:SPRocessing:CTLequalizer:ACGain?
```

The :SPRocessing:CTLequalizer:ACGain? query returns the CTLE's AC Gain parameter setting.

See Also

- ":SPRocessing:CTLequalizer:NUMPoles" on page 1894

History New in version 3.10.

:SPRocessing:CTLequalizer:DCGain

Command

```
:SPRocessing:CTLequalizer:DCGain <dc_gain>
```

The :CTLequalizer:DCGain command sets the DC Gain parameter for the Continuous Time Linear Equalization.

<dc_gain> A real number

Example This example sets the CTLE DC Gain parameter to 1.

```
myScope.WriteString ":SPRocessing:CTLequalizer:DCGain 1"
```

Query

```
:SPRocessing:CTLequalizer:DCGain?
```

```
The :SPRocessing:CTLequalizer:DCGain? query returns the CTLE's DC Gain p
arameter.
```

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:DISPlay

Command

```
:SPRocessing:CTLequalizer:DISPlay {(OFF | 0) | (ON | 1)}
```

The :CTLequalizer:DISPlay command turns the display of a Continuous Time Linear Equalizer (CTLE) waveform on or off.

Example This example turns on the display of a CTLE waveform.

```
myScope.WriteString ":SPRocessing:CTLequalizer:DISPlay ON"
```

Query

```
:SPRocessing:CTLequalizer:DISPlay?
```

The :SPRocessing:CTLequalizer:DISPlay? query returns whether or not the CTLE waveform is displayed.

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:FDISplay

Command `:SPRocessing:CTLequalizer:FDISplay {{0 | OFF} | {1 | ON}}`

The :SPRocessing:CTLequalizer:FDISplay command enables or disables the "display CTLE as function" setting.

You may want to disable the "display CTLE as function" setting to enable hardware acceleration.

Query `:SPRocessing:CTLequalizer:FDISplay?`

The :SPRocessing:CTLequalizer:FDISplay? query returns whether the "display CTLE as function" setting is enabled or disabled.

Returned Format `<setting><NL>`

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

See Also

- ":SPRocessing:CTLequalizer:DISPlay" on page 1892
- ":SPRocessing:FFEQualizer:FDISplay" on page 1927

History New in version 10.10.

:SPRocessing:CTLequalizer:NUMPoles

Command

```
:SPRocessing:CTLequalizer:NUMPoles {{P2Z1 | POLE2} | {P3Z1 | POLE3}
    | P3Z2 | P4Z1 | {P2ACG | USB31}}
```

The :SPRocessing:CTLequalizer:NUMPoles command selects from these Continuous Time Linear Equalizer (CTLE) options:

- {P2Z1 | POLE2} — 2 Pole 1 Zero.
- {P3Z1 | POLE3} — 3 Pole 1 Zero.
- P3Z2 — 3 Pole 2 Zeros.
- P4Z1 — 4 Pole 1 Zero.
- {P2ACG | USB31} — 2 Pole AC Gain.

Example This example selects a 2 Pole, 1 Zero CTLE.

```
myScope.WriteString ":SPRocessing:CTLequalizer:NUMPoles P2Z1"
```

Query

```
:SPRocessing:CTLequalizer:NUMPoles?
```

The :SPRocessing:CTLequalizer:NUMPoles? query returns the current "number of poles" selection.

Returned Format

```
[:SPRocessing:CTLequalizer:NUMPoles] {P2Z1 | P3Z1 | P3Z2 | P4Z1 | P2ACG}
```

See Also

- ":SPRocessing:CTLequalizer:Z1" on page 1904
- ":SPRocessing:CTLequalizer:Z2" on page 1905
- ":SPRocessing:CTLequalizer:ACGain" on page 1890

History New in version 3.10.

Version 5.75: The previous POLE3 option has been replaced by P3Z1 and P3Z2.

Version 10.10: The P4Z1 option has been added, and the new option names P2Z1 and P2ACG replace the old option names POLE2 and USB31, respectively (but operations are the same).

:SPRocessing:CTLequalizer:P1

Command

```
:SPRocessing:CTLequalizer:P1 <pole1_freq>
```

The :CTLequalizer:P1 command sets the Pole 1 frequency for the Continuous Time Linear Equalization.

<pole1_freq> A real number

Example This example sets the CTLE Pole 1 frequency to 1GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P1 1e9"
```

Query

```
:SPRocessing:CTLequalizer:P1?
```

```
The :SPRocessing:CTLequalizer:P1? query returns the CTLE's Pole 1 freque
ncy.
```

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:P2

Command

```
:SPRocessing:CTLequalizer:P2 <pole2_freq>
```

The :CTLequalizer:P1 command sets the Pole 2 frequency for the Continuous Time Linear Equalization.

<pole2_freq> A real number

Example This example sets the CTLE Pole 2 frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P2 4e9"
```

Query

```
:SPRocessing:CTLequalizer:P2?
```

The :SPRocessing:CTLequalizer:P2? query returns the CTLE's Pole 2 freque ncy.

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:P3

Command

```
:SPRocessing:CTLequalizer:P3 <pole3_freq>
```

The :CTLequalizer:P1 command sets the Pole 3 frequency for the Continuous Time Linear Equalization.

<pole3_freq> A real number

Example This example sets the CTLE Pole 3 frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P3 4e9"
```

Query

```
:SPRocessing:CTLequalizer:P3?
```

The :SPRocessing:CTLequalizer:P3? query returns the CTLE's Pole 3 frequency.

History New in version 3.10.

:SPRocessing:CTLequalizer:P4

Command

```
:SPRocessing:CTLequalizer:P4 <pole4_freq>
```

The :CTLequalizer:P4 command sets the Pole 4 frequency for the Continuous Time Linear Equalization.

<pole4_freq> A real number

Example This example sets the CTLE Pole 4 frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P4 4e9"
```

Query

```
:SPRocessing:CTLequalizer:P4?
```

The :SPRocessing:CTLequalizer:P4? query returns the CTLE's Pole 4 frequency.

History New in version 10.10.

:SPRocessing:CTLequalizer:RATe

Command `:SPRocessing:CTLequalizer:RATe <data_rate>`

The :CTLequalizer:RATe command sets the data rate for the CTLE equalizer.

<data_rate> A real number.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

Example This example sets the CTLE data rate to 3e9.

```
myScope.WriteString ":SPRocessing:CTLequalizer:RATe 3e9"
```

Query `:SPRocessing:CTLequalizer:RATe?`

```
The :SPRocessing:CTLequalizer:Rate? query returns the CTLE's data rate.
```

See Also

- ":ANALyze:SIGNal:TYPE" on page 404

History Legacy command (existed before version 3.10).

Version 5.50: When the signal type is PAM-4, a symbol rate (baud) is specified instead of a data rate (b/s).

:SPRocessing:CTLequalizer:SOURce

Command `:SPRocessing:CTLequalizer:SOURce {CHANnel<N> | FUNCtion<F> | WMEMory<R>}`

The :CTLequalizer:SOURce command sets the source for the Continuous Time Linear Equalization.

<N> An integer, 1 to the number of analog input channels.

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example sets the CTLE source to Channel 1.

```
myScope.WriteString ":SPRocessing:CTLequalizer:SOURce CHANnel1"
```

Query `:SPRocessing:CTLequalizer:SOURce?`

The :SPRocessing:CTLequalizer:SOURce? query returns the CTLE source.

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:VERTical

Command

```
:SPRocessing:CTLequalizer:VERTical {AUTO | MANual}
```

The :SPRocessing:CTLequalizer:VERTical command sets the CTLE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the CTLE signal's vertical scale mode to automatic.

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical AUTO"
```

Query

```
:SPRocessing:CTLequalizer:VERTical?
```

The :SPRocessing:CTLequalizer:VERTical? query returns the current CTLE signal's vertical scale mode setting.

Returned Format

```
[:SPRocessing:CTLequalizer:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the CTLE signal's vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History

Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:VERTical:OFFSet

Command

```
:SPRocessing:CTLequalizer:VERTical:OFFSet <offset>
```

The :SPRocessing:CTLequalizer:VERTical:OFFSet command sets the CTLE signal's vertical offset.

<offset> A real number for the CTLE signal's vertical offset.

Example This example sets the CTLE signal's vertical offset to 1 volt.

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical:OFFSet 1"
```

Query

```
:SPRocessing:CTLequalizer:VERTical:OFFSet?
```

The:SPRocessing:CTLequalizer:VERTical:OFFSet? query returns the CTLE signal's vertical offset setting.

Returned Format

```
[:SPRocessing:CTLequalizer:VERTical:OFFSet] <value><NL>
```

<value> The CTLE signal's vertical offset setting.

Example This example places the current value of the CTLE signal's vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:VERTical:RANGe

Command

```
:SPRocessing:CTLequalizer:VERTical:RANGe <range>
```

The :SPRocessing:CTLequalizer:VERTical:RANGe command sets the CTLE signal's vertical range.

<range> A real number for the full-scale CTLE signal's vertical range.

Example This example sets the CTLE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical:RANGe 16"
```

Query

```
:SPRocessing:CTLequalizer:VERTical:RANGe?
```

The :SPRocessing:CTLequalizer:VERTical:RANGe? query returns the CTLE signal's vertical range setting.

Returned Format

```
[:SPRocessing:CTLequalizer:VERTical:RANGe] <value><NL>
```

<value> The CTLE signal's vertical range setting.

Example This example places the current value of the CTLE signal's vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:SPRocessing:CTLequalizer:Z1

Command

```
:SPRocessing:CTLequalizer:Z1 <zero_freq_1>
```

The :SPRocessing:CTLequalizer:Z1 command sets the first zero frequency for the 3-pole Continuous Time Linear Equalization.

<zero_freq_1> A real number in NR3 format.

Example This example sets the 3-pole CTLE's first zero frequency to 900 MHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:Z1 650e6"
```

Query

```
:SPRocessing:CTLequalizer:Z1?
```

The :SPRocessing:CTLequalizer:Z1? query returns the 3-pole CTLE's first zero frequency.

Returned Format

```
<zero_freq_1><NL>
```

See Also

- ":SPRocessing:CTLequalizer:Z2" on page 1905
- ":SPRocessing:CTLequalizer:NUMPoles" on page 1894

History New in version 5.75.

:SPRocessing:CTLequalizer:Z2

Command `:SPRocessing:CTLequalizer:Z2 <zero_freq_2>`

The :SPRocessing:CTLequalizer:Z2 command sets the second zero frequency for the 3-pole Continuous Time Linear Equalization.

<zero_freq_2> A real number in NR3 format.

Example This example sets the 3-pole CTLE's second zero frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:Z2 4e9"
```

Query `:SPRocessing:CTLequalizer:Z2?`

The :SPRocessing:CTLequalizer:Z2? query returns the 3-pole CTLE's second zero frequency.

Returned Format `<zero_freq_2><NL>`

See Also

- ":SPRocessing:CTLequalizer:Z1" on page 1904
- ":SPRocessing:CTLequalizer:NUMPoles" on page 1894

History New in version 5.75.

:SPRocessing:CTLequalizer:ZERo

Command

```
:SPRocessing:CTLequalizer:ZERo <zero_freq>
```

The :CTLequalizer:ZERo command sets the zero frequency for the Continuous Time Linear Equalization.

<zero_freq> A real number.

Example This example sets the CTLE zero frequency to 900 MHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:ZERo 9e6"
```

Query

```
:SPRocessing:CTLequalizer:ZERo?
```

The :SPRocessing:CTLequalizer:ZERo? query returns the CTLE's zero frequency.

History Legacy command (existed before version 3.10).

Version 5.75: Now that you can specify up to two zeros for a 3-pole CTLE, this command has been replaced by ":SPRocessing:CTLequalizer:Z1" on page 1904 and ":SPRocessing:CTLequalizer:Z2" on page 1905.

:SPRocessing:DFEQualizer:NTAPs

Command

```
:SPRocessing:DFEQualizer:NTAPs <number>
```

The :DFEQualizer:NTAPs command sets the number of taps to be used in the DFE algorithm.

DFE tap indices always begin with 1 and extend to the number of taps.

<number> An integer between 1 and 40

Example This example sets the number of DFE taps to 3.

```
myScope.WriteString ":SPRocessing:DFEQualizer:NTAPs 3"
```

Query

```
:SPRocessing:DFEQualizer:NTAPs?
```

The :SPRocessing:DFEQualizer:NTAPs? query returns the number of DFE taps.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:SOURce

Command

```
:SPRocessing:DFEQualizer:SOURce {CHANnel<N> | DIFF<D> | COMMonmode<C>
    | FUNCtion<F> | WMEMory<R> | EQUalized}
```

The :DFEQualizer:SOURce command sets the source for the Decision Feedback Equalization.

Setting the source to EQUalized means the Feed-Forward Equalized (FFE) waveform is used as the DFE source.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example sets the DFE source to Channel 1.

```
myScope.WriteString ":SPRocessing:DFEQualizer:SOURce CHANnel1"
```

Query

```
:SPRocessing:DFEQualizer:SOURce?
```

The :SPRocessing:DFEQualizer:SOURce? query returns the DFE source.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:STATe

Command
```
:SPRocessing:DFEQualizer:STATe {(OFF | 0) | (ON | 1)}
```

The :DFEQualizer:STATe command turns the Decision Feedback Equalization on or off.

Example This example turns on DFE.

```
myScope.WriteString ":SPRocessing:DFEQualizer:STATe ON"
```

Query
```
:SPRocessing:DFEQualizer:STATe?
```

The :SPRocessing:DFEQualizer:STATe? query returns whether or not DFE is turned on.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP

Command

```
:SPRocessing:DFEQualizer:TAP <tap>, <value>
```

The :DFEQualizer:TAP command sets the tap value for each DFE tap. For example, when <tap> is equal to 1 then the 1st tap is set to <value>.

DFE tap indices always start at 1 and extend to the number of taps.

<tap> The tap number.

<value> The tap value

Example This example sets the DFE Tap 1 to 0.432.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP 1,0.432"
```

Query

```
:SPRocessing:DFEQualizer:TAP? <tap>
```

The :SPRocessing:DFEQualizer:TAP? query returns the DFE tap values.

See Also

- ":SPRocessing:DFEQualizer:NTAPs" on page 1907

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:AUTomatic

Command

```
:SPRocessing:DFEQualizer:TAP:AUTomatic
```

The :DFEQualizer:TAP:AUTomatic command starts the DFE tap optimization. Be sure to first specify the number of taps, the max/min tap values, and the Normalize DC Gain setting.

Example This example starts the DFE tap optimization.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:AUTomatic"
```

See Also

- ":SPRocessing:DFEQualizer:NTAPs" on page 1907
- ":SPRocessing:DFEQualizer:TAP:MIN" on page 1918
- ":SPRocessing:DFEQualizer:TAP:MAX" on page 1916
- ":SPRocessing:DFEQualizer:TAP:NORMalize" on page 1920

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:DELay

Command

```
:SPRocessing:DFEQualizer:TAP:DELay <delay>
```

The :DFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<delay> A real number

Query

```
:SPRocessing:DFEQualizer:TAP:DELay?
```

The :SPRocessing:DFEQualizer:TAP:DELay? query returns the value for the DFE Delay field.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:DELay:AUTomatic

Command `:SPRocessing:DFEQualizer:TAP:DELay:AUTomatic`

The :SPRocessing:DFEQualizer:TAP:DELay:AUTomatic command computes a DFE delay value to center a DFE eye on the screen horizontally. The current real-time eye data is used to center the DFE eye.

See Also

- ":SPRocessing:DFEQualizer:TAP:DELay" on page 1912

History New in version 10.10.

:SPRocessing:DFEQualizer:TAP:GAIN

Command `:SPRocessing:DFEQualizer:TAP:GAIN <gain>`

The eye diagram drawn after DFE is applied is attenuated. To amplify the eye back to its original size (so you can directly compare the eye at the receiver to the eye at the transmitter), a gain factor needs to be applied. The :DFEQualizer:TAP:GAIN command allows you to set this gain. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<gain> A real number

Example This example sets the gain to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:GAIN 3.23"
```

Query `:SPRocessing:DFEQualizer:TAP:GAIN?`

The :SPRocessing:DFEQualizer:TAP:GAIN? query returns the current gain value.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:LTARget

Command `:SPRocessing:DFEQualizer:TAP:LTARget <lower_target>`

The Lower Target field dictates the logical low value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical low, its value will be equal to Lower Target. The :DFEQualizer:TAP:LTARget command allows you to set this value.

<lower_target> A real number

Example This example sets the Lower Target to 1.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:LTARget 1.0"
```

Query `:SPRocessing:DFEQualizer:TAP:LTARget?`

The :SPRocessing:DFEQualizer:TAP:LTARget? query returns the current value for the Lower Target field.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:MAX

Command

```
:SPRocessing:DFEQualizer:TAP:MAX <max_tap_value>
```

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MAX command sets the upper limit on taps determined through optimization.

<max_tap_value> A real number

Example This example sets the Upper Limit field to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:MAX 3.23"
```

Query

```
:SPRocessing:DFEQualizer:TAP:MAX?
```

The :SPRocessing:DFEQualizer:TAP:MAX? query returns the Upper Limit used in the DFE tap optimization.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:MAXV

Command

```
:SPRocessing:DFEQualizer:TAP:MAXV <max_tap_value_in_volts>
```

The :SPRocessing:DFEQualizer:TAP:MAXV command sets the maximum tap value for DFE auto tap setup in volts as opposed to the :SPRocessing:DFEQualizer:TAP:MAX command that sets the max in unitless values.

If the unitless values are changed by the :SPRocessing:DFEQualizer:TAP:MAX command, they supersede the voltage values.

<max_tap_value_in_volts> A real number.

Query

```
:SPRocessing:DFEQualizer:TAP:MAXV?
```

The :SPRocessing:DFEQualizer:TAP:MAXV? query returns the maximum tap value in volts used in the DFE auto tap setup.

See Also

- ":SPRocessing:DFEQualizer:TAP:MINV" on page 1919

History New in version 10.10.

:SPRocessing:DFEQualizer:TAP:MIN

Command

```
:SPRocessing:DFEQualizer:TAP:MIN <min_tap_value>
```

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MIN command sets the lower limit on taps determined through optimization.

<min_tap_value> A real number

Example This example sets the Lower Limit field to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:MIN 3.23"
```

Query

```
:SPRocessing:DFEQualizer:TAP:MIN?
```

The :SPRocessing:DFEQualizer:TAP:MIN? query returns the Lower Limit used in the DFE tap optimization.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:MINV

Command

```
:SPRocessing:DFEQualizer:TAP:MINV <min_tap_value_in_volts>
```

The :SPRocessing:DFEQualizer:TAP:MINV command sets the minimum tap value for DFE auto tap setup in volts as opposed to the :SPRocessing:DFEQualizer:TAP:MIN command that sets the min in unitless values.

If the unitless values are changed by the :SPRocessing:DFEQualizer:TAP:MIN command, they supersede the voltage values.

<min_tap_value_in_volts> A real number.

Query

```
:SPRocessing:DFEQualizer:TAP:MINV?
```

The :SPRocessing:DFEQualizer:TAP:MINV? query returns the minimum tap value in volts used in the DFE auto tap setup.

See Also

- ":SPRocessing:DFEQualizer:TAP:MAXV" on page 1917

History New in version 10.10.

:SPRocessing:DFEQualizer:TAP:NORMalize

Command `:SPRocessing:DFEQualizer:TAP:NORMalize {{0 | OFF} | {1 | ON}}`

The :SPRocessing:DFEQualizer:TAP:NORMalize command specifies whether the Normalize DC Gain setting is ON or OFF. When ON, the eye diagram is automatically scaled so that it is the same size as the transmitted eye.

the Normalize DC Gain setting should be set (if desired) prior to calling the :SPRocessing:DFEQualizer:TAP:AUTOmatic command.

This command maps to the **Normalize DC Gain** setting in the Equalization Auto Tap Setup dialog box in the front panel graphical user interface.

Query `:SPRocessing:DFEQualizer:TAP:NORMalize?`

The :SPRocessing:DFEQualizer:TAP:NORMalize? query returns the Normalize DC Gain setting.

Returned Format `<setting><NL>`

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

See Also

- ":SPRocessing:DFEQualizer:TAP:AUTomatic" on page 1911

History New in version 6.20.

:SPRocessing:DFEQualizer:TAP:UTARget

Command

```
:SPRocessing:DFEQualizer:TAP:UTARget <upper_target>
```

The Upper Target field dictates the logical high value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical high, its value will be equal to Upper Target. The :DFEQualizer:TAP:UTARget command allows you to set this value.

<upper_target> A real number

Example This example sets the Upper Target to 1.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:UTARget 1.0"
```

Query

```
:SPRocessing:DFEQualizer:TAP:UTARget?
```

The :SPRocessing:DFEQualizer:TAP:UTARget? query returns the current value for the Upper Target field.

History Legacy command (existed before version 3.10).

:SPRocessing:DFEQualizer:TAP:WIDTh

Command `:SPRocessing:DFEQualizer:TAP:WIDTh <width>`

The :DFEQualizer:TAP:WIDTh command sets the Eye Width field for the DFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for DFE is 0.0. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:WIDTh 0.0"
```

Query `:SPRocessing:DFEQualizer:TAP:WIDTh?`

The :SPRocessing:DFEQualizer:TAP? query returns the eye width used in the DFE tap optimization.

History Legacy command (existed before version 3.10).

:SPRocessing:EQUalizer:FDCouple

Command `:SPRocessing:EQUalizer:FDCouple {{0 | OFF} | {1 | ON}}`

The :SPRocessing:EQUalizer:FDCouple command specifies whether the "display FFE as function" setting is coupled with the command that enables Feed-Forward Equalized (FFE):

- ON — The ":SPRocessing:FFEQualizer:DISPlay ON" command will enable Feed-Forward Equalized (FFE) and the "display FFE as function" setting.
- OFF — The ":SPRocessing:FFEQualizer:DISPlay ON" command will enable Feed-Forward Equalized (FFE) only.

Query `:SPRocessing:EQUalizer:FDCouple?`

The :SPRocessing:EQUalizer:FDCouple? query returns the "display FFE as function" coupling setting.

Returned Format `<setting><NL>`

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

See Also

- ":SPRocessing:FFEQualizer:DISPlay" on page 1926
- ":SPRocessing:FFEQualizer:FDISplay" on page 1927

History New in version 10.10.

:SPRocessing:FFEQualizer:BANDwidth

Command

```
:SPRocessing:FFEQualizer:BANDwidth <bandwidth>
```

The :FFEQualizer:BANDwidth command is only needed if the FFEQualizer:BWMode command is set to CUSTom and in this case it sets the bandwidth at which the response generated by equalization rolls off. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

<bandwidth> The bandwidth at which the response generated by equalization rolls off.

Query

```
:SPRocessing:FFEQualizer:BANDwidth?
```

The :SPRocessing:FFEQualizer:BANDwidth? query returns the current value for the BANDwidth parameter.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:BWMode

Command `:SPRocessing:FFEQualizer:BWMode {TSBandwidth | TTDelay | CUSTom}`

The :FFEQualizer:BWMode command sets the bandwidth at which the response generated by equalization is rolled off. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

Example This example sets the FFE Bandwidth Mode to TTDelay.

```
myScope.WriteString ":SPRocessing:FFEQualizer:BWMode TTDelay"
```

Query `:SPRocessing:FFEQualizer:BWMode?`

The :SPRocessing:FFEQualizer:BWMode? query returns the FFE Bandwidth Mode.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:DISPlay

Command

```
:SPRocessing:FFEQualizer:DISPlay {(OFF | 0) | (ON | 1)}
```

The :FFEQualizer:DISPlay command turns the display of a Feed-Forward Equalized (FFE) waveform on or off.

Example This example turns on the display of a FFE waveform.

```
myScope.WriteString ":SPRocessing:FFEQualizer:DISPlay ON"
```

Query

```
:SPRocessing:FFEQualizer:DISPlay?
```

The :SPRocessing:FFEQualizer:DISPlay? query returns whether or not the FFE waveform is displayed.

See Also

- ":SPRocessing:FFEQualizer:FDISplay" on page 1927
- ":SPRocessing:EQUalizer:FDCouple" on page 1923

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:FDISplay

Command `:SPRocessing:FFEQualizer:FDISplay {{0 | OFF} | {1 | ON}}`

The :SPRocessing:FFEQualizer:FDISplay command enables or disables the "display FFE as function" setting.

You may want to disable the "display FFE as function" setting to enable hardware acceleration.

Query `:SPRocessing:FFEQualizer:FDISplay?`

The :SPRocessing:FFEQualizer:FDISplay? query returns whether the "display FFE as function" setting is enabled or disabled.

Returned Format `<setting><NL>`

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

See Also

- ":SPRocessing:EQUalizer:FDCouple" on page 1923
- ":SPRocessing:FFEQualizer:DISPlay" on page 1926
- ":SPRocessing:CTLequalizer:FDISplay" on page 1893

History New in version 10.10.

:SProcessing:FFEQualizer:NPRecursor

Command `:SPRocessing:FFEQualizer:NPRecursor <number>`

The :FFEQualizer:NPRecursor command sets the number of precursor taps to be used in the FFE algorithm.

<number> An integer between 1 and (NTAPs - 1)

Example This example sets the number of FFE precursor taps to 3.

```
myScope.WriteString ":SPRocessing:FFEQualizer:NPRecursor 3"
```

Query `:SPRocessing:FFEQualizer:NPRecursor?`

The :SPRocessing:FFEQualizer:NPRecursor? query returns the number of FFE precursor taps.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:NTAPs

Command `:SPRocessing:FFEQualizer:NTAPs <number>`

The :FFEQualizer:NTAPs command sets the number of taps to be used in the FFE algorithm.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<number> an integer between 2 and 40

Example This example sets the number of FFE taps to 3.

```
myScope.WriteString ":SPRocessing:FFEQualizer:NTAPs 3"
```

Query `:SPRocessing:FFEQualizer:NTAPs?`

The :SPRocessing:FFEQualizer:NTAPs? query returns the number of FFE taps.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:RATe

Command `:SPRocessing:FFEQualizer:RATe <data_rate>`

The :FFEQualizer:RATe command sets the data rate for the FFE equalizer.

<data_rate> A real number.

When the signal type is PAM-4 (see :ANALyze:SIGNal:TYPE), a symbol rate (baud) is specified instead of a data rate (b/s).

Example This example sets the FFE data rate to 3e9.

```
myScope.WriteString ":SPRocessing:FFEQualizer:RATe 3e9"
```

Query `:SPRocessing:FFEQualizer:RATe?`

`The :SPRocessing:FFEQualizer:Rate? query returns the FFE's data rate.`

See Also

- ":ANALyze:SIGNal:TYPE" on page 404

History Legacy command (existed before version 3.10).

Version 5.50: When the signal type is PAM-4, a symbol rate (baud) is specified instead of a data rate (b/s).

:SPRocessing:FFEQualizer:SOURce

Command

```
:SPRocessing:FFEQualizer:SOURce {CHANnel<N> | DIFF<D> | COMMonmode<C>
    | FUNCtion<F> | WMEMory<R>}
```

The :FFEQualizer:SOURce command sets the source for the Feed-Forward Equalization.

<N> An integer, 1 to the number of analog input channels.

<D>, <C> Integers that map to the channels that display the differential and common mode waveforms, respectively.

The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.

<F> An integer, 1-16.

<R> An integer, 1-4.

Example This example sets the FFE source to Channel 1.

```
myScope.WriteString ":SPRocessing:FFEQualizer:SOURce CHANnel1"
```

Query

```
:SPRocessing:FFEQualizer:SOURce?
```

The :SPRocessing:FFEQualizer:SOURce? query returns the FFE source.

History Legacy command (existed before version 3.10).

:SProcessing:FFEQualizer:TAP

Command

```
:SPRocessing:FFEQualizer:TAP <tap>, <value>
```

The :FFEQualizer:TAP command sets the tap value for each FFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<tap> The tap number; when <tap> == 0, Tap 0 is set

<value> The tap value

Example This example sets the second FFE tap to -1.432.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP 2,-1.432"
```

Query

```
:SPRocessing:FFEQualizer:TAP? <tap>
```

The :SPRocessing:FFEQualizer:TAP? query returns the FFE tap values.

See Also

- ":SPRocessing:FFEQualizer:NTAPs" on page 1929

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:TAP:AUTomatic

Command

```
:SPRocessing:FFEQualizer:TAP:AUTomatic
```

The :FFEQualizer:TAP:AUTomatic command starts the FFE tap optimization. Be sure to first specify the number of taps and specify the Pattern and Eye Width parameters.

Example This example starts the FFE tap optimization.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:AUTomatic"
```

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:TAP:DELay

Command `:SPRocessing:FFEQualizer:TAP:DELay <delay>`

The :FFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<delay> A real number

Query `:SPRocessing:FFEQualizer:TAP:DELay?`

The :SPRocessing:FFEQualizer:TAP:DELay? query returns the value for the FFE Delay field.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:TAP:WIDTh

Command

```
:SPRocessing:FFEQualizer:TAP:WIDTh <width>
```

The :FFEQualizer:TAP:WIDTh command sets the Eye Width field for the FFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for FFE is 0.33. For more information on this parameter, refer to the Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:WIDTh 0.0"
```

Query

```
:SPRocessing:FFEQualizer:TAP:WIDTh?
```

The :SPRocessing:FFEQualizer:TAP:WIDTh? query returns the eye width used in the FFE tap optimization.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:TDELay

Command `:SPRocessing:FFEQualizer:TDELay <delay_value>`

The :FFEQualizer:TDELay command is only needed if the FFEQualizer:TDMode is set to CUSTom. To determine what this value should be, use the equation: tap delay = 1/[(data rate)x(# of taps per bit)]. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

<delay_value> A real number

Query `:SPRocessing:FFEQualizer:TDELay?`

The :SPRocessing:FFEQualizer:TDELay? query returns the current value for the tap delay.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:TDMode

Command

```
:SPRocessing:FFEQualizer:TDMode {TBITrate | CUSTom}
```

The :FFEQualizer:TDMode command sets Tap Delay field to either Track Data Rate or Custom. If you are using one tap per bit, use the TBITrate selection. If you are using multiple taps per bit, use CUSTom and then use the FFEQualizer:TDELay command to set the value. To understand more about this parameter, consult the Infiniium Serial Data Equalization User's Guide.

Example This example sets the FFE Tap Delay mode to TBITrate.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TDMode TBITrate"
```

Query

```
:SPRocessing:FFEQualizer:TDMode?
```

The :SPRocessing:FFEQualizer:TDMode? query returns the current Tap Delay mode.

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:VERTical

Command

```
:SPRocessing:FFEQualizer:VERTical {AUTO | MANual}
```

The :SPRocessing:FFEQualizer:VERTical command sets the FFE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example

This example sets the FFEE signal's vertical scale mode to automatic.

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical AUTO"
```

Query

```
:SPRocessing:FFEQualizer:VERTical?
```

The :SPRocessing:FFEQualizer:VERTical? query returns the current FFE signal's vertical scale mode setting.

Returned Format

```
[:SPRocessing:FFEQualizer:VERTical] {AUTO | MANual}
```

Example

This example places the current setting of the FFE signal's vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

History

Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:VERTical:OFFSet

Command

```
:SPRocessing:FFEQualizer:VERTical:OFFSet <offset>
```

The :SPRocessing:FFEQualizer:VERTical:OFFSet command sets the FFE signal's vertical offset.

<offset> A real number for the FFE signal's vertical offset.

Example This example sets the FFE signal's vertical offset to 1 volt.

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical:OFFSet 1"
```

Query

```
:SPRocessing:FFEQualizer:VERTical:OFFSet?
```

The:SPRocessing:FFEQualizer:VERTical:OFFSet? query returns the FFE signal's vertical offset setting.

Returned Format

```
[:SPRocessing:FFEQualizer:VERTical:OFFSet] <value><NL>
```

<value> The FFE signal's vertical offset setting.

Example This example places the current value of the FFE signal's vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical:OFFSet?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

:SPRocessing:FFEQualizer:VERTical:RANGe

Command

```
:SPRocessing:FFEQualizer:VERTical:RANGe <range>
```

The :SPRocessing:FFEQualizer:VERTical:RANGe command sets the FFE signal's vertical range.

<range> A real number for the full-scale FFE signal's vertical range.

Example This example sets the FFE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical:RANGe 16"
```

Query

```
:SPRocessing:FFEQualizer:VERTical:RANGe?
```

The :SPRocessing:FFEQualizer:VERTical:RANGe? query returns the FFE signal's vertical range setting.

Returned Format

```
[:SPRocessing:FFEQualizer:VERTical:RANGe] <value><NL>
```

<value> The FFE signal's vertical range setting.

Example This example places the current value of the FFE signal's vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"   ' Response headers off.
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical:RANGe?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

History Legacy command (existed before version 3.10).

Obsolete Trigger Commands

- ":TRIGger:PWIDth:DIRection" on page 1942
- ":TRIGger:TRANsition:DIRection" on page 1943

:TRIGger:PWIDth:DIRection

Command `:TRIGger:PWIDth[{1 | 2}]:DIRection {GTHan | LTHan}`

This command specifies whether a pulse must be wider or narrower than the width value to trigger the oscilloscope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:DIRection?`

The query returns the currently defined direction for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:DIRection] {GTHan | LTHan}<NL>`

History Legacy command (existed before version 3.10).

:TRIGger:TRANsition:DIRection

Command `:TRIGger:TRANsition[{1 | 2}]:DIRection {GTHan | LTHan}`

This command lets you look for transition violations that are greater than or less than the time specified by the :TRIGger:TRANsition:TIME command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:TRANsition:DIRection?`

The query returns the currently defined direction for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:DIRection] {GTHan | LTHan}]<NL>`

History Legacy command (existed before version 3.10).

Obsolete :TRIGger:ADVanced:PATTern Commands

Logic triggering is similar to the way that a logic analyzer captures data. This mode is useful when you are looking for a particular set of ones and zeros on a computer bus or control lines. You determine which channels the oscilloscope uses to form the trigger pattern. Because you can set the voltage level that determines a logic 1 or a logic 0, any logic family that you are probing can be captured.

There are two types of logic triggering: Pattern and State. The difference between pattern and state triggering modes is that state triggering uses one of the oscilloscope channels as a clock.

Use pattern triggering to trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.

The Pattern Trigger Mode identifies a trigger condition by looking for a specified pattern. A pattern is a logical combination of the channels. Each channel can have a value of High (H), Low (L) or Don't Care (X). A value is considered a High when your waveform's voltage level is greater than its trigger level, and a Low when the voltage level is less than its trigger level. If a channel is set to Don't Care, it is not used as part of the pattern criteria.

One additional qualifying condition determines when the oscilloscope triggers once the pattern is found. The :PATTern:CONDition command has five possible ways to qualify the trigger:

Entered The oscilloscope will trigger on the edge of the source that makes the pattern true.

Exited The oscilloscope will trigger on the edge of the source that makes the pattern false.

Present > The oscilloscope will trigger when the pattern is present for greater than the time that you specify. An additional parameter allows the oscilloscope to trigger when the pattern goes away or when the time expires.

Present < The oscilloscope will trigger when the pattern is present for less than the time that you specify.

Range The oscilloscope will trigger on the edge of the waveform that makes the pattern invalid as long as the pattern is present within the range of times that you specify.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands Before you can execute the :TRIGger:ADVanced:PATTern commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE PATTern
```

To query the oscilloscope for the advanced trigger mode, enter:

```
:TRIGger:ADVanced:MODE?
```

The :TRIGger:ADVanced:PATTern commands define the conditions for the Pattern Trigger Mode. As described in the following commands, you set up the :TRIGger:ADVanced:PATTern commands with the following commands and queries:

- ":TRIGger:ADVanced:PATTern:CONDition" on page 1946
- ":TRIGger:ADVanced:PATTern:LOGic" on page 1947
- ":TRIGger:ADVanced:PATTern:THReshold:LEVel" on page 1948

:TRIGger:ADVanced:PATTern:CONDition

Command

```
:TRIGger:ADVanced:PATTern:CONDition {
   ENTered
   | EXITed
   | {GT,<time>[,{PEXits | TIMeout}]}
   | {LT,<time>}
   | {RANGe,<gt_time>,<lt_time>}
   | {ORANGe,<gt_time>,<lt_time>}
}
```

This command describes the condition applied to the trigger pattern to actually generate a trigger.

The RANGe option specifies "inside range", and the ORANGe option specifies "outside range".

<gt_time> The minimum time (greater than time) for the trigger pattern.

<lt_time> The maximum time (less than time) for the trigger pattern.

<time> The time condition, in seconds, for the pattern trigger.

When using the GT (Present >) parameter, the PEXits (Pattern Exits) or the TIMeout parameter controls when the trigger is generated.

Query

```
:TRIGger:ADVanced:PATTern:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:ADVanced:PATTern:CONDition] {
   ENTered
   | EXITed
   | {GT,<time>[,{PEXits | TIMeout}]}
   | {LT,<time>}
   | {RANGe,<gt_time>, <lt_time>}
   | {ORANGe,<gt_time>, <lt_time>}
}<NL>
```

History Legacy command (existed before version 3.10).

Version 6.20: The OR parameter has been added.

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PATTern:CONDition" on page 1613. The outside range (ORANGe) option is added. The OR option is removed.

:TRIGger:ADVanced:PATTern:LOGic

Command

```
:TRIGger:ADVanced:PATTern:LOGic {{CHANnel<N> | <channel_list>},
   {HIGH | LOW | DONTcare | RISing | FALLing}}
```

This command defines the logic criteria for a selected channel.

<N> An integer, 1 to the number of analog input channels.

<channel_list> The channel range is from 0 to 15 in the following format.

(@1,5,7,9)	channels 1, 5, 7, and 9 are turned on.
(@1:15)	channels 1 through 15 are turned on.
(@1:5,8,14)	channels 1 through 5, channel 8, and channel 14 are turned on.

Query

```
:TRIGger:ADVanced:PATTern:LOGic? {CHANnel<N> | <channel_list>}
```

The query returns the current logic criteria for a selected channel.

Returned Format

```
[:TRIGger:ADVanced:PATTern:LOGic {CHANnel<N>|<channel_list>},]
   {HIGH | LOW | DONT | RIS | FALL}<NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PATTern:LOGic" on page 1614.

:TRIGger:ADVanced:PATTern:THReshold:LEVel

Command

```
:TRIGger:ADVanced:PATTern:THReshold:LEVel  {CHANnel<N>},<level>
```

The :TRIGger:ADVanced:PATTern:THReshold:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialogs (Edge, Glitch, and Advanced). This level also applies to all the High Threshold (HTHReshold) values in the Advanced Violation menus.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the trigger level on the specified channel, External Trigger, or Auxiliary Trigger Input.

Query

```
:TRIGger:ADVanced:PATTern:THReshold:LEVel? {CHANnel<N>}
```

The query returns the specified channel's trigger level.

Returned Format

```
[:TRIGger:ADVanced:PATTern:THReshold:LEVel {CHANnel<N>},] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LEVel" on page 1569.

Obsolete :TRIGger:ADVanced:STATe Commands

Logic triggering is similar to the way that a logic analyzer captures data. This mode is useful when you are looking for a particular set of ones and zeros on a computer bus or control lines. You determine which channels the oscilloscope uses to form the trigger pattern. Because you can set the voltage level that determines a logic 1 or a logic 0, any logic family that you are probing can be captured.

There are two types of logic triggering: Pattern and State. The difference between pattern and state triggering modes is that state triggering uses one of the oscilloscope channels as a clock.

Use state triggering when you want the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.

The State trigger identifies a trigger condition by looking for a clock edge on one channel and a pattern on the remaining channels. A pattern is a logical combination of the remaining channels. Each channel can have a value of High (H), Low (L) or Don't Care (X). A value is considered a High when your waveform's voltage level is greater than the trigger level and a Low when the voltage level is less than the trigger level. If a channel is set to Don't Care, it is not used as part of the pattern criteria. You can select the clock edge as either rising or falling.

The logic type control determines whether or not the oscilloscope will trigger when the specified pattern is found on a clock edge. When AND is selected, the oscilloscope will trigger on a clock edge when input waveforms match the specified pattern. When NAND is selected, the oscilloscope will trigger when the input waveforms are different from the specified pattern and a clock edge occurs.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands

Before you can execute the :TRIGger:ADVanced:STATe commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE STATe
```

To query the oscilloscope for the advanced trigger mode, enter:

```
:TRIGger:ADVanced:MODE?
```

The :TRIGger:ADVanced:STATe commands define the conditions for the State Trigger Mode. As described in the following commands, you set up the :TRIGger:ADVanced:STATe commands with the following commands and queries:

- ":TRIGger:ADVanced:STATe:CLOCk" on page 1950
- ":TRIGger:ADVanced:STATe:LOGic" on page 1951
- ":TRIGger:ADVanced:STATe:LTYPe" on page 1952
- ":TRIGger:ADVanced:STATe:SLOPe" on page 1953
- ":TRIGger:ADVanced:STATe:THReshold:LEVel" on page 1954

:TRIGger:ADVanced:STATe:CLOCk

Command `:TRIGger:ADVanced:STATe:CLOCk {CHANnel<N> | DONTcare}`

This command selects the source for the clock waveform in the State Trigger Mode.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:STATe:CLOCk?`

The query returns the currently selected clock source.

Returned Format `[:TRIGger:ADVanced:STATe:CLOCk] {CHAN<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:STATe:CLOCk" on page 1645.

:TRIGger:ADVanced:STATe:LOGic

Command

```
:TRIGger:ADVanced:STATe:LOGic {{CHANnel<N> | <channel_list>},
   {LOW | HIGH | DONTcare | RISing | FALLing}}
```

This command defines the logic state of the specified source for the state pattern. The command produces a settings conflict on a channel that has been defined as the clock.

<N> An integer, 1 to the number of analog input channels.

<channel_list> The channel range is from 0 to 15 in the following format.

(@1,5,7,9)	channels 1, 5, 7, and 9 are turned on.
(@1:15)	channels 1 through 15 are turned on.
(@1:5,8,14)	channels 1 through 5, channel 8, and channel 14 are turned on.

Query

```
:TRIGger:ADVanced:STATe:LOGic? {CHANnel<N> | <channel_list>}
```

The query returns the logic state definition for the specified source.

<N> N is the channel number, an integer in the range of 1 - 4.

Returned Format

```
[:TRIGger:ADVanced:STATe:LOGic {CHAN<N> | <channel_list>},]
    {LOW | HIGH | DONT | RIS | FALL}<NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:STATe:LOGic" on page 1646.

:TRIGger:ADVanced:STATe:LTYPe

Command

```
:TRIGger:ADVanced:STATe:LTYPe {AND | NAND}
```

This command defines the state trigger logic type. If the logic type is set to AND, then a trigger is generated on the edge of the clock when the input waveforms match the pattern specified by the :TRIGger:ADVanced:STATe:LOGic command. If the logic type is set to NAND, then a trigger is generated on the edge of the clock when the input waveforms do not match the specified pattern.

Query

```
:TRIGger:ADVanced:STATe:LTYPe?
```

The query returns the currently specified state trigger logic type.

Returned Format

```
[:TRIGger:ADVanced:STATe:LTYPe] {AND | NAND}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:STATe:LTYPe" on page 1647.

:TRIGger:ADVanced:STATe:SLOPe

Command

```
:TRIGger:ADVanced:STATe:SLOPe {{RISing | POSitive} | {FALLing | NEGative
}}
```

This command specifies the edge of the clock that is used to generate a trigger. The waveform source used for the clock is selected by using the :TRIGger:ADVanced:STATe:CLOCk command.

Query

```
:TRIGger:ADVanced:STATe:SLOPe?
```

The query returns the currently defined slope for the clock in State Trigger Mode.

Returned Format

```
[:TRIGger:ADVanced:STATe:SLOPe] {RIS | FALL}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:STATe:SLOPe" on page 1648.

:TRIGger:ADVanced:STATe:THReshold:LEVel

Command

```
:TRIGger:ADVanced:STATe:THReshold:LEVel  {CHANnel<N>},<level>
```

The :TRIGger:ADVanced:STATe:THReshold:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialogs (Edge, Glitch, and Advanced). This level also applies to all the High Threshold (HTHReshold) values in the Advanced Violation menus.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the trigger level on the specified channel, External Trigger, or Auxiliary Trigger Input.

Query

```
:TRIGger:ADVanced:STATe:THReshold:LEVel? {CHANnel<N>}
```

The query returns the specified channel's trigger level.

Returned Format

```
[:TRIGger:ADVanced:STATe:THReshold:LEVel {CHANnel<N>},] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LEVel" on page 1569.

Obsolete :TRIGger:ADVanced:DELay:EDLY Commands

You can set the delay mode to delay by events or time. Use Delay By Event mode to view pulses in your waveform that occur a number of events after a specified waveform edge. Infiniium Oscilloscopes identify a trigger by arming on the edge you specify, counting a number of events, then triggering on the specified edge.

- ":TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe" on page 1957
- ":TRIGger:ADVanced:DELay:EDLY:ARM:SOURce" on page 1958
- ":TRIGger:ADVanced:DELay:EDLY:EVENt:DELay" on page 1959
- ":TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe" on page 1960
- ":TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce" on page 1961
- ":TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe" on page 1962
- ":TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce" on page 1963

Arm On Use Arm On to set the source, level, and slope for arming the trigger circuitry. When setting the arm level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your arm level. The reason this is the best choice is that there may be some ringing or noise at both the 0volt and 5volt levels that can cause false triggers.

When you adjust the arm level control, a horizontal dashed line with a T on the right-hand side appears showing you where the arm level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the arm level control again, or activate the Trigger dialog.

Delay By Event Use Delay By Event to set the source, level, and edge to define an event. When setting the event level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your event level. The reason this is the best choice is that there may be some ringing or noise at both the 0volt and 5volt levels that can cause false triggers.

Event Use Event to set the number of events (edges) that must occur after the oscilloscope is armed until it starts to look for the trigger edge.

Trigger On Use Trigger On to set the trigger source and trigger slope required to trigger the oscilloscope. Each source can have only one level, so if you are arming and triggering on the same source, only one level is used.

Set the Mode Before Executing Commands Before you can execute the :TRIGger:ADVanced:DELay commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE DELay
```

The ADVanced DELay commands define the conditions for the Delay Trigger Mode. The Delay By Events Mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. After entering the commands above, to select Delay By Events Mode, enter:

```
:TRIGger:ADVanced:DELay:MODE EDLY
```

Then you can use the Event Delay (EDLY) commands and queries for ARM, EVENt, and TRIGger on the following pages.

To query the oscilloscope for the advanced trigger mode or the advanced trigger delay mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:DELay:MODE?
```

:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe

Command `:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe {NEGative|POSitive}`

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay By Event trigger mode.

Query `:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe?`

The query returns the currently defined slope for the Delay By Event trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:EDLY:ARM:SLOPe] {NEGative|POSitive}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:ARM:SLOPe" on page 1576.

:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce

Command `:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce {CHANnel<N>}`

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay By Event trigger mode.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce?`

The query returns the currently defined Arm On source for the Delay By Event trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:EDLY:ARM:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:ARM:SOURce" on page 1577.

:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay

Command `:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay <edge_number>`

This command sets the event count for a Delay By Event trigger event.

<edge_num> An integer from 0 to 16,000,000 specifying the number of edges to delay.

Query `:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay?`

The query returns the currently defined number of events to delay before triggering on the next Trigger On condition in the Delay By Event trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:EDLY:EVENt:DELay] <edge_number><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:EDELay:COUNt" on page 1578.

:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe

Command

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Event trigger event.

Query

```
:TRIGger:ADVanced:DELay:EDLY:EVENt:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:EDLY:EVENt:SLOPe] {NEGative|POSitive}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:EDELay:SLOPe" on page 1579.

:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce

Command `:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce {CHANnel<N>}`

This command sets the Event source for a Delay By Event trigger event.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce?`

The query returns the currently defined Event source in the Delay By Event trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:EDLY:EVENt:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:EDELay:SOURce" on page 1580.

:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe

Command

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Event trigger event.

Query

```
:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:EDLY:TRIGger:SLOPe] {NEGative|POSitive}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:TRIGger:SLOPe" on page 1584.

:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce

Command `:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce {CHANnel<N>}`

This command sets the Trigger On source for a Delay By Event trigger event.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce?`

The query returns the currently defined Trigger On source for the event in the Delay By Event trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:EDLY:TRIGger:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:TRIGger:SOURce" on page 1585.

Obsolete :TRIGger:ADVanced:DELay:TDLY Commands

You can set the delay mode to delay by events or time. Use Delay By Time mode to view pulses in your waveform that occur a long time after a specified waveform edge. The Delay by Time identifies a trigger condition by arming on the edge you specify, waiting a specified amount of time, then triggering on a specified edge. This can be thought of as two-edge triggering, where the two edges are separated by a selectable amount of time.

It is also possible to use the Horizontal Position control to view a pulse some period of time after the trigger has occurred. The problem with this method is that the further the pulse is from the trigger, the greater the possibility that jitter will make it difficult to view. Delay by Time eliminates this problem by triggering on the edge of interest.

- ":TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe" on page 1966
- ":TRIGger:ADVanced:DELay:TDLY:ARM:SOURce" on page 1967
- ":TRIGger:ADVanced:DELay:TDLY:DELay" on page 1968
- ":TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe" on page 1969
- ":TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce" on page 1970

Arm On Use Arm On to set the source, level, and slope for the arming condition. When setting the arm level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your arm level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the arm level control, a horizontal dashed line with a T on the right-hand side appears showing you where the arm level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the arm level control again, or activate the Trigger dialog.

Delay By Time Use Delay By Time to set the amount of delay time from when the oscilloscope is armed until it starts to look for the trigger edge. The range is from 30 ns to 160 ms.

Trigger On Use Trigger On to set the source and slope required to trigger the oscilloscope. Trigger On Level is slaved to Arm On Level.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands Before you can execute the :TRIGger:ADVanced:DELay commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE DELay
```

The ADVanced DELay commands define the conditions for the Delay Trigger Mode. The Delay By Time Mode lets you view pulses in your waveform that occur a specified time after a specified waveform edge. After entering the commands above, to select Delay By Time Mode, enter:

```
:TRIGger:ADVanced:DELay:MODE TDLY
```

Then you can use the Time Delay (TDLY) commands and queries for ARM, DELay, and TRIGger on the following pages.

To query the oscilloscope for the advanced trigger mode or the advanced trigger delay mode, enter:

```
:TRIGger:ADVanced:MODE?  or
:TRIGger:ADVanced:DELay:MODE?
```

:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe

Command `:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe {NEGative|POSitive}`

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay By Time trigger mode.

Query `:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe?`

The query returns the currently defined slope for the Delay By Time trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:TDLY:ARM:SLOPe] {NEGative|POSitive}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:ARM:SLOPe" on page 1576.

:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce

Command `:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce {CHANnel<N>}`

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay By Time trigger mode.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce?`

The query returns the currently defined channel source for the Delay By Time trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:TDLY:ARM:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:ARM:SOURce" on page 1577.

:TRIGger:ADVanced:DELay:TDLY:DELay

Command

```
:TRIGger:ADVanced:DELay:TDLY:DELay <delay>
```

This command sets the delay for a Delay By Time trigger event.

<delay> Time, in seconds, set for the delay trigger, from 5 ns to 10 s.

Query

```
:TRIGger:ADVanced:DELay:TDLY:DELay?
```

The query returns the currently defined time delay before triggering on the next Trigger On condition in the Delay By Time trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:TDLY:DELay] <delay><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:TDELay:TIME" on page 1582.

:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe

Command

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe
{NEGative|POSitive}
```

This command sets the trigger slope for the Delay By Time trigger event.

Query

```
:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe?
```

The query returns the currently defined slope for an event in the Delay By Time trigger mode.

Returned Format

```
[:TRIGger:ADVanced:DELay:TDLY:TRIGger:SLOPe] {NEGative|POSitive}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:TRIGger:SLOPe" on page 1584.

:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce

Command `:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce {CHANnel<N>}`

This command sets the Trigger On source for a Delay By Time trigger event.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce?`

The query returns the currently defined Trigger On source in the Delay By Time trigger mode.

Returned Format `[:TRIGger:ADVanced:DELay:TDLY:TRIGger:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:DELay:MODE" on page 1581 and ":TRIGger:DELay:TRIGger:SOURce" on page 1585.

Obsolete Advanced Violation Trigger Modes

Violation triggering helps you find conditions within your circuit that violate the design rules. There are four types of violation triggering: Pulse Width, Setup and Hold Time, and Transition.

- ":TRIGger:ADVanced:VIOLation:MODE" on page 1972

PWIDth This mode lets you find pulses that are wider than the rest of the pulses in your waveform. It also lets you find pulses that are narrower than the rest of the pulses in the waveform.

SETup This mode lets you find violations of setup and hold times in your circuit. Use this mode to select setup time triggering, hold time triggering, or both setup and hold time triggering.

TRANsition This mode lets you find any edge in your waveform that violates a rise time or fall time specification. The Infiniium oscilloscope can be set to trigger on rise times or fall times that are too slow or too fast.

:TRIGger:ADVanced:VIOLation:MODE

Command

```
:TRIGger:ADVanced:VIOLation:MODE {PWIDth | SETup | TRANsition}
```

After you have selected the advanced trigger mode with the commands :TRIGger:MODE ADVanced and :TRIGger:ADVanced:MODE VIOLation, the :TRIGger:ADVanced:VIOLation:MODE <violation_mode> command specifies the mode for trigger violations. The <violation_mode> is either PWIDth, SETup, or TRANsition.

Query

```
:TRIGger:ADVanced:VIOLation:MODE?
```

The query returns the currently defined mode for trigger violations.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:MODE] {PWIDth | SETup | TRANsition}<NL>
```

History

Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:MODE" on page 1572.

Obsolete :TRIGger:ADVanced:VIOLation:PWIDth Commands

Use Pulse Width Violation Mode to find pulses that are wider than the rest of the pulses in your waveform. You can also use this mode to find pulses that are narrower than the rest of the pulses in the waveform.

The oscilloscope identifies a pulse width trigger by looking for a pulse that is either wider than or narrower than other pulses in your waveform. You specify the pulse width and pulse polarity (positive or negative) that the oscilloscope uses to determine a pulse width violation. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level.

When looking for narrower pulses, pulse width less than (Width <) trigger is the same as glitch trigger.

- ":TRIGger:ADVanced:VIOLation:PWIDth:DIRection" on page 1975
- ":TRIGger:ADVanced:VIOLation:PWIDth:POLarity" on page 1976
- ":TRIGger:ADVanced:VIOLation:PWIDth:SOURce" on page 1977
- ":TRIGger:ADVanced:VIOLation:PWIDth:WIDTh" on page 1978

Source Use Source to select the oscilloscope channel used to trigger the oscilloscope.

Level Use the Level control to set the voltage level through which the pulse must pass before the oscilloscope will trigger.

When setting the trigger level for your waveform, it is usually best to choose a voltage value that is equal to the voltage value at the mid point of your waveform. For example, if you have a waveform with a minimum value of 0 (zero) volts and a maximum value of 5 volts, then 2.5 volts is the best place to set your trigger level. The reason this is the best choice is that there may be some ringing or noise at both the 0-volt and 5-volt levels that can cause false triggers.

When you adjust the trigger level control, a horizontal dashed line with a T on the right-hand side appears showing you where the trigger level is with respect to your waveform. After a period of time the dashed line will disappear. To redisplay the line, adjust the trigger level control again, or activate the Trigger dialog. A permanent icon with arrow (either T, T_L, or T_H) is also displayed on the right side of the waveform area, showing the trigger level.

Polarity Use the Polarity control to specify positive or negative pulses.

Direction Use Direction to set whether a pulse must be wider (Width >) or narrower (Width <) than the width value to trigger the oscilloscope.

Width Use the Width control to define how wide of a pulse will trigger the oscilloscope. The glitch width range is from 1.5 ns to 10 s.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands

Before executing the :TRIGger:ADVanced:VIOLation:PWIDth commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE PWIDth
```

To query the oscilloscope for the advanced trigger violation mode, enter:

```
:TRIGger:ADVanced:VIOLation:MODE?
```

:TRIGger:ADVanced:VIOLation:PWIDth:DIRection

Command `:TRIGger:ADVanced:VIOLation:PWIDth:DIRection {GTHan | LTHan}`

This command specifies whether a pulse must be wider or narrower than the width value to trigger the oscilloscope.

Query `:TRIGger:ADVanced:VIOLation:PWIDth:DIRection?`

The query returns the currently defined direction for the pulse width trigger.

Returned Format `[:TRIGger:ADVanced:VIOLation:PWIDth:DIRection] {GTHan | LTHan}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PWIDth:MODE" on page 1616.

:TRIGger:ADVanced:VIOLation:PWIDth:POLarity

Command

```
:TRIGger:ADVanced:VIOLation:PWIDth:POLarity {NEGative | POSitive}
```

This command specifies the pulse polarity that the oscilloscope uses to determine a pulse width violation. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level.

Query

```
:TRIGger:ADVanced:VIOLation:PWIDth:POLarity?
```

The query returns the currently defined polarity for the pulse width trigger.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:PWIDth:POLarity] {NEGative | POSitive}<NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PWIDth:POLarity" on page 1617.

:TRIGger:ADVanced:VIOLation:PWIDth:SOURce

Command `:TRIGger:ADVanced:VIOLation:PWIDth:SOURce {CHANnel<N>}`

This command specifies the channel source used to trigger the oscilloscope with the pulse width trigger.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage through which the pulse must pass before the oscilloscope will trigger.

Query `:TRIGger:ADVanced:VIOLation:PWIDth:SOURce?`

The query returns the currently defined channel source for the pulse width trigger.

Returned Format `[:TRIGger:ADVanced:VIOLation:PWIDth:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PWIDth:SOURce" on page 1619.

:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh

Command `:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh <width>`

This command specifies how wide a pulse must be to trigger the oscilloscope.

<width> Pulse width, which can range from 1.5 ns to 10 s.

Query `:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh?`

The query returns the currently defined width for the pulse.

Returned Format `[:TRIGger:ADVanced:VIOLation:PWIDth:WIDTh] <width><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:PWIDth:WIDTh" on page 1621.

Obsolete :TRIGger:ADVanced:VIOLation:SETup Commands

Use Setup Violation Mode to find violations of setup and hold times in your circuit.

- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce" on page 1982
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE" on page 1983
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel" on page 1984
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce" on page 1985
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold" on page 1986
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold" on page 1987
- ":TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME" on page 1988
- ":TRIGger:ADVanced:VIOLation:SETup:MODE" on page 1989
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce" on page 1990
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE" on page 1991
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel" on page 1992
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce" on page 1993
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold" on page 1994
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold" on page 1995
- ":TRIGger:ADVanced:VIOLation:SETup:SETup:TIME" on page 1996
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce" on page 1997
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE" on page 1998
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel" on page 1999
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce" on page 2000
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshold" on page 2001
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshold" on page 2002
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:HoldTIMe (HTIMe)" on page 2003
- ":TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe (STIMe)" on page 2004

Mode Use MODE to select Setup, Hold, or both Setup and Hold time triggering.

You can have the oscilloscope trigger on violations of setup time, hold time, or both setup and hold time. To use Setup Violation Type, the oscilloscope needs a clock waveform, used as the reference, and a data waveform for the trigger source.

Setup Time Mode When using the Setup Time Mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the thresholds during this time window. If the waveform crosses a threshold within the time window, a violation event occurs and the oscilloscope triggers.

Hold Time Mode When using Hold Time Mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

Setup and Hold Time Mode When using the Setup and Hold Time Mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers.

Data Source Use the data source (DSOurce) command to select the channel used as the data, the low-level data threshold, and the high-level data threshold. For data to be considered valid, it must be below the lower threshold or above the upper threshold during the time of interest.

DSOurce Use DSOurce to select the channel you want to use for the data source.

Low Threshold Use the low threshold (LTHReshold) to set the minimum threshold for your data. Data is valid below this threshold.

High Threshold Use the high threshold (HTHReshold) to set the maximum threshold for your data. Data is valid above this threshold.

Clock Source Use the clock source (CSOurce) command to select the clock source, trigger level, and edge polarity for your clock. Before the trigger circuitry looks for a setup or hold time violation, the clock must pass through the voltage level you have set.

CSOurce Use CSOurce to select the channel you want to use for the clock source.

LEVel Use LEVel to set voltage level on the clock waveform as given in the data book for your logic family.

RISing or FALLing Use RISing or FALLing to select the edge of the clock the oscilloscope uses as a reference for the setup or hold time violation trigger.

Time

Setup Time Use SETup to set the amount of setup time used to test for a violation. The setup time is the amount of time that the data has to be stable and valid prior to a clock edge. The minimum is 1.5 ns; the maximum is 20 ns.

Hold Time Use HOLD to set the amount of hold time used to test for a violation. The hold time is the amount of time that the data has to be stable and valid after a clock edge. The minimum is 1.5 ns; the maximum is 20 ns.

Setup and Hold Use SHOLd (Setup and Hold) to set the amount of setup and hold time used to test for a violation.

The setup time is the amount of time that the data has to be stable and valid prior to a clock edge. The hold time is the amount of time that the data waveform has to be stable and valid after a clock edge.

The setup time plus hold time equals 20 ns maximum. So, if the setup time is 1.5 ns, the maximum hold time is 18.5 ns.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands

Before executing the :TRIGger:ADVanced:VIOLation:SETup commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE SETup  and
:TRIGger:ADVanced:VIOLation:SETup:MODE <setup_mode>
```

Where <setup_mode> includes SETup, HOLD, and SHOLd.

To query the oscilloscope for the advanced trigger violation setup mode, enter:

```
:TRIGger:ADVanced:VIOLation:SETup:MODE?
```

:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce

Command `:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce {CHANnel<N>}`

This command specifies the clock source for the clock used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce?`

The query returns the currently defined clock source for the trigger hold violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce" on page 1638.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE

Command `:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE {RISing | FALLing}`

This command specifies the edge for the clock source used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

Query `:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE?`

The query returns the currently defined clock source edge for the trigger hold violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:EDGE] {RIS | FALL}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce:EDGE" on page 1639.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel {{CHANnel<N>},<leve
l>}
```

This command specifies the level for the clock source used for the trigger hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage level for the trigger hold violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel? {CHANnel<N>}
```

The query returns the specified clock source level for the trigger hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:CSOurce:LEVel {CHANnel<N>},] <le
vel><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LEVel" on page 1569.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce

Command `:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce {CHANnel<N>}`

The data source commands specify the data source for the trigger hold violation.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce?`

The query returns the currently defined data source for the trigger hold violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:DSOurce" on page 1640.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold {{CHANnel<N>},
<level>}
```

This command specifies the data source for the trigger hold violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold? {CHANnel<N>}
```

The query returns the specified data source for the trigger hold violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:HTHReshold {CHANnel<N>},
] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:HTHReshold" on page 1567.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold {{CHANnel<N>},
<level>}
```

This command specifies the data source for the trigger hold violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold? {CHANnel<N>}
```

The query returns the specified data source for the trigger hold violation, and the low data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:DSOurce:LTHReshold {CHANnel<N>},
] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LTHReshold" on page 1571.

:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME

Command

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME <time>
```

This command specifies the amount of hold time used to test for a trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

<time> Hold time, in seconds.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME?
```

The query returns the currently defined hold time for the trigger violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:HOLD:TIME] <time><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:HoldTIMe (HTIMe)" on page 1641.

:TRIGger:ADVanced:VIOLation:SETup:MODE

Command `:TRIGger:ADVanced:VIOLation:SETup:MODE {SETup | HOLD | SHOLd}`

SETup When using the setup time mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the trigger level thresholds during this time window. If the waveform crosses a threshold during this time window, a violation event occurs and the oscilloscope triggers.

HOLD When using the hold time mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

SHOLd When using the setup and hold time mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers. The total time allowed for the sum of setup time plus hold time is 20 ns maximum.

Query `:TRIGger:ADVanced:VIOLation:SETup:MODE?`

The query returns the currently selected trigger setup violation mode.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:MODE] {SETup | HOLD | SHOLd}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:MODE" on page 1642.

:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce {CHANnel<N>}
```

This command specifies the clock source for the clock used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 to the number of analog input channels.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce?
```

The query returns the currently defined clock source for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce]
{CHANnel<N>}<NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce" on page 1638.

:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE

Command `:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE {RISing | FALLing}`

This command specifies the edge for the clock source used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

Query `:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE?`

The query returns the currently defined clock source edge for the trigger setup violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:EDGE] {RIS | FALL}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce:EDGE" on page 1639.

:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel {{CHANnel<N>},<lev
el>}
```

This command specifies the level for the clock source used for the trigger setup violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup or hold time violation.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage level for the trigger setup violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel? {CHANnel<N>}
```

The query returns the specified clock source level for the trigger setup violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:CSOurce:LEVel
{CHANnel<N>},] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LEVel" on page 1569.

:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce

Command `:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce {CHANnel<N>}`

The data source commands specify the data source for the trigger setup violation.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce?`

The query returns the currently defined data source for the trigger setup violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:DSOurce" on page 1640.

:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold {{CHANnel<N>}
,<level>}
```

This command specifies the data source for the trigger setup violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger setup violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold? {CHANnel<N>}
```

The query returns the specified data source for the trigger setup violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:HTHReshold {CHANnel<N>}
,] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:HTHReshold" on page 1567.

:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold {{CHANnel<N>}
,<level>}
```

This command specifies the data source for the trigger setup violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger setup violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold? {CHANnel<N>}
```

The query returns the specified data source for the trigger setup violation, and the low data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SETup:DSOurce:LTHReshold {CHANnel<N>}
,] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LTHReshold" on page 1571.

:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME

Command `:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME <time>`

This command specifies the amount of setup time used to test for a trigger violation. The setup time is the amount of time that the data must be stable and valid prior to a clock edge.

<time> Setup time, in seconds.

Query `:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME?`

The query returns the currently defined setup time for the trigger violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SETup:TIME] <time><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:SetupTIMe" on page 1643.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce

Command `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce {CHANnel<N>}`

This command specifies the clock source for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce?`

The query returns the currently defined clock source for the trigger setup and hold violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce" on page 1638.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE

Command `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE {RISing | FALLing}`

This command specifies the clock source trigger edge for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

Query `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE?`

The query returns the currently defined clock source edge for the trigger setup and hold violation level for the clock source.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:EDGE] {RIS | FALL}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:CSOurce:EDGE" on page 1639.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel {{CHANnel<N>},<level>}
```

This command specifies the clock source trigger level for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage level for the trigger setup and hold violation clock waveform, and depends on the type of circuitry logic you are using.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel? {CHANnel<N>}
```

The query returns the specified clock source level for the trigger setup and hold violation level for the clock source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:CSOurce:LEVel {CHANnel<N>},] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LEVel" on page 1569.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce {CHANnel<N>}
```

The data source commands specify the data source for the trigger setup and hold violation.

<N> An integer, 1 to the number of analog input channels.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce?
```

The query returns the currently defined data source for the trigger setup and hold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce] {CHANnel<N>}<NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:DSOurce" on page 1640.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshold {{CHANnel<N>}
,<level>}
```

This command specifies the data source for the trigger setup and hold violation, and the high-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger setup and hold violation, and is used with the high and low threshold data source commands.

Query

```
:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshold? {CHANnel<N>}
```

The query returns the specified data source for the trigger setup and hold violation, and the high data threshold for the data source.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:HTHReshold {CHANnel<N>}
,] <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:HTHReshold" on page 1567.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshold

Command `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshold {{CHANnel<N>},<level>}`

This command specifies the data source for the trigger setup and hold violation, and the low-level data threshold for the selected data source. Data is valid when it is above the high-level data threshold, and when it is below the low-level data threshold.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the data threshold level for the trigger setup and hold violation, and is used with the high and low threshold data source commands.

Query `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshold? {CHANnel<N>}`

The query returns the specified data source for the setup and trigger hold violation, and the low data threshold for the data source.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:DSOurce:LTHReshold {CHANnel<N>},] <level><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LTHReshold" on page 1571.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:HoldTIMe (HTIMe)

Command `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:HoldTIMe <time>`

This command specifies the amount of hold time used to test for both a setup and hold trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

<time> Hold time, in seconds.

Query `:TRIGger:ADVanced:VIOLation:SETup:SHOLD:HoldTIMe?`

The query returns the currently defined hold time for the setup and hold trigger violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SHOLD:HoldTIMe] <time><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:HoldTIMe (HTIMe)" on page 1641.

:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe (STIMe)

Command `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe <time>`

This command specifies the amount of setup time used to test for both a setup and hold trigger violation. The setup time is the amount of time that the data must be stable and valid before a clock edge.

<time> Setup time, in seconds.

Query `:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe?`

The query returns the currently defined setup time for the setup and hold trigger violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:SETup:SHOLd:SetupTIMe] <time><NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:SHOLd:SetupTIMe" on page 1643.

Obsolete :TRIGger:ADVanced:VIOLation:TRANsition Commands

Use Transition Violation Mode to find any edge in your waveform that violates a rise time or fall time specification. Infiniium Oscilloscopes find a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

The rise time is measured from the time that your waveform crosses the low threshold until it crosses the high threshold. The fall time is measured from the time that the waveform crosses the high threshold until it crosses the low threshold.

- ":TRIGger:ADVanced:VIOLation:TRANsition" on page 2006
- ":TRIGger:ADVanced:VIOLation:TRANsition:SOURce" on page 2007
- ":TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold" on page 2008
- ":TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold" on page 2009
- ":TRIGger:ADVanced:VIOLation:TRANsition:TYPE" on page 2010

Source Use Source to select the channel used for a transition violation trigger.

Low Threshold Use Low Threshold to set the low voltage threshold.

High Threshold Use High Threshold to set the high voltage threshold.

Type Use Type to select Rise Time or Fall Time violation.

Trigger On Trigger On parameters include > Time and < Time.

> Time Use > Time to look for transition violations that are longer than the time specified.

< Time Use < Time to look for transition violations that are less than the time specified.

Time Use Time to set the amount of time to determine a rise time or fall time violation.

Available trigger conditioning includes HOLDoff and HYSTeresis (Noise Reject).

Set the Mode Before Executing Commands Before executing the :TRIGger:ADVanced:VIOLation:TRANsition commands, set the mode by entering:

```
:TRIGger:MODE ADVanced  and
:TRIGger:ADVanced:MODE VIOLation  and
:TRIGger:ADVanced:VIOLation:MODE TRANsition
```

To query the oscilloscope for the advanced trigger violation mode, enter:

```
:TRIGger:ADVanced:VIOLation:MODE?
```

:TRIGger:ADVanced:VIOLation:TRANsition

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:{GTHan | LTHan} <time>
```

This command lets you look for transition violations that are greater than or less than the time specified.

<time> The time for the trigger violation transition, in seconds.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:{GTHan | LTHan}?
```

The query returns the currently defined time for the trigger transition violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:{GTHan | LTHan}] <time><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:TRANsition:MODE" on page 1654 and ":TRIGger:TRANsition:TIME" on page 1657.

:TRIGger:ADVanced:VIOLation:TRANsition:SOURce

Command `:TRIGger:ADVanced:VIOLation:TRANsition:SOURce {CHANnel<N>}`

The transition source command lets you find any edge in your waveform that violates a rise time or fall time specification. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

<N> An integer, 1 to the number of analog input channels.

Query `:TRIGger:ADVanced:VIOLation:TRANsition:SOURce?`

The query returns the currently defined transition source for the trigger transition violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce] {CHANnel<N>}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:TRANsition:SOURce" on page 1656.

:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold {{CHANnel<N>},<
level>}
```

This command lets you specify the source and high threshold for the trigger violation transition. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage threshold level for the trigger transition violation, and is used with the high and low threshold transition source commands.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
HTHReshold? {CHANnel<N>}
```

The query returns the specified transition source for the trigger transition high threshold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:HTHReshold {CHANnel<N>},]
 <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:HTHReshold" on page 1567.

:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold

Command

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold {{CHANnel<N>},<
level>}
```

This command lets you specify the source and low threshold for the trigger violation transition. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

<N> An integer, 1 to the number of analog input channels.

<level> A real number for the voltage threshold level for the trigger transition violation, and is used with the high and low threshold transition source commands.

Query

```
:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:
LTHReshold? {CHANnel<N>}
```

The query returns the currently defined transition source for the trigger transition low threshold violation.

Returned Format

```
[:TRIGger:ADVanced:VIOLation:TRANsition:SOURce:LTHReshold {CHANnel<N>},]
 <level><NL>
```

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:LTHReshold" on page 1571.

:TRIGger:ADVanced:VIOLation:TRANsition:TYPE

Command `:TRIGger:ADVanced:VIOLation:TRANsition:TYPE {RISetime | FALLtime}`

This command lets you select either a rise time or fall time transition violation trigger event.

Query `:TRIGger:ADVanced:VIOLation:TRANsition:TYPE?`

The query returns the currently defined transition type for the trigger transition violation.

Returned Format `[:TRIGger:ADVanced:VIOLation:TRANsition:TYPE] {RISetime | FALLtime}<NL>`

History Legacy command (existed before version 3.10).

Version 10.00: This command is deprecated. Use instead: ":TRIGger:TRANsition:TYPE" on page 1658.

50 Waveform Sources

Many commands have a waveform <source> parameters:

```
<source> ::= {CHANnel<N> | DIFF<D> | COMMonmode<C> | WMEMory<R>
    | FUNCtion<F> | CLOCk | EQUalized<L> | MTRend | MSPectrum
    | XT<X> | PNOise | INPut | CORRected | ERRor | LFPR
    | DIGital<M> | BUS<B>}
```

where:

CHANnel<N>	<N> is an integer, 1 to the number of analog input channels (up to 40 in a MultiScope system).
DIFF<D>, COMMonmode<C>	<D>, <C> are integers that map to the channels that display the differential and common mode waveforms, respectively. The DIFF and COMMonmode sources are just aliases that can be used in place of channel names when referring to differential or common mode waveforms. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF<D> refers to the differential waveform of a differential pair and COMMonmode<C> refers to the common mode waveform of a differential pair. Channels are paired according to the ":ACQuire:DIFFerential:PARTner" on page 302 setting.
WMEMory<R>	<R> is an integer, 1-4.
FUNCtion<F>	<F> is an integer, 1-16.
CLOCk	The CLOCk source is available when the recovered clock is displayed.
EQUalized<L>	The EQUalized<L> source is available when the Advanced Signal Integrity Software license is installed and the equalized waveform is displayed as a function. <L> is an integer, 1-4, identifying the equalized function waveform.
MTRend, MSPectrum	The MTRend and MSPectrum sources are available when the Jitter Analysis Software license is installed and the features are enabled.
HISTogram	HISTogram sources are available in commands that measure Meas Histogram math functions, the histogram waveform, or memories containing histograms.

XT<X>	<X> is an integer, 1-4, identifying the crosstalk waveform.
PNOise	The PNOise source is available when the Jitter and Vertical Noise Analysis Software license is installed and the Phase Noise analysis feature is enabled.
INPut, CORRected, ERRor, LFPR	These are the SNDR Input, linear fit Pulse Corrected, linear fit Error e(k), and Linear Fit Pulse Response p(k) waveforms that can be enabled by the Signal to Noise and Distortion Ratio (SNDR) measurements. See ":MEASure:SNDR" on page 1122.
DIGital<M>	<M> is an integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.
BUS<B>	<B> is an integer, 1-4. Buses are available on mixed-signal oscilloscopes.
PODALL, POD1, POD2	The PODALL source is all digital channels, POD1 is d0-d7, and POD2 is d8-d15.

51 Error Messages

This chapter describes the error messages and how they are generated. The possible causes for the generation of the error messages are also listed in the following table.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error -350,"Queue overflow". Any time the error queue overflows, the oldest 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).

Reading an error from the head of the queue removes that error from the queue, and opens a position at the tail of the queue for a new error. When all errors have been read from the queue, subsequent error queries return 0,"No error".

The error queue is cleared when any of the following occur:

- the instrument is powered up,
- a *CLS command is sent,
- the last item from the queue is read, or
- the instrument is switched from talk only to addressed mode on the front panel.

Error Numbers

The error numbers are grouped according to the type of error that is detected.

- +0 indicates no errors were detected.
- -100 to -199 indicates a command error was detected
- -200 to -299 indicates an execution error was detected.
- -300 to -399 indicates a device-specific error was detected.
- -400 to-499 indicates a query error was detected.
- +1 to +32767 indicates an oscilloscope specific error has been detected.

Command Errors

An error number in the range -100 to -199 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class sets the command error bit (bit 5) in the event status register and indicates that one of the following events occurred:

- An IEEE 488.2 syntax error was detected by the parser. That is, a computer-to-oscilloscope message was received that is in violation of the IEEE 488.2 standard. This may be a data element that violates the oscilloscope's listening formats, or a data type that is unacceptable to the oscilloscope.
- An unrecognized header was received. Unrecognized headers include incorrect oscilloscope-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors do not generate execution errors, oscilloscope-specific errors, or query errors.

Execution Errors

An error number in the range -200 to -299 indicates that an error was detected by the instrument's execution control block. The occurrence of any error in this class causes the execution error bit (bit 4) in the event status register to be set. It also indicates that one of the following events occurred:

- The program data following a header is outside the legal input range or is inconsistent with the oscilloscope's capabilities.
- A valid program message could not be properly executed due to some oscilloscope condition.

Execution errors are reported by the oscilloscope after expressions are evaluated and rounding operations are completed. For example, rounding a numeric data element will not be reported as an execution error. Events that generate execution errors do not generate command errors, oscilloscope specific errors, or query errors.

Device- or Oscilloscope-Specific Errors

An error number in the range of -300 to -399 or +1 to +32767 indicates that the instrument has detected an error caused by an oscilloscope operation that did not properly complete. This may be due to an abnormal hardware or firmware condition. For example, this error may be generated by a self-test response error, or a full error queue. The occurrence of any error in this class causes the oscilloscope-specific error bit (bit 3) in the event status register to be set.

Query Errors

An error number in the range -400 to -499 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. An occurrence of any error in this class should cause the query error bit (bit 2) in the event status register to be set. An occurrence of an error also means one of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

List of Error Messages

The following table lists the error messages that can occur.

Table 22 Error Messages

Error #	Error String	Description
294	Out Of Hw Mask Resources.	The mask is too complicated to be used with hardware-accelerated mask testing. MXR/EXR-Series oscilloscopes have fewer hardware-accelerated mask test resources than UXR-Series oscilloscopes.
293	Concave Mask.	Hardware-accelerated mask testing cannot use concave polygons in the mask. Try breaking the concave polygon into multiple non-concave polygons.
273	Screen captures during Remote Desktop may not work properly.	There is a known issue (occasional crashes) when performing screen captures while the Windows operating system's Remote Desktop application is connected to the Infiniium oscilloscope. This error occurs after :DISK:SAVE:IMAGe commands and :DISPlay:DATA? queries during Remote Desktop control to warn you about the issue. (The issue can also occur when choosing **File > Copy Screen Image** in the front panel graphical user interface.) There is no issue (and no message) when VNC is used to remotely control the oscilloscope or when Remote Desktop is disconnected.
270	In order to trigger, multiscope acquisitions require the trigger source to be displayed.	
269	In order to trigger, multiscope acquisitions require an active channel per frame.	
268	In order to trigger, all frames must be connected.	
217	The Disk Save was Cancelled.	
183	Trigger is broken. Please contact a Keysight service center.	
169	Mask align failed, either no trigger or lost trigger.	The mask test mask alignment feature was not able to succeed.
168	Mask align failed, eye not well defined in waveform.	The mask test mask alignment feature was not able to succeed.
167	Mask align failed, waveform clipped or off screen.	The mask test mask alignment feature was not able to succeed.
166	Mask align failed, could not find pulse in waveform.	The mask test mask alignment feature was not able to succeed.

Table 22 Error Messages (continued)

Error #	Error String	Description
165	Non-differential transfer function is being applied to differential channels.	
164	Mask align failed because more than one channel is on.	
161	Infiniium's probe calibration data has been upgraded. The default values are being used.	
160	Mask align completed with mask failures.	The mask test mask alignment feature completed but resultant fit gives mask failures.
159	Stop mask testing to change control value.	The mask test mask alignment feature completed but eye period exceeds delta T by at least 20%.
150	Operation permitted for standard masks only.	Auto fit works for standard masks only.
143	Please turn on color grade (in the Display dialog) for this measurement.	
141	Please turn on a waveform of the type required for this measurement.	There are no valid signals for this measurement.
140	Exceeded maximum ASCII list length.	When reading in an ASCII waveform, the maximum number of points was exceeded.
136	The waveform is too large to be loaded due to memory constraints.	
130	Infiniium is unable to recover a clock. Adjust loop bandwidth.	
129	Too many vertices.	There are too many data points in a mask test polygon.
128		Mask test errors
126	Waveform is clipped: Instrument setup is unchanged.	Autoscale fails because the signal is too big to fit on the screen.
125	There is something in the header that Infiniium was not able to parse.	The oscilloscope could not parse the waveform header information correctly.
120	Execution not possible: Calibration does not match mainframe.	The calibration file on disk does not go with this oscilloscope model. The oscilloscope cannot accept a mainframe cal block input (over the SCPI interface) because it does not match the mainframe model or serial number.
117	The waveform is too large to save as the selected file type.	

Table 22 Error Messages (continued)

Error #	Error String	Description
116	The waveform received has more points than expected. The extra points were discarded.	There was an attempt to send too many points into a waveform memory over the SCPI interface.
115	Debug already in use.	
113	This directory is not valid.	
112	Unknown file type.	
111	Infiniium was unable to write to the file. The file may be read-only.	
109	Unable to read the disk: The disk might not be formatted.	
108	The disk is write protected.	
107	Infiniium could not find the destination disk.	
88	Self Test Warning: Please re-calibrate the instrument and re-run the self test.	
85	Combined filter too long to run. Please disable one or more filters.	The combined FIR Filter is too long to run.
79	Probe calibration error found! The attenuation (or gain) exceeds the limits.	
74	Internal error creating cal factor backup. SSD may not be replaced or reformatted.	
72	Infiniium has an internal error. Service is required.	Calibration factors were unable to be saved to the disk.
68	Infiniium is not calibrated! Please perform calibration before making measurements.	It is important to calibrate the oscilloscope so that it will work properly.
59	Overload detected! Infiniium changed the impedance and scale.	A channel overload condition occurred. Protective changes were automatically performed.
50	General failure to read disk	
41	Waveform data is not valid.	
40	This command can't be performed on the selected waveform.	The selected operation cannot be performed on the selected waveform.
39	The chosen function can't be performed on the selected waveform.	The selected function cannot be performed on the selected waveform.

Table 22 Error Messages (continued)

Error #	Error String	Description
38	The measurement you chose can't be performed on the selected waveform.	The selected measurement cannot be performed on the selected waveform.
31	Sorry, you can't make this measurement on the selected source.	There was an attempt to drag-and-drop a measurement on an invalid source for the measurement.
30	Infiniium is running remotely right now.	The instrument is not listening to local commands. It is listening only to remote commands.
26	This entry isn't valid. The control has been set to its default value.	This error occurs when a command tries to perform a setting that is not a valid. In this case, the control is set to its default setting.
25	This entry isn't a valid selection. The control has not been modified.	This error occurs when a command tries to perform a setting that isn't a valid.
24	Control is at its maximum value.	An out of range value was entered.
22	Control is at its minimum value.	An out of range value was entered.
15	The requested GPIB operation can't be performed.	This is a generic SCPI error that the requested operation cannot be performed.
11	Signal amplitude too small to build the Real Time Eye. Increase vertical sensitivity.	
8	This function can't be performed on a peak detect waveform.	The function requested is not permitted on peak detect waveforms.
7	Mask align failed.	The mask test mask alignment feature was not able to succeed.
6	File not compatible with destination signal type.	This happens when an internal signal is read from disk and its format is not recognized.
1	Phase noise error. Check phase noise configuration.	
0	No error	The error queue is empty. Every error in the queue has been read (:SYSTem:ERRor? query) or the queue was cleared by power-up or *CLS.
-100	Command error	This is the generic syntax error used if the oscilloscope cannot detect more specific errors.
-101	Invalid character	A syntactic element contains a character that is invalid for that type.
-102	Syntax error	An unrecognized command or data type was encountered.
-103	Invalid separator	The parser was expecting a separator and encountered an illegal character.
-104	Data type error	The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was received.

Table 22 Error Messages (continued)

Error #	Error String	Description
-105	GET not allowed	A Group Execute Trigger was received within a program message.
-108	Parameter not allowed	More parameters were received than expected for the header.
-109	Missing parameter	Fewer parameters were received than required for the header.
-112	Program mnemonic too long	The header or character data element contains more than twelve characters.
-113	Undefined header	The header is syntactically correct, but it is undefined for the oscilloscope. For example, *XYZ is not defined for the oscilloscope.
-121	Invalid character in number	An invalid character for the data type being parsed was encountered. For example, a "9" in octal data.
-123	Numeric overflow	Number is too large or too small to be represented internally.
-124	Too many digits	The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros.
-128	Numeric data not allowed	A legal numeric data element was received, but the oscilloscope does not accept one in this position for the header.
-131	Invalid suffix	The suffix does not follow the syntax described in IEEE 488.2 or the suffix is inappropriate for the oscilloscope.
-138	Suffix not allowed	A suffix was encountered after a numeric element that does not allow suffixes.
-141	Invalid character data	Either the character data element contains an invalid character or the particular element received is not valid for the header.
-144	Character data too long	
-148	Character data not allowed	A legal character data element was encountered where prohibited by the oscilloscope.
-150	String data error	This error can be generated when parsing a string data element. This particular error message is used if the oscilloscope cannot detect a more specific error.
-151	Invalid string data	A string data element was expected, but was invalid for some reason. For example, an END message was received before the terminal quote character.
-158	String data not allowed	A string data element was encountered but was not allowed by the oscilloscope at this point in parsing.
-160	Block data error	This error can be generated when parsing a block data element. This particular error message is used if the oscilloscope cannot detect a more specific error.
-161	Invalid block data	
-168	Block data not allowed	A legal block data element was encountered but was not allowed by the oscilloscope at this point in parsing.

Table 22 Error Messages (continued)

Error #	Error String	Description
-170	Expression error	This error can be generated when parsing an expression data element. It is used if the oscilloscope cannot detect a more specific error.
-171	Invalid expression	
-178	Expression data not allowed	Expression data was encountered but was not allowed by the oscilloscope at this point in parsing.
-200	Execution error	This is a generic syntax error which is used if the oscilloscope cannot detect more specific errors.
-212	Arm ignored	
-213	Init ignored	
-214	Trigger deadlock	
-215	Arm deadlock	
-220	Parameter error	
-221	Settings conflict	
-222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value is outside the legal range defined by the oscilloscope.
-223	Too much data	Indicates that a legal program data element of block, expression, or string type was received that contained more data than the oscilloscope could handle due to memory or related oscilloscope-specific requirements.
-224	Illegal parameter value	
-230	Data corrupt or stale	
-231	Data questionable	
-240	Hardware error	
-241	Hardware missing	
-250	Mass storage error	
-251	Missing mass storage	
-252	Missing media	
-253	Corrupt media	
-254	Media full	
-255	Directory full	
-256	File name not found	
-257	File name error	
-258	Media protected	

Table 22 Error Messages (continued)

Error #	Error String	Description
-260	Expression error	
-261	Math error in expression	
-300	Device specific error	
-310	System error	Indicates that a system error occurred.
-311	Memory error	
-312	PUD memory error	
-313	Calibration memory lost	
-314	Save/recall memory lost	
-315	Configuration memory lost	
-321	Out of memory	
-330	Self-test failed	
-350	Queue overflow	Indicates that there is no room in the error queue and an error occurred but was not recorded.
-370	No sub tests are defined for the selected self test	
-371	Self Test status is corrupt or no self test has been executed	
-372	This product configuration does not support the requested self test	
-373	This product configuration does not support the requested source	
-374	The requested self test log file could not be found	
-375	Attenuator relay actuation counts can only be modified during factory service	
-400	Query error	This is the generic query error.
-410	Query INTERRUPTED	
-420	Query UNTERMINATED	
-430	Query DEADLOCKED	
-440	Query UNTERMINATED after indefinite response	

52 Example Programs

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

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:
      - VISA COM 5.11 Type Library
      - Microsoft Scripting Runtime
   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 Infiniium Series oscilloscope.
' -------------------------------------------------------------------

Option Explicit

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

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' 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("TCPIP0::141.121.237.226::hislip0::INSTR")
  myScope.IO.Timeout = 15000   ' Set I/O communication timeout.
  myScope.IO.Clear   ' Clear the interface.

  ' 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

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  Debug.Print "Identification string: " + strQueryResult

  ' Load the default setup.
  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

  ' Set probe attenuation factor.
  'DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically set up oscilloscope.
  ' ---------------------------------------------------------------
  Debug.Print "Autoscale."
  DoCommand ":AUToscale"

  ' Set trigger mode.
  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:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope setup.
  ' ---------------------------------------------------------------
  varQueryResult = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

  ' Output setup string to a file:
  Dim strPath As String
  strPath = "c:\scope\config\setup.dat"
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Put hFile, , varQueryResult   ' Write data.
  Close hFile   ' Close file.
  Debug.Print "Setup bytes saved: " + CStr(LenB(varQueryResult))

  ' Change oscilloscope settings with individual commands:
  ' ---------------------------------------------------------------

  ' Set vertical scale and offset.
  DoCommand ":CHANnel1:SCALe 0.1"
  Debug.Print "Channel 1 vertical scale: " + _
      DoQueryString(":CHANnel1:SCALe?")
```

```
  DoCommand ":CHANnel1:OFFSet 0.0"
  Debug.Print "Channel 1 vertical offset: " + _
      DoQueryString(":CHANnel1:OFFSet?")

  ' Set horizontal scale and offset.
  DoCommand ":TIMebase:SCALe 200E-6"
  Debug.Print "Timebase scale: " + _
      DoQueryString(":TIMebase:SCALe?")

  DoCommand ":TIMebase:POSition 0.0"
  Debug.Print "Timebase position: " + _
      DoQueryString(":TIMebase:POSition?")

  ' Set the acquisition mode.
  DoCommand ":ACQuire:MODE RTIMe"
  Debug.Print "Acquire mode: " + _
      DoQueryString(":ACQuire:MODE?")

  ' Or, configure by loading a previously saved setup.
  ' -----------------------------------------------------------------
  Dim varSetupString As Variant
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As hFile   ' Open file for input.
  Get hFile, , varSetupString   ' Read data.
  Close hFile   ' Close file.
  ' Write learn string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  DoCommandIEEEBlock ":SYSTem:SETup", varSetupString
  Debug.Print "Setup bytes restored: " + CStr(LenB(varSetupString))

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' -----------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

  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 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 byteData() As Byte
byteData = 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, , byteData   ' Write data.
Close hFile   ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
    " bytes) written to " + strPath


' Download waveform data.
' -----------------------------------------------------------------

' Get the waveform type.
Debug.Print "Waveform type: " + _
    DoQueryString(":WAVeform:TYPE?")

' Get the number of waveform points.
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:
DoCommand ":WAVeform:FORMat WORD"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings from preamble:
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 intCoupling As Integer
Dim dblXDispRange As Double
Dim dblXDispOrigin As Double
Dim dblYDispRange As Double
Dim dblYDispOrigin As Double
Dim strDate As String
Dim strTime As String
Dim strFrameModel As String
Dim intAcqMode As Integer
Dim intCompletion As Integer
Dim intXUnits As Integer
Dim intYUnits As Integer
Dim dblMaxBwLimit As Double
Dim dblMinBwLimit As Double

Dim dctWavFormat As Scripting.Dictionary
Set dctWavFormat = New Scripting.Dictionary
dctWavFormat.Add 0, "ASCii"
dctWavFormat.Add 1, "BYTE"
dctWavFormat.Add 2, "WORD"
dctWavFormat.Add 3, "LONG"
dctWavFormat.Add 4, "LONGLONG"

Dim dctAcqType As Scripting.Dictionary
Set dctAcqType = New Scripting.Dictionary
dctAcqType.Add 1, "RAW"
dctAcqType.Add 2, "AVERage"
dctAcqType.Add 3, "VHIStogram"
dctAcqType.Add 4, "HHIStogram"
dctAcqType.Add 6, "INTerpolate"
dctAcqType.Add 10, "PDETect"

Dim dctAcqMode As Scripting.Dictionary
Set dctAcqMode = New Scripting.Dictionary
dctAcqMode.Add 0, "RTIMe"
dctAcqMode.Add 1, "ETIMe"
dctAcqMode.Add 3, "PDETect"

Dim dctCoupling As Scripting.Dictionary
Set dctCoupling = New Scripting.Dictionary
dctCoupling.Add 0, "AC"
dctCoupling.Add 1, "DC"
dctCoupling.Add 2, "DCFIFTY"
dctCoupling.Add 3, "LFREJECT"

Dim dctUnits As Scripting.Dictionary
Set dctUnits = New Scripting.Dictionary
```

```
dctUnits.Add 0, "UNKNOWN"
dctUnits.Add 1, "VOLT"
dctUnits.Add 2, "SECOND"
dctUnits.Add 3, "CONSTANT"
dctUnits.Add 4, "AMP"
dctUnits.Add 5, "DECIBEL"

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)
intCoupling = Preamble(10)
dblXDispRange = Preamble(11)
dblXDispOrigin = Preamble(12)
dblYDispRange = Preamble(13)
dblYDispOrigin = Preamble(14)
strDate = Preamble(15)
strTime = Preamble(16)
strFrameModel = Preamble(17)
intAcqMode = Preamble(18)
intCompletion = Preamble(19)
intXUnits = Preamble(20)
intYUnits = Preamble(21)
dblMaxBwLimit = Preamble(22)
dblMinBwLimit = Preamble(23)

Debug.Print "Waveform format: " + dctWavFormat.Item(intFormat)
Debug.Print "Acquisition type: " + dctAcqType.Item(intType)

Debug.Print "Waveform points desired: " + _
    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)

Debug.Print "Coupling: " + dctCoupling.Item(intCoupling)

Debug.Print "Waveform X display range: " + _
    Format(dblXDispRange, "Scientific")

Debug.Print "Waveform X display origin: " + _
    Format(dblXDispOrigin, "Scientific")

Debug.Print "Waveform Y display range: " + _
    Format(dblYDispRange, "Scientific")

Debug.Print "Waveform Y display origin: " + _
    Format(dblYDispOrigin, "Scientific")

Debug.Print "Date: " + strDate
Debug.Print "Time: " + strTime
Debug.Print "Frame model: " + strFrameModel
Debug.Print "Acquire mode: " + dctAcqMode.Item(intAcqMode)

Debug.Print "Completion pct: " + _
    FormatNumber(intCompletion, 0)

Debug.Print "Waveform X units: " + dctUnits.Item(intXUnits)
Debug.Print "Waveform Y units: " + dctUnits.Item(intYUnits)

Debug.Print "Max BW limit: " + _
    Format(dblMaxBwLimit, "Scientific")

Debug.Print "Min BW limit: " + _
    Format(dblMinBwLimit, "Scientific")

' Get the waveform data.
DoCommand ":WAVeform:STReaming OFF"
varQueryResult = DoQueryIEEEBlock_I2(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(UBound(varQueryResult) + 1)

' 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)
  lngDataValue = varQueryResult(lngI)

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
```

```
        ", " + _
        FormatNumber((lngDataValue * 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 Function DoQueryIEEEBlock_I2(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryIEEEBlock_I2 = myScope.ReadIEEEBlock(BinaryType_I2)
  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? STRing"  ' 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? STRing" ' 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 2013:

1 Open Visual Studio.
2 Choose **FILE > New > Project...**.
3 Create a new Visual C#, Console Application project.
4 Cut-and-paste the code that follows into the C# source file.
5 Edit the program to use the VISA address of your oscilloscope.
6 Add a reference to the VISA COM 5.11 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, under **COM**, select **Type Libraries**.
    d Select **VISA COM 5.11 Type Library**; then click **OK**.
    e In the Solution Explorer, with the References node expanded, select the **VisaComLib** reference you just added. In the Properties window, set the value of the **Embed Interop Types** property 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 Infiniium Series oscilloscope.
 * -------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace Infiniium
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
```

```
{
  try
  {
    myScope = new
      VisaComInstrument("TCPIP0::141.121.237.226::hislip0::INSTR");
    myScope.SetTimeoutSeconds(10);

    // Initialize - start from a known state.
    Initialize();

    // Capture data.
    Capture();

    // Analyze the captured waveform.
    Analyze();

    Console.WriteLine("Press any key to exit");
    Console.ReadKey();
  }
  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;

  // Clear status.
  myScope.DoCommand("*CLS");

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Load the default setup.
  myScope.DoCommand("*RST");
}
```

```
/*
 * Capture the waveform.
 * --------------------------------------------------------------
 */
private static void Capture()
{
  // Set probe attenuation factor.
  //myScope.DoCommand(":CHANnel1:PROBe 1.0");
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      myScope.DoQueryString(":CHANnel1:PROBe?"));

  // Use auto-scale to automatically set up oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode.
  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:LEVel CHANnel1,-2E-3");
  Console.WriteLine("Trigger level, channel 1: {0}",
      myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"));

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

  // Save oscilloscope setup.
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Query and read setup string.
  ResultsArray = myScope.DoQueryIEEEBlock_UI1(":SYSTem:SETup?");
  nLength = ResultsArray.Length;

  // Write setup string to file.
  strPath = "c:\\scope\\config\\setup.stp";
  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 0.1");
  Console.WriteLine("Channel 1 vertical scale: {0}",
      myScope.DoQueryString(":CHANnel1:SCALe?"));

  myScope.DoCommand(":CHANnel1:OFFSet 0.0");
  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 mode.
    myScope.DoCommand(":ACQuire:MODE RTIMe");
    Console.WriteLine("Acquire mode: {0}",
        myScope.DoQueryString(":ACQuire:MODE?"));

    // Or, configure by loading a previously saved setup.
    byte[] DataArray;
    int nBytesWritten;

    // Read setup string from file.
    strPath = "c:\\scope\\config\\setup.stp";
    DataArray = File.ReadAllBytes(strPath);
    nBytesWritten = DataArray.Length;

    // Restore setup string.
    myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
    Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

    // Set the desired number of waveform points,
    // and capture an acquisition.
    myScope.DoCommand(":ACQuire:POINts 32000");
    myScope.DoCommand(":DIGitize");
  }

  /*
   * Analyze the captured waveform.
   * --------------------------------------------------------------
   */
  private static void Analyze()
  {
    byte[] ResultsArray;   // Results array.
    int nLength;   // Number of bytes returned from instrument.
    string strPath;

    // Make 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_UI1(":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.
// -----------------------------------------------------------

// Get the waveform points mode.
Console.WriteLine("Waveform type: {0}",
    myScope.DoQueryString(":WAVeform:TYPE?"));

// Get the number of waveform points.
Console.WriteLine("Waveform points: {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:
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
```

```
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
  new Dictionary<string, string>()
{
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

strPreamble = myScope.DoQueryString(":WAVeform:PREamble?");
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
```

```
    Console.WriteLine("Date: {0}", strsPreamble[15]);
    Console.WriteLine("Time: {0}", strsPreamble[16]);
    Console.WriteLine("Frame model: {0}", strsPreamble[17]);
    Console.WriteLine("Acquire mode: {0}",
        dctAcqMode[strsPreamble[18]]);
    Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
    Console.WriteLine("Waveform X inits: {0}",
        dctUnits[strsPreamble[20]]);
    Console.WriteLine("Waveform Y units: {0}",
        dctUnits[strsPreamble[21]]);
    Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
    Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

    // Get numeric values for later calculations.
    double fXincrement;
    fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?");
    double fXorigin;
    fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?");
    double fYincrement;
    fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?");
    double fYorigin;
    fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?");

    // Get the waveform data.
    myScope.DoCommand(":WAVeform:STReaming OFF");
    short[] WordDataArray;   // Results array.
    WordDataArray = myScope.DoQueryIEEEBlock_I2(":WAVeform:DATA?");
    nLength = WordDataArray.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)WordDataArray[i])
          * 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_UI1(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;
  }

  public short[] DoQueryIEEEBlock_I2(string strQuery)
  {
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the results array.
    System.Threading.Thread.Sleep(2000); // Delay before reading.
    short[] ResultsArray;
    ResultsArray = (short[])m_IoObject.ReadIEEEBlock(
      IEEEBinaryType.BinaryType_I2, 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? STRing", 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 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. Create a new Visual Basic, Console Application project.
4. Cut-and-paste the code that follows into the Visual Basic .NET source file.
5. Edit the program to use the VISA address of your oscilloscope.
6. Add a reference to the VISA COM 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, under **COM**, select **Type Libraries**.
   d. Select **VISA COM 5.11 Type Library**; then click **OK**.
7. Specify the Startup object and set Embed Interop Types to false:
   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. In the Properties dialog box, under **References**, select the **VISA COM 5.11 Type Library** reference. In the Properties window, set the value of the **Embed Interop Types** property to **False**.
   e. Save your change and close the Properties dialog box.
8. 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 Infiniium Series oscilloscope.
' -------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports System.Collections.Generic
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace Infiniium
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New  _
          VisaComInstrument("TCPIP0::141.121.237.226::hislip0::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

      ' Clear status.
      myScope.DoCommand("*CLS")
```

```
    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Load the default setup.
    myScope.DoCommand("*RST")

  End Sub

  ' Capture the waveform.
  ' --------------------------------------------------------------

  Private Shared Sub Capture()

    ' Set probe attenuation factor.
    'myScope.DoCommand(":CHANnel1:PROBe 1.0")
    Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
        myScope.DoQueryString(":CHANnel1:PROBe?"))

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode.
    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:LEVel CHANnel1,-2E-3")
    Console.WriteLine("Trigger level, channel 1: {0}", _
        myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"))

    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_UI1(":SYSTem:SETup?")
    nLength = ResultsArray.Length

    ' Write setup string to file.
    strPath = "c:\scope\config\setup.stp"
    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 0.1")
  Console.WriteLine("Channel 1 vertical scale: {0}", _
      myScope.DoQueryString(":CHANnel1:SCALe?"))

  myScope.DoCommand(":CHANnel1:OFFSet 0.0")
  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 mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe")
  Console.WriteLine("Acquire mode: {0}", _
      myScope.DoQueryString(":ACQuire:MODE?"))

  ' 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.stp"
  DataArray = File.ReadAllBytes(strPath)
  nBytesWritten = DataArray.Length

  ' Restore setup string.
  myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000")
  myScope.DoCommand(":DIGitize")

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 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_UI1(":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.
' -------------------------------------------------------------

' Get the waveform type.
Console.WriteLine("Waveform type: {0}", _
    myScope.DoQueryString(":WAVeform:TYPE?"))

' Get the number of waveform points.
Console.WriteLine("Waveform points: {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:
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")
```

```
Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

strPreamble = myScope.DoQueryString(":WAVeform:PREamble?")
strsPreamble = strPreamble.Split(","c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))

Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
Console.WriteLine("Waveform Y display range: {0}", _
    strsPreamble(13))
Console.WriteLine("Waveform Y display origin: {0}", _
    strsPreamble(14))
Console.WriteLine("Date: {0}", strsPreamble(15))
```

```
    Console.WriteLine("Time: {0}", strsPreamble(16))
    Console.WriteLine("Frame model: {0}", strsPreamble(17))
    Console.WriteLine("Acquire mode: {0}", _
        dctAcqMode(strsPreamble(18)))
    Console.WriteLine("Completion pct: {0}", strsPreamble(19))
    Console.WriteLine("Waveform X inits: {0}", _
        dctUnits(strsPreamble(20)))
    Console.WriteLine("Waveform Y units: {0}", _
        dctUnits(strsPreamble(21)))
    Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
    Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

    ' Get numeric values for later calculations.
    Dim fXincrement As Double
    fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?")
    Dim fXorigin As Double
    fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?")
    Dim fYincrement As Double
    fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?")
    Dim fYorigin As Double
    fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?")

    ' Get the waveform data.
    myScope.DoCommand(":WAVeform:STReaming OFF")
    Dim WordDataArray As Short()
    WordDataArray = myScope.DoQueryIEEEBlock_I2(":WAVeform:DATA?")
    nLength = WordDataArray.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 index As Integer = 0 To nLength - 1
      ' Write time value, voltage value.
      writer.WriteLine("{0:f9}, {1:f6}", _
          fXorigin + (CSng(index) * fXincrement), _
          (CSng(WordDataArray(index)) * 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_UI1(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

Public _
    Function _
        DoQueryIEEEBlock_I2(ByVal strQuery As String) As Short()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the results array.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
```

```
  Dim ResultsArray As Short()
  ResultsArray = _
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_I2, _
      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? STRing", 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 3

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
#
# Keysight VISA COM Example in Python using "comtypes"
# *********************************************************
# This program illustrates a few commonly used programming
# features of your Keysight Infiniium Series oscilloscope.
# *********************************************************

# Import Python modules.
# ---------------------------------------------------------
import string
import time
import sys
import array

from comtypes.client import GetModule
from comtypes.client import CreateObject
from comtypes.automation import VARIANT

# 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 (booleans: 0 = False, 1 = True).
# ---------------------------------------------------------


# =========================================================
# Initialize:
# =========================================================
def initialize():
 # Get and display the device's *IDN? string.
 idn_string = do_query_string("*IDN?")
 print("Identification string '%s'" % idn_string)

 # Clear status and load the default setup.
 do_command("*CLS")
 do_command("*RST")


# =========================================================
# Capture:
# =========================================================
def capture():

 # Set probe attenuation factor.
 #do_command(":CHANnel1:PROBe 1.0")
 qresult = do_query_string(":CHANnel1:PROBe?")
 print("Channel 1 probe attenuation factor: %s" % qresult)

 # 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("Trigger mode: %s" % qresult)

 # Set EDGE trigger parameters.
 do_command(":TRIGger:EDGE:SOURce CHANnel1")
 qresult = do_query_string(":TRIGger:EDGE:SOURce?")
 print("Trigger edge source: %s" % qresult)

 do_command(":TRIGger:LEVel CHANnel1,-2E-3")
 qresult = do_query_string(":TRIGger:LEVel? CHANnel1")
 print("Trigger level, channel 1: %s" % qresult)

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print("Trigger edge slope: %s" % qresult)

 # Save oscilloscope setup.
 setup_bytes = do_query_ieee_block_UI1(":SYSTem:SETup?")
 nLength = len(setup_bytes)
 f = open("setup.stp", "wb")
 f.write(bytearray(setup_bytes))
 f.close()
 print("Setup bytes saved: %d" % nLength)
```

```
 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(":CHANnel1:SCALe 0.1")
 qresult = do_query_number(":CHANnel1:SCALe?")
 print("Channel 1 vertical scale: %f" % qresult)

 do_command(":CHANnel1:OFFSet 0.0")
 qresult = do_query_number(":CHANnel1:OFFSet?")
 print("Channel 1 offset: %f" % qresult)

 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 200e-6")
 qresult = do_query_string(":TIMebase:SCALe?")
 print("Timebase scale: %s" % qresult)

 do_command(":TIMebase:POSition 0.0")
 qresult = do_query_string(":TIMebase:POSition?")
 print("Timebase position: %s" % qresult)

 # Set the acquisition mode.
 do_command(":ACQuire:MODE RTIMe")
 qresult = do_query_string(":ACQuire:MODE?")
 print("Acquire mode: %s" % qresult)

 # Or, configure by loading a previously saved setup.
 f = open("setup.stp", "rb")
 setup_bytes = f.read()
 f.close()
 do_command_ieee_block(":SYSTem:SETup", array.array('B', setup_bytes))
 print("Setup bytes restored: %d" % len(setup_bytes))

 # Set the desired number of waveform points,
 # and capture an acquisition.
 do_command(":ACQuire:POINts 32000")
 do_command(":DIGitize")


# ==========================================================
# Analyze:
# ==========================================================
def analyze():

 # Make measurements.
 # ---------------------------------------------------------
 do_command(":MEASure:SOURce CHANnel1")
 qresult = do_query_string(":MEASure:SOURce?")
 print("Measure source: %s" % qresult)

 do_command(":MEASure:FREQuency")
 qresult = do_query_string(":MEASure:FREQuency?")
 print("Measured frequency on channel 1: %s" % qresult)

 do_command(":MEASure:VAMPlitude")
 qresult = do_query_string(":MEASure:VAMPlitude?")
 print("Measured vertical amplitude on channel 1: %s" % qresult)
```

```
# Download the screen image.
# --------------------------------------------------------
image_bytes = do_query_ieee_block_UI1(":DISPlay:DATA? PNG")
nLength = len(image_bytes)
f = open("screen_image.png", "wb")
f.write(bytearray(image_bytes))
f.close()
print("Screen image written to 'screen_image.png'.")

# Download waveform data.
# --------------------------------------------------------

# Get the waveform type.
qresult = do_query_string(":WAVeform:TYPE?")
print("Waveform type: %s" % qresult)

# Get the number of waveform points.
qresult = do_query_string(":WAVeform:POINts?")
print("Waveform points: %s" % qresult)

# Set the waveform source.
do_command(":WAVeform:SOURce CHANnel1")
qresult = do_query_string(":WAVeform:SOURce?")
print("Waveform source: %s" % qresult)

# Choose the format of the data returned:
do_command(":WAVeform:FORMat WORD")
print("Waveform format: %s" % do_query_string(":WAVeform:FORMat?"))

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
```

```
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
 5 : "DECIBEL",
}

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, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = preamble_string.split(",")

print("Waveform format: %s" % wav_form_dict[int(wav_form)])
print("Acquire type: %s" % acq_type_dict[int(acq_type)])
print("Waveform points desired: %s" % wfmpts)
print("Waveform average count: %s" % avgcnt)
print("Waveform X increment: %s" % x_increment)
print("Waveform X origin: %s" % x_origin)
print("Waveform X reference: %s" % x_reference)   # Always 0.
print("Waveform Y increment: %s" % y_increment)
print("Waveform Y origin: %s" % y_origin)
print("Waveform Y reference: %s" % y_reference)   # Always 0.
print("Coupling: %s" % coupling_dict[int(coupling)])
print("Waveform X display range: %s" % x_display_range)
print("Waveform X display origin: %s" % x_display_origin)
print("Waveform Y display range: %s" % y_display_range)
print("Waveform Y display origin: %s" % y_display_origin)
print("Date: %s" % date)
print("Time: %s" % time)
print("Frame model #: %s" % frame_model)
print("Acquire mode: %s" % acq_mode_dict[int(acq_mode)])
print("Completion pct: %s" % completion)
print("Waveform X units: %s" % units_dict[int(x_units)])
print("Waveform Y units: %s" % units_dict[int(y_units)])
print("Max BW limit: %s" % max_bw_limit)
print("Min BW limit: %s" % min_bw_limit)

# 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?")

# Get the waveform data.
do_command(":WAVeform:STReaming OFF")
data_words = do_query_ieee_block_I2(":WAVeform:DATA?")
nLength = len(data_words)
print("Number of data values: %d" % nLength)

# Open file for output.
strPath = "waveform_data.csv"
```

```
 f = open(strPath, "w")

 # Output waveform data in CSV format.
 for i in range(0, nLength - 1):
  time_val = x_origin + (i * x_increment)
  voltage = (data_words[i] * y_increment) + y_origin
  f.write("%E, %f\n" % (time_val, voltage))

 # Close output file.
 f.close()
 print("Waveform format WORD data written to %s." % strPath)


# =========================================================
# Send a command and check for errors:
# =========================================================
def do_command(command):
 myScope.WriteString("%s" % command, True)
 check_instrument_errors(command)


# =========================================================
# Send a command and check for errors:
# =========================================================
def do_command_ieee_block(command, data):
 myScope.WriteIEEEBlock(command, VARIANT(array.array('B', data)), True)
 check_instrument_errors(command)


# =========================================================
# Send a query, check for errors, return string:
# =========================================================
def do_query_string(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadString()
 check_instrument_errors(query)
 return result


# =========================================================
# Send a query, check for errors, return string:
# =========================================================
def do_query_ieee_block_UI1(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_UI1, \
   False, True)
 check_instrument_errors(query)
 return result


# =========================================================
# Send a query, check for errors, return string:
# =========================================================
def do_query_ieee_block_I2(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_I2, \
   False, True)
```

```
 check_instrument_errors(query)
 return result


# =========================================================
# Send a query, check for errors, return values:
# =========================================================
def do_query_number(query):
 myScope.WriteString("%s" % 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("%s" % 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? STRing", True)
  error_string = myScope.ReadString()
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    print("ERROR: %s, command: '%s'" % (error_string, command))
    print("Exited because of error.")
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print("ERROR: :SYSTem:ERRor? STRing 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("TCPIP0::141.121.231.13::hislip0::INSTR")

# 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 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. In the New Project dialog box, create a new Visual C++, Win32 Console Application project.
4. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
5. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
6. In Visual Studio 2013, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
7. Edit the program to use the VISA address of your oscilloscope.
8. Choose **Project > Properties...**. In the Property Pages dialog, 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, select Preprocessor Definitions and add _CRT_SECURE_NO_WARNINGS.
   d. Under Configuration Properties, VC++ Directories, select Include Directories and add the include directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\Include).
   e. Under Configuration Properties, VC++ Directories, select Library Directories and add the include directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\Include).
   f. Click **OK** to close the Property Pages dialog.
9. Build and run the program.

```
/*
 * Keysight VISA Example in C
 * ------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight Infiniium Series 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 "TCPIP0::141.121.237.226::hislip0::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 byte block. */
int do_query_ieeeblock_words(char *query);   /* Query for word 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(). */
signed short ieeeblock_data_words[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock_words(). */
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();
```

```
  /* Clear the interface. */
  err = viClear(vi);
  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 status. */
  do_command("*CLS");

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Load the default setup. */
  do_command("*RST");
}

/* Capture the waveform.
 * -------------------------------------------------------------- */
void capture (void)
{
  int num_values;
  FILE *fp;

  /* Set probe attenuation factor. */
  /* do_command(":CHANnel1:PROBe 1.0"); */
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope. */
  do_command(":AUToscale");

  /* Set trigger mode. */
  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:LEVel CHANnel1,-2E-3");
do_query_string(":TRIGger:LEVel? CHANnel1");
printf("Trigger level, channel 1: %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 setup. */

/* Read system setup. */
num_values = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_values);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.1");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet 0.0");
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 mode. */
do_command(":ACQuire:MODE RTIMe");
do_query_string(":ACQuire:MODE?");
printf("Acquire mode: %s\n", str_result);

/* Or, set up by loading a previously saved setup. */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_values = fread (ieeeblock_data, sizeof(unsigned char),
  IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");
```

```
  /* Restore setup string. */
  num_values = do_command_ieeeblock(":SYSTem:SETup", num_values);
  printf("Restored setup string (%d bytes).\n", num_values);

  /* Set the desired number of waveform points,
   * and capture an acquisition. */
  do_command(":ACQuire:POINts 32000");
  do_command(":DIGitize");
}

/* 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 y_increment;
  double y_origin;

  FILE *fp;
  int num_values;   /* Number of bytes returned from instrument. */
  int i;

  /* Make 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_values = do_query_ieeeblock(":DISPlay:DATA? PNG");
  printf("Screen image bytes: %d\n", num_values);

  /* Write screen image bytes to file. */
  fp = fopen ("c:\\scope\\data\\screen.png", "wb");
  num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
    fp);
  fclose (fp);
  printf("Wrote screen image (%d bytes) to ", num_values);
  printf("c:\\scope\\data\\screen.bmp.\n");
```

```
/* Download waveform data.
 * ------------------------------------------------------------ */

/* Get the waveform type. */
do_query_string(":WAVeform:TYPE?");
printf("Waveform type: %s\n", str_result);

/* Get the number of waveform points. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points: %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: */
do_command(":WAVeform:FORMat WORD");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_number(":WAVeform:XINCrement?");
x_increment = num_result;
printf("Waveform X increment: %e\n", x_increment);

do_query_number(":WAVeform:XORigin?");
x_origin = num_result;
printf("Waveform X origin: %e\n", x_origin);

do_query_number(":WAVeform:YINCrement?");
y_increment = num_result;
printf("Waveform Y increment: %e\n", y_increment);

do_query_number(":WAVeform:YORigin?");
y_origin = num_result;
printf("Waveform Y origin: %e\n", y_origin);

/* Read waveform data. */
num_values = do_query_ieeeblock_words(":WAVeform:DATA?");
printf("Number of data values: %d\n", num_values);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

/* Output waveform data in CSV format. */
for (i = 0; i < num_values - 1; i++)
{
  /* Write time value, voltage value. */
  fprintf(fp, "%9f, %6f\n",
    x_origin + ((float)i * x_increment),
    ((float)ieeeblock_data_words[i] * 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 byte 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);
}

/* Query for an IEEE definite-length word block result.
 * --------------------------------------------------------------- */
int do_query_ieeeblock_words(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, "%#hb\n", &data_length, ieeeblock_data_words);
  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? STRing\n", "%t", str_err_val);
  if (err != VI_SUCCESS) error_handler();
  while(strncmp(str_err_val, "0,", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    err = viQueryf(vi, ":SYSTem:ERRor? STRing\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 Visual Basic

To run this example in Visual Basic for Applications:

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. Add the visa32.bas file to your project:
   a. Choose **File > Import File...**.
   b. Navigate to the header file, visa32.bas (installed with Keysight IO Libraries Suite and found in the Program Files (x86)\IVI Foundation\VISA\WinNT\Include), select it, and click **Open**.
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 Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight Infiniium Series oscilloscope.
' -------------------------------------------------------------------

Option Explicit
```

```
Public err As Long    ' Error returned by VISA function calls.
Public drm As Long    ' Session to Default Resource Manager.
Public vi As Long     ' Session to instrument.

' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public Const WordArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public wordArray(WordArraySize) As Integer
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' --------------------------------------------------------------------

Sub Main()

  ' Open the default resource manager session.
  err = viOpenDefaultRM(drm)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Open the session using the oscilloscope's VISA address.
  err = viOpen(drm, _
      "TCPIP0::141.121.237.226::hislip0::INSTR", 0, 15000, vi)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Set the I/O timeout to ten seconds.
  err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 10000)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Clear the interface.
  err = viClear(vi)
  If Not (err = VI_SUCCESS) Then HandleVISAError vi

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  err = viClose(vi)
```

```
  err = viClose(drm)

End Sub

'
' Initialize the oscilloscope to a known state.
' --------------------------------------------------------------------

Private Sub Initialize()

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

  ' Load the default setup.
  DoCommand "*RST"

End Sub

'
' Capture the waveform.
' --------------------------------------------------------------------

Private Sub Capture()

  ' Set probe attenuation factor.
  'DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' 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:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' ------------------------------------------------------------------
```

```
Dim lngSetupStringSize As Long
lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
If Len(Dir(strPath)) Then
  Kill strPath   ' Remove file if it exists.
End If

' Open file for output.
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngSetupStringSize - 1
  Put hFile, , byteArray(lngI)   ' Write data.
Next lngI
Close hFile   ' Close file.

' Change settings with individual commands:
' -----------------------------------------------------------------

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.1"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet 0.0"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and position.
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 mode.
DoCommand ":ACQuire:MODE RTIMe"
Debug.Print "Acquire mode: " + _
    DoQueryString(":ACQuire:MODE?")

' Or, configure by loading a previously saved setup.
' -----------------------------------------------------------------
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile   ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile)  ' Length of file.
Get hFile, , byteArray   ' Read data.
Close hFile   ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
```

```
  Dim lngRestored As Long
  lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' -----------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

End Sub

'
' Analyze the captured waveform.
' -------------------------------------------------------------------

Private Sub Analyze()

  ' Make a couple of measurements.
  ' -----------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"


  ' Download the screen image.
  ' -----------------------------------------------------------------

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' 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
  Dim lngI As Long
  For lngI = 0 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
```

```
MsgBox "Screen image written to " + strPath


' Download waveform data.
' ------------------------------------------------------------------

' Get the waveform type.
Debug.Print "Waveform type: " + _
    DoQueryString(":WAVeform:TYPE?")

' Get the number of waveform points.
Debug.Print "Waveform points: " + _
    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:
DoCommand ":WAVeform:FORMat WORD"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double

dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment: " + _
    Format(dblYIncrement, "Scientific")

dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin: " + _
    FormatNumber(dblYOrigin, 0)

' Get the waveform data
DoCommand ":WAVeform:STReaming OFF"
Dim lngNumWords As Long
lngNumWords = DoQueryIEEEBlock_Words(":WAVeform:DATA?")
Debug.Print "Number of data values: " + CStr(lngNumWords)

' 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.
  For lngI = 0 To lngNumWords - 1

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber((wordArray(lngI) * dblYIncrement) + dblYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format WORD data written to " + _
      "c:\scope\data\waveform_data.csv."

End Sub

Private Sub DoCommand(command As String)

  err = viVPrintf(vi, command + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  CheckInstrumentErrors

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  retCount = lngBlockSize

  Dim strCommandAndLength As String
  strCommandAndLength = command + " %#" + _
      Format(lngBlockSize) + "b"

  err = viVPrintf(vi, strCommandAndLength + vbLf, paramsArray(1))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoCommandIEEEBlock = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryString(query As String) As String

  Dim strResult As String * 200

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strResult)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoQueryString = strResult
```

```
  CheckInstrumentErrors

End Function

Private Function DoQueryNumber(query As String) As Variant

  Dim dblResult As Double

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblResult))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoQueryNumber = dblResult

  CheckInstrumentErrors

End Function

Private Function DoQueryNumbers(query As String) As Long

  Dim dblResult As Double

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(dblArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = DblArraySize

  ' Read numbers.
  err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of values returned by query.
  DoQueryNumbers = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(byteArray(0))
```

```
  ' Set retCount to max number of elements array can hold.
  retCount = ByteArraySize

  ' Get unsigned integer bytes.
  err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_READ_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_WRITE_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of bytes returned by query.
  DoQueryIEEEBlock_Bytes = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Words(query As String) As Long

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(wordArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = WordArraySize

  ' Get signed integer words.
  err = viVScanf(vi, "%#hb" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_READ_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_WRITE_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of bytes returned by query.
  DoQueryIEEEBlock_Words = retCount

  CheckInstrumentErrors

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String
```

```
  err = viVPrintf(vi, ":SYSTem:ERRor? STRing" + vbLf, 0) ' Query any err
ors.
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strErrVal)   ' Read: Errnum,"Error String".
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  While Val(strErrVal) <> 0             ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal

    err = viVPrintf(vi, ":SYSTem:ERRor? STRing" + vbLf, 0) ' Request err
or.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strErrVal)   ' Read error message.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"

    err = viFlush(vi, VI_READ_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_WRITE_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub HandleVISAError(session As Long)

  Dim strVisaErr As String * 200
  Call viStatusDesc(session, err, strVisaErr)
  MsgBox "*** VISA Error : " + strVisaErr, vbExclamation

  ' If the error is not a warning, close the session.
  If err < VI_SUCCESS Then
    If session <> 0 Then Call viClose(session)
    End
  End If

End Sub
```

VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2013:

1 Open Visual Studio.

2 Choose **FILE > New > Project...**.

3 Create a new Visual C#, Console Application project.

4 Cut-and-paste the code that follows into the C# source file.

5 Edit the program to use the VISA address of your oscilloscope.

6 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 Click **Add** and then click **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.

7 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with the Keysight IO Libraries Suite.

```
/*
 * Keysight VISA Example in C#
 * -------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight Infiniium Series oscilloscope.
 * -------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;

namespace Infiniium
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
```

```
        VisaInstrument("TCPIP0::141.121.237.226::hislip0::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;

    // Clear status.
    myScope.DoCommand("*CLS");

    // Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?");
    Console.WriteLine("*IDN? result is: {0}", strResults);

    // Load the default setup.
    myScope.DoCommand("*RST");
  }

  /*
   * Capture the waveform.
   * --------------------------------------------------------------
   */
  private static void Capture()
  {
```

```
// Set probe attenuation factor.
//myScope.DoCommand(":CHANnel1:PROBe 1.0");
Console.WriteLine("Channel 1 probe attenuation factor: {0}",
    myScope.DoQueryString(":CHANnel1:PROBe?"));

// Use auto-scale to automatically set up oscilloscope.
myScope.DoCommand(":AUToscale");

// Set trigger mode.
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:LEVel CHANnel1,-2E-3");
Console.WriteLine("Trigger level, channel 1: {0}",
    myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"));

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_Bytes(":SYSTem:SETup?",
  out ResultsArray);

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
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 0.1");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet 0.0");
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 mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe");
  Console.WriteLine("Acquire mode: {0}",
      myScope.DoQueryString(":ACQuire:MODE?"));

  // Or, set up by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);

  // Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
    DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Set the desired number of waveform points,
  // and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000");
  myScope.DoCommand(":DIGitize");
}

/*
 * Analyze the captured waveform.
 * ----------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  short[] WordResultsArray;   // Results array for WORD data.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make 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_Bytes(":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.
// -------------------------------------------------------------

// Get the waveform type.
Console.WriteLine("Waveform type: {0}",
    myScope.DoQueryString(":WAVeform:TYPE?"));

// Get the number of waveform points.
Console.WriteLine("Waveform points: {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:
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
  new Dictionary<string, string>()
{
```

```
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

strPreamble =
    myScope.DoQueryString(":WAVeform:PREamble?").ToString();
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
Console.WriteLine("Date: {0}", strsPreamble[15]);
Console.WriteLine("Time: {0}", strsPreamble[16]);
Console.WriteLine("Frame model: {0}", strsPreamble[17]);
Console.WriteLine("Acquire mode: {0}",
    dctAcqMode[strsPreamble[18]]);
```

```
    Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
    Console.WriteLine("Waveform X inits: {0}",
        dctUnits[strsPreamble[20]]);
    Console.WriteLine("Waveform Y units: {0}",
        dctUnits[strsPreamble[21]]);
    Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
    Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

    // Get numeric values for later calculations.
    double fXincrement;
    fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?");
    double fXorigin;
    fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?");
    double fYincrement;
    fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?");
    double fYorigin;
    fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?");

    // Get the waveform data.
    myScope.DoCommand(":WAVeform:STReaming OFF");
    nLength = myScope.DoQueryIEEEBlock_Words(":WAVeform:DATA?",
        out WordResultsArray);
    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)WordResultsArray[i] * 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_Bytes(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Bytes(out ResultsArray);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

public int DoQueryIEEEBlock_Words(string strQuery,
  out short[] ResultsArray)
{
```

```
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Words(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_Bytes(out byte[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new byte[5000000];
  int length;   // Number of bytes returned from instrument.

  // Set the default number of bytes that will be contained in
  // the ResultsArray to 5,000,000.
  length = 5000000;

  // 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 int VisaGetResultIEEEBlock_Words(out short[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new short[5000000];
  int length;   // Number of words returned from instrument.

  // Set the default number of words that will be contained in
  // the ResultsArray to 5,000,000.
  length = 5000000;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%#hb", 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? STRing");
    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 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. In the New Project dialog box, create a new Visual Basic, Console Application project.
4. Cut-and-paste the code that follows into the Visual Basic .NET source file.
5. Edit the program to use the VISA address of your oscilloscope.
6. 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** and then 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 "Infiniium.VisaInstrumentApp" as the **Startup object**.
7. 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 Infiniium Series oscilloscope.
' -------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace Infiniium
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
          New VisaInstrument("TCPIP0::141.121.237.226::hislip0::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)
      Finally
        myScope.Close()
      End Try
    End Sub

    '
    ' Initialize the oscilloscope to a known state.
    ' --------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As StringBuilder

      ' Clear status.
      myScope.DoCommand("*CLS")

      ' Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?")
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Load the default setup.
      myScope.DoCommand("*RST")

    End Sub
```

```
'
' Capture the waveform.
' -----------------------------------------------------------------

Private Shared Sub Capture()

  ' Set probe attenuation factor.
  'myScope.DoCommand(":CHANnel1:PROBe 1.0")
  Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
      myScope.DoQueryString(":CHANnel1:PROBe?"))

  ' Use auto-scale to automatically set up oscilloscope.
  myScope.DoCommand(":AUToscale")

  ' Set trigger mode.
  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:LEVel CHANnel1,-2E-3")
  Console.WriteLine("Trigger edge level: {0}", _
      myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"))

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
  Console.WriteLine("Trigger edge slope: {0}", _
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

  ' Save oscilloscope setup.
  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_Bytes(":SYSTem:SETup?", _
      ResultsArray)

  ' Write setup string to file.
  strPath = "c:\scope\config\setup.stp"
  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 0.1")
  Console.WriteLine("Channel 1 vertical scale: {0}", _
      myScope.DoQueryString(":CHANnel1:SCALe?"))
```

```
  myScope.DoCommand(":CHANnel1:OFFSet 0.0")
  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 mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe")
  Console.WriteLine("Acquire mode: {0}", _
      myScope.DoQueryString(":ACQuire:MODE?"))

  ' Or, set up by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.stp"
  DataArray = File.ReadAllBytes(strPath)

  ' Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
      DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000")
  myScope.DoCommand(":DIGitize")

End Sub

'
' Analyze the captured waveform.
' ----------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()    ' Results array.
  Dim WordResultsArray As Short()    ' Results array for WORD data.
  Dim nLength As Integer    ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make 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_Bytes(":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.
' -----------------------------------------------------------

' Get the waveform type.
Console.WriteLine("Waveform type: {0}", _
    myScope.DoQueryString(":WAVeform:TYPE?"))

' Get the number of waveform points.
Console.WriteLine("Waveform points: {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:
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")

Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
```

```
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

strPreamble = _
    myScope.DoQueryString(":WAVeform:PREamble?").ToString()
strsPreamble = strPreamble.Split(","c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))

Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
Console.WriteLine("Waveform Y display range: {0}", _
    strsPreamble(13))
Console.WriteLine("Waveform Y display origin: {0}", _
    strsPreamble(14))
Console.WriteLine("Date: {0}", strsPreamble(15))
Console.WriteLine("Time: {0}", strsPreamble(16))
Console.WriteLine("Frame model: {0}", strsPreamble(17))
Console.WriteLine("Acquire mode: {0}", _
    dctAcqMode(strsPreamble(18)))
```

```
      Console.WriteLine("Completion pct: {0}", strsPreamble(19))
      Console.WriteLine("Waveform X inits: {0}", _
          dctUnits(strsPreamble(20)))
      Console.WriteLine("Waveform Y units: {0}", _
          dctUnits(strsPreamble(21)))
      Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
      Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

      ' Get numeric values for later calculations.
      Dim fXincrement As Double
      fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?")
      Dim fXorigin As Double
      fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?")
      Dim fYincrement As Double
      fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?")
      Dim fYorigin As Double
      fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?")

      ' Get the waveform data.
      myScope.DoCommand(":WAVeform:STReaming OFF")
      nLength = myScope.DoQueryIEEEBlock_Words(":WAVeform:DATA?", _
          WordResultsArray)
      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 index As Integer = 0 To nLength - 1
        ' Write time value, voltage value.
        writer.WriteLine("{0:f9}, {1:f6}", _
            fXorigin + (CSng(index) * fXincrement), _
            (CSng(WordResultsArray(index)) * 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_Bytes(ByVal strQuery As String, _
      ByRef ResultsArray As Byte()) As Integer
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock_Bytes(ResultsArray)

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
```

```
  Return length
End Function

Public Function DoQueryIEEEBlock_Words(ByVal strQuery As String, _
    ByRef ResultsArray As Short()) As Integer
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim length As Integer
  ' Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Words(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_Bytes(ByRef ResultsArray _
    As Byte()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Byte(4999999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
  ' the ResultsArray to 5,000,000.
  length = 5000000

  ' 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 Function VisaGetResultIEEEBlock_Words(ByRef ResultsArray _
    As Short()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Short(4999999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
  ' the ResultsArray to 5,000,000.
  length = 5000000

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%#hb", 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? STRing")
    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 3

You can use the Python programming language with the PyVISA package to control Keysight Infiniium Series oscilloscopes.

The Python language and PyVISA package can be downloaded from the web at http://www.python.org/ and http://pyvisa.readthedocs.org/, 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

# *********************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight Infiniium Series oscilloscope.
# *********************************************************

# Import modules.
# ---------------------------------------------------------
import pyvisa
import string
import struct
import sys

# Global variables (booleans: 0 = False, 1 = True).
# ---------------------------------------------------------
debug = 0


# =========================================================
# Initialize:
# =========================================================
def initialize():

 # Clear status.
 do_command("*CLS")
```

```
    # Get and display the device's *IDN? string.
    idn_string = do_query_string("*IDN?")
    print("Identification string: '%s'" % idn_string)

    # Load the default setup.
    do_command("*RST")


# =========================================================
# Capture:
# =========================================================
def capture():

    # Set probe attenuation factor.
    #do_command(":CHANnel1:PROBe 1.0")
    qresult = do_query_string(":CHANnel1:PROBe?")
    print("Channel 1 probe attenuation factor: %s" % qresult)

    # 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("Trigger mode: %s" % qresult)

    # Set EDGE trigger parameters.
    do_command(":TRIGger:EDGE:SOURce CHANnel1")
    qresult = do_query_string(":TRIGger:EDGE:SOURce?")
    print("Trigger edge source: %s" % qresult)

    do_command(":TRIGger:LEVel CHANnel1,150E-3")
    qresult = do_query_string(":TRIGger:LEVel? CHANnel1")
    print("Trigger level, channel 1: %s" % qresult)

    do_command(":TRIGger:EDGE:SLOPe POSitive")
    qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
    print("Trigger edge slope: %s" % qresult)

    # Save oscilloscope setup.
    setup_bytes = do_query_ieee_block(":SYSTem:SETup?")

    f = open("setup.set", "wb")
    f.write(setup_bytes)
    f.close()
    print("Setup bytes saved: %d" % len(setup_bytes))

    # Change oscilloscope settings with individual commands:

    # Set vertical scale and offset.
    do_command(":CHANnel1:SCALe 0.1")
    qresult = do_query_number(":CHANnel1:SCALe?")
    print("Channel 1 vertical scale: %f" % qresult)

    do_command(":CHANnel1:OFFSet 0.0")
    qresult = do_query_number(":CHANnel1:OFFSet?")
```

```
    print("Channel 1 offset: %f" % qresult)

    # Set horizontal scale and offset.
    do_command(":TIMebase:SCALe 200e-6")
    qresult = do_query_string(":TIMebase:SCALe?")
    print("Timebase scale: %s" % qresult)

    do_command(":TIMebase:POSition 0.0")
    qresult = do_query_string(":TIMebase:POSition?")
    print("Timebase position: %s" % qresult)

    # Set the acquisition mode.
    do_command(":ACQuire:MODE RTIMe")
    qresult = do_query_string(":ACQuire:MODE?")
    print("Acquire mode: %s" % qresult)

    # Or, set up oscilloscope by loading a previously saved setup.
    setup_bytes = ""
    f = open("setup.set", "rb")
    setup_bytes = f.read()
    f.close()
    do_command_ieee_block(":SYSTem:SETup", setup_bytes)
    print("Setup bytes restored: %d" % len(setup_bytes))

    # Set the desired number of waveform points,
    # and capture an acquisition.
    do_command(":ACQuire:POINts 32000")
    do_command(":DIGitize")


# =========================================================
# Analyze:
# =========================================================
def analyze():

    # Make measurements.
    # --------------------------------------------------------
    do_command(":MEASure:SOURce CHANnel1")
    qresult = do_query_string(":MEASure:SOURce?")
    print("Measure source: %s" % qresult)

    do_command(":MEASure:FREQuency")
    qresult = do_query_string(":MEASure:FREQuency?")
    print("Measured frequency on channel 1: %s" % qresult)

    do_command(":MEASure:VAMPlitude")
    qresult = do_query_string(":MEASure:VAMPlitude?")
    print("Measured vertical amplitude on channel 1: %s" % qresult)

    # Download the screen image.
    # --------------------------------------------------------
    screen_bytes = do_query_ieee_block(":DISPlay:DATA? PNG")

    # Save display data values to file.
    f = open("screen_image.png", "wb")
    f.write(screen_bytes)
    f.close()
```

```
print("Screen image written to screen_image.png.")

# Download waveform data.
# ---------------------------------------------------------

# Get the waveform type.
qresult = do_query_string(":WAVeform:TYPE?")
print("Waveform type: %s" % qresult)

# Get the number of waveform points.
qresult = do_query_string(":WAVeform:POINts?")
print("Waveform points: %s" % qresult)

# Set the waveform source.
do_command(":WAVeform:SOURce CHANnel1")
qresult = do_query_string(":WAVeform:SOURce?")
print("Waveform source: %s" % qresult)

# Choose the format of the data returned:
do_command(":WAVeform:FORMat BYTE")
print("Waveform format: %s" % do_query_string(":WAVeform:FORMat?"))

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
 5 : "DECIBEL",
}
```

```
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, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = preamble_string.split(",")

print("Waveform format: %s" % wav_form_dict[int(wav_form)])
print("Acquire type: %s" % acq_type_dict[int(acq_type)])
print("Waveform points desired: %s" % wfmpts)
print("Waveform average count: %s" % avgcnt)
print("Waveform X increment: %s" % x_increment)
print("Waveform X origin: %s" % x_origin)
print("Waveform X reference: %s" % x_reference)   # Always 0.
print("Waveform Y increment: %s" % y_increment)
print("Waveform Y origin: %s" % y_origin)
print("Waveform Y reference: %s" % y_reference)   # Always 0.
print("Coupling: %s" % coupling_dict[int(coupling)])
print("Waveform X display range: %s" % x_display_range)
print("Waveform X display origin: %s" % x_display_origin)
print("Waveform Y display range: %s" % y_display_range)
print("Waveform Y display origin: %s" % y_display_origin)
print("Date: %s" % date)
print("Time: %s" % time)
print("Frame model #: %s" % frame_model)
print("Acquire mode: %s" % acq_mode_dict[int(acq_mode)])
print("Completion pct: %s" % completion)
print("Waveform X units: %s" % units_dict[int(x_units)])
print("Waveform Y units: %s" % units_dict[int(y_units)])
print("Max BW limit: %s" % max_bw_limit)
print("Min BW limit: %s" % min_bw_limit)

# 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?")

# Get the waveform data.
do_command(":WAVeform:STReaming OFF")
sData = do_query_ieee_block(":WAVeform:DATA?")

# Unpack signed byte data.
values = struct.unpack("%db" % len(sData), sData)
print("Number of data values: %d" % len(values))

# Save waveform data values to CSV file.
f = open("waveform_data.csv", "w")

for i in range(0, len(values) - 1):
 time_val = x_origin + (i * x_increment)
 voltage = (values[i] * y_increment) + y_origin
 f.write("%E, %f\n" % (time_val, voltage))
```

```
 f.close()
 print("Waveform format BYTE data written to waveform_data.csv.")


# =========================================================
# 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("\nCmd = '%s'" % header)
 else:
  if debug:
   print("\nCmd = '%s'" % command)

 Infiniium.write("%s" % 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("Cmb = '%s'" % command)
 Infiniium.write_binary_values("%s " % command, values, datatype='B')
 check_instrument_errors(command)


# =========================================================
# Send a query, check for errors, return string:
# =========================================================
def do_query_string(query):
 if debug:
  print("Qys = '%s'" % query)
 result = Infiniium.query("%s" % query)
 check_instrument_errors(query)
 return result


# =========================================================
# Send a query, check for errors, return floating-point value:
# =========================================================
def do_query_number(query):
 if debug:
  print("Qyn = '%s'" % query)
 results = Infiniium.query("%s" % 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("Qyb = '%s'" % query)
 result = Infiniium.query_binary_values("%s" % query, datatype='s', cont
ainer=bytes)
 check_instrument_errors(query, exit_on_error=False)
 return result


# =========================================================
# Check for instrument errors:
# =========================================================
def check_instrument_errors(command, exit_on_error=True):

 while True:
  error_string = Infiniium.query(":SYSTem:ERRor? STRing")
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".

    print("ERROR: %s, command: '%s'" % (error_string, command))
    if exit_on_error:
     print("Exited because of error.")
     sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print("ERROR: :SYSTem:ERRor? STRing returned nothing, command: '%s'"
% command)
   print("Exited because of error.")
   sys.exit(1)


# =========================================================
# Main program:
# =========================================================

rm = pyvisa.ResourceManager("C:\\Windows\\System32\\visa64.dll")
Infiniium = rm.open_resource("TCPIP0::141.121.231.13::hislip0::INSTR")

Infiniium.timeout = 20000
Infiniium.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

Infiniium.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 2117
- "VISA.NET Example in Visual Basic .NET" on page 2124
- "VISA.NET Example in Python 3" on page 2131

VISA.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. In the New Project dialog box, select **.NET Framework 4.5.2**.
4. Create a new Visual C#, Console Application project.
5. Cut-and-paste the code that follows into the C# source file.
6. Edit the program to use the VISA address of your oscilloscope.
7. 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 4.5.2" list, select the **Ivi.Visa Assembly** check box; then, click **OK**.
8. 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 Infiniium Series 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 = "TCPIP0::141.121.231.13::hislip0::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;

      // Clear status.
      myScope.WriteLine("*CLS");

      // Get and display the device's *IDN? string.
      myScope.WriteLine("*IDN?");
      strResults = myScope.ReadLine();
      Console.WriteLine("*IDN? result is: {0}", strResults);

      // Load the default setup.
      myScope.WriteLine("*RST");
```

```
// Capture data.
// ==================================================================
// Set probe attenuation factor.
//myScope.WriteLine(":CHANnel1:PROBe 1.0");
myScope.WriteLine(":CHANnel1:PROBe?");
strResults = myScope.ReadLine();
Console.WriteLine("Channel 1 probe attenuation factor: {0}",
    strResults);

// 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:LEVel CHANnel1,-2E-3");
myScope.WriteLine(":TRIGger:LEVel? CHANnel1");
strResults = myScope.ReadLine();
Console.WriteLine("Trigger level, channel 1: {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.stp";
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 0.1");
myScope.WriteLine(":CHANnel1:SCALe?");
strResults = myScope.ReadLine();
```

```
Console.WriteLine("Channel 1 vertical scale: {0}", strResults);

myScope.WriteLine(":CHANnel1:OFFSet 0.0");
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 mode.
myScope.WriteLine(":ACQuire:MODE RTIMe");
myScope.WriteLine(":ACQuire:MODE?");
strResults = myScope.ReadLine();
Console.WriteLine("Acquire mode: {0}", strResults);

// Or, configure by loading a previously saved setup.
byte[] DataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.stp";
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);

// Set the desired number of waveform points,
// and capture an acquisition.
myScope.WriteLine(":ACQuire:POINts 32000");
myScope.WriteLine(":DIGitize");

// 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.
// -----------------------------------------------------------

// Get the waveform type.
myScope.WriteLine(":WAVeform:TYPE?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform type: {0}", strResults);

// Get the number of waveform points.
myScope.WriteLine(":WAVeform:POINts?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform points: {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:
myScope.WriteLine(":WAVeform:FORMat WORD");
myScope.WriteLine(":WAVeform:FORMat?");
strResults = myScope.ReadLine();
Console.WriteLine("Waveform format: {0}", strResults);

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
```

```
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
  new Dictionary<string, string>()
{
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

myScope.WriteLine(":WAVeform:PREamble?");
strPreamble = myScope.ReadLine();
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
```

```
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
Console.WriteLine("Date: {0}", strsPreamble[15]);
Console.WriteLine("Time: {0}", strsPreamble[16]);
Console.WriteLine("Frame model: {0}", strsPreamble[17]);
Console.WriteLine("Acquire mode: {0}",
    dctAcqMode[strsPreamble[18]]);
Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
Console.WriteLine("Waveform X inits: {0}",
    dctUnits[strsPreamble[20]]);
Console.WriteLine("Waveform Y units: {0}",
    dctUnits[strsPreamble[21]]);
Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

// Get numeric values for later calculations.
double fXincrement;
myScope.WriteLine(":WAVeform:XINCrement?");
fXincrement = myScope.ReadLineDouble();

double fXorigin;
myScope.WriteLine(":WAVeform:XORigin?");
fXorigin = myScope.ReadLineDouble();

double fYincrement;
myScope.WriteLine(":WAVeform:YINCrement?");
fYincrement = myScope.ReadLineDouble();

double fYorigin;
myScope.WriteLine(":WAVeform:YORigin?");
fYorigin = myScope.ReadLineDouble();

// Read waveform data.
myScope.WriteLine(":WAVeform:STReaming OFF");
short[] WordDataArray;   // Waveform data array.
myScope.WriteLine(":WAVeform:DATA?");
WordDataArray = myScope.ReadBinaryBlockOfInt16();
//WordDataArray = myScope.ReadBinaryBlockOfInt16(true);
nLength = WordDataArray.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)WordDataArray[i])
                  * 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 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. In the New Project dialog box, select **.NET Framework 4.5.2**.
4. Create a new Visual Basic, Console Application project.
5. Cut-and-paste the code that follows into the Visual Basic .NET source file.
6. Edit the program to use the VISA address of your oscilloscope.
7. 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 4.5.2" list, select the **Ivi.Visa Assembly** check box; then, click **OK**.
8. 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.

9 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 Infiniium Series 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 = "TCPIP0::141.121.231.13::hislip0::INS
TR"

      ' 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

' Clear status.
myScope.WriteLine("*CLS")

' Get and display the device's *IDN? string.
myScope.WriteLine("*IDN?")
strResults = myScope.ReadLine()
Console.WriteLine("*IDN? result is: {0}", strResults)

' Load the default setup.
myScope.WriteLine("*RST")

' Capture data.
' ==============================================================
' Set probe attenuation factor.
'myScope.WriteLine(":CHANnel1:PROBe 1.0")
myScope.WriteLine(":CHANnel1:PROBe?")
strResults = myScope.ReadLine()
Console.WriteLine("Channel 1 probe attenuation factor: {0}",
    strResults)

' 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:LEVel CHANnel1,-2E-3")
myScope.WriteLine(":TRIGger:LEVel? CHANnel1")
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.stp"
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 0.1")
myScope.WriteLine(":CHANnel1:SCALe?")
strResults = myScope.ReadLine()
Console.WriteLine("Channel 1 vertical scale: {0}", strResults)

myScope.WriteLine(":CHANnel1:OFFSet 0.0")
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 mode.
myScope.WriteLine(":ACQuire:MODE RTIMe")
myScope.WriteLine(":ACQuire:MODE?")
strResults = myScope.ReadLine()
Console.WriteLine("Acquire mode: {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.stp"
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)

' Set the desired number of waveform points,
```

```
' and capture an acquisition.
myScope.WriteLine(":ACQuire:POINts 32000")
myScope.WriteLine(":DIGitize")

' 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 type.
myScope.WriteLine(":WAVeform:TYPE?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform type: {0}", strResults)

' Get the number of waveform points.
myScope.WriteLine(":WAVeform:POINts?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform points: {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:
myScope.WriteLine(":WAVeform:FORMat WORD")
myScope.WriteLine(":WAVeform:FORMat?")
strResults = myScope.ReadLine()
Console.WriteLine("Waveform format: {0}", strResults)

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")

Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

myScope.WriteLine(":WAVeform:PREamble?")
strPreamble = myScope.ReadLine()
strsPreamble = strPreamble.Split(","c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))
```

```
Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
Console.WriteLine("Waveform Y display range: {0}", _
    strsPreamble(13))
Console.WriteLine("Waveform Y display origin: {0}", _
    strsPreamble(14))
Console.WriteLine("Date: {0}", strsPreamble(15))
Console.WriteLine("Time: {0}", strsPreamble(16))
Console.WriteLine("Frame model: {0}", strsPreamble(17))
Console.WriteLine("Acquire mode: {0}", _
    dctAcqMode(strsPreamble(18)))
Console.WriteLine("Completion pct: {0}", strsPreamble(19))
Console.WriteLine("Waveform X inits: {0}", _
    dctUnits(strsPreamble(20)))
Console.WriteLine("Waveform Y units: {0}", _
    dctUnits(strsPreamble(21)))
Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

' Get numeric values for later calculations.
Dim fXincrement As Double
myScope.WriteLine(":WAVeform:XINCrement?")
fXincrement = myScope.ReadLineDouble()

Dim fXorigin As Double
myScope.WriteLine(":WAVeform:XORigin?")
fXorigin = myScope.ReadLineDouble()

Dim fYincrement As Double
myScope.WriteLine(":WAVeform:YINCrement?")
fYincrement = myScope.ReadLineDouble()

Dim fYorigin As Double
myScope.WriteLine(":WAVeform:YORigin?")
fYorigin = myScope.ReadLineDouble()

' Read waveform data.
myScope.WriteLine(":WAVeform:STReaming OFF")
Dim WordDataArray As Short()
myScope.WriteLine(":WAVeform:DATA?")
WordDataArray = myScope.ReadLineBinaryBlockOfInt16()
nLength = WordDataArray.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 index As Integer = 0 To nLength - 1
          ' Write time value, voltage value.
          writer.WriteLine("{0:f9}, {1:f6}", _
              fXorigin + (CSng(index) * fXincrement), _
              (CSng(WordDataArray(index)) * 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 3

You can use the Python programming language with the "Python.NET" package 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
#
# Keysight VISA.NET Example in Python.NET
# *********************************************************
# This program illustrates a few commonly used programming
# features of your Keysight InfiniiVision oscilloscope.
```

```
# Prerequisites: "pip install pythonnet"
# ***********************************************************

# Import Python modules.
# -----------------------------------------------------------
import sys
import array
sys.path.append("C:\\Program Files\\IVI Foundation\\VISA\\Microsoft.NET\
\Framework64\\v2.0.50727\\VISA.NET Shared Components 5.11.0")

# 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 = "TCPIP0::lab-myst-lp2-41.cos.is.keysight.com::inst0::INSTR"
io_timeout_ms = 60000
input_channel = "CHANnel2"
waveform_format = "WORD"   # "BYTE" also supported


# ===========================================================
# Check for instrument errors:
# ===========================================================
def check_instrument_errors(when):

 errors_found = False
 while True:
  # Keep reading errors until "No error".
  myScope.WriteLine(":SYSTem:ERRor? STRing")
  error_string = myScope.ReadLine().strip()
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    errors_found = True
    print(f"ERROR: {error_string}")

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing 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(f":TRIGger:LEVel {input_channel},1.75")
 myScope.WriteLine(f":TRIGger:LEVel? {input_channel}")
 qresult = myScope.ReadLine().strip()
 print(f"Trigger level, {input_channel}: {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.set", 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 0.1")
 myScope.WriteLine(f":{input_channel}:SCALe?")
 qresult = myScope.ReadLine().strip()
 print(f"{input_channel} vertical scale: {qresult}")

 myScope.WriteLine(f":{input_channel}:OFFSet 0.0")
 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 mode.
 myScope.WriteLine(":ACQuire:MODE RTIMe")
 myScope.WriteLine(":ACQuire:MODE?")
 qresult = myScope.ReadLine().strip()
 print(f"Acquire mode: {qresult}")

 # Or, configure by loading a previously saved setup.

 # Read setup string from file.
 setup_bytes = File.ReadAllBytes("setup.set")

 # 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)}")

 # Set the desired number of waveform points.
 myScope.WriteLine(":ACQuire:POINts 32000")
 myScope.WriteLine(":ACQuire:POINts?")
 qresult = myScope.ReadLine().strip()
 print(f"Acquire points setting: {qresult}")

 # Capture an acquisition using :DIGitize.
 myScope.WriteLine(":DIGitize")


# ==========================================================
# 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}")

myScope.WriteLine(":MEASure:VAMPlitude")
myScope.WriteLine(":MEASure:VAMPlitude?")
qresult = myScope.ReadLineDouble()
print(f"Measured vertical amplitude on {input_channel}: {qresult}")

# Download the screen image.
# ---------------------------------------------------------

# Get the screen data.
myScope.WriteLine(":DISPlay:DATA? PNG")
image_bytes = myScope.ReadLineBinaryBlockOfByte()
nLength = len(image_bytes)
fStream = File.Open("screen_image.png", FileMode.Create)
fStream.Write(image_bytes, 0, nLength)
fStream.Close()
print("Screen image written to screen_image.png.")

# 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}")

# Get the waveform type.
myScope.WriteLine(":WAVeform:TYPE?")
qresult = myScope.ReadLine().strip()
print(f"Waveform type: {qresult}")

# Get the number of waveform points available.
myScope.WriteLine(":WAVeform:POINts?")
qresult = myScope.ReadLine().strip()
print(f"Waveform points available: {qresult}")

# Choose the format of the data returned:
myScope.WriteLine(f":WAVeform:FORMat {waveform_format}")
myScope.WriteLine(":WAVeform:FORMat?")
qresult = myScope.ReadLine().strip()
print(f"Waveform format: {qresult}")

if waveform_format == "WORD":
 # Use the LSB first byte ordering.
 myScope.WriteLine(":WAVeform:BYTeorder LSBFirst")
 myScope.WriteLine(":WAVeform:BYTeorder?")
```

```
 qresult = myScope.ReadLine().strip()
 print(f"Waveform byte order: {qresult}")

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
 5 : "DECIBEL",
}

myScope.WriteLine(":WAVeform:PREamble?")
(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
 x_reference, y_increment, y_origin, y_reference, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = myScope.ReadLine().strip().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}")   # Always 0.
    print(f"Coupling: {coupling_dict[int(coupling)]}")
    print(f"Waveform X display range: {x_display_range}")
    print(f"Waveform X display origin: {x_display_origin}")
    print(f"Waveform Y display range: {y_display_range}")
    print(f"Waveform Y display origin: {y_display_origin}")
    print(f"Date: {date}")
    print(f"Time: {time}")
    print(f"Frame model #: {frame_model}")
    print(f"Acquire mode: {acq_mode_dict[int(acq_mode)]}")
    print(f"Completion pct: {completion}")
    print(f"Waveform X units: {units_dict[int(x_units)]}")
    print(f"Waveform Y units: {units_dict[int(y_units)]}")
    print(f"Max BW limit: {max_bw_limit}")
    print(f"Min BW limit: {min_bw_limit}")

    # Get numeric values for later calculations.
    myScope.WriteLine(":WAVeform:XINCrement?")
    x_increment = myScope.ReadLineDouble()
    myScope.WriteLine(":WAVeform:XORigin?")
    x_origin = myScope.ReadLineDouble()
    myScope.WriteLine(":WAVeform:YINCrement?")
    y_increment = myScope.ReadLineDouble()
    myScope.WriteLine(":WAVeform:YORigin?")
    y_origin = myScope.ReadLineDouble()

    # Get the waveform data.
    myScope.WriteLine(":WAVeform:STReaming OFF")
    myScope.WriteLine(":WAVeform:DATA?")
    if waveform_format == "BYTE":
     block_data = myScope.ReadLineBinaryBlockOfSByte()
     data = array.array("b", block_data)
    else:   # waveform_format == "WORD":
     block_data = myScope.ReadLineBinaryBlockOfInt16()
     data = array.array("h", block_data)
     data.byteswap()
    nLength = len(data)
    print(f"Number of data values: {nLength}")

    # Open file for output.
    strPath = "waveform_data.csv"
    writer = File.CreateText(strPath)

    # Output waveform data in CSV format.
    for i in range(0, nLength - 1):
     time_val = x_origin + (i * x_increment)
     voltage = (data[i] * y_increment) + y_origin
     writer.WriteLine("%E, %f" % (time_val, voltage))

    # Close output file.
    writer.Close()
    print(f"Waveform format {waveform_format} data written to {strPath}.")


   # ==========================================================
   # 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" on page 2139
- "SICL Example in Visual Basic" on page 2148

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2013:

1. Open Visual Studio.
2. Choose **FILE > New > Project...**.
3. In the New Project dialog box, create a new Visual C++, Win32 Console Application project.
4. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
5. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
6. In Visual Studio 2013, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
7. Edit the program to use the SICL address of your oscilloscope.
8. Choose **Project > Properties...**. In the Property Pages dialog, 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, select Preprocessor Definitions and add _CRT_SECURE_NO_WARNINGS.
   d. Under Configuration Properties, VC++ Directories, select Include Directories and add the include directory (for example, Program Files\Agilent\ IO Libraries Suite\include).
   e. Under Configuration Properties, VC++ Directories, select Library Directories and add the include directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
   f. Click **OK** to close the Property Pages dialog.
9. Build and run the program.

```
/*
 * Keysight SICL Example in C
 * ------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight Infiniium Series 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      "lan,4880;hislip[141.121.237.226]:hislip0"
#define TIMEOUT           15000
#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. */
int do_query_ieeeblock_words(char *query);   /* Query for word data. */
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(). */
signed short ieeeblock_data_words[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock_words(). */
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.  On Windows 98SE or Windows Me, view messages with
   * the SICL Message Viewer.  For Windows 2000 or XP, use the Event
   * Viewer.
   */
  ionerror(I_ERROR_EXIT);

  /* Open a device session using the SICL_ADDRESS */
  id = iopen(SICL_ADDRESS);

  if (id == 0)
  {
    printf ("Oscilloscope iopen failed!\n");
  }
  else
  {
    printf ("Oscilloscope session opened!\n");
  }
```

```
  /* Set the I/O timeout value for this session to 5 seconds. */
  itimeout(id, TIMEOUT);

  /* Clear the interface. */
  iclear(id);

  /* 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)
{
  /* Clear status. */
  do_command("*CLS");

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * -------------------------------------------------------------- */
void capture (void)
{
  int num_values;
  FILE *fp;

  /* Set probe attenuation factor. */
  /* do_command(":CHANnel1:PROBe 1.0"); */
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope.
   * ------------------------------------------------------------ */
  do_command(":AUToscale");
```

```
/* Set trigger mode. */
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:LEVel CHANnel1,-2E-3");
do_query_string(":TRIGger:LEVel? CHANnel1");
printf("Trigger level, channel 1: %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_values = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_values);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:
 * ------------------------------------------------------------ */

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.1");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet 0.0");
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 mode. */
do_command(":ACQuire:MODE RTIMe");
```

```
  do_query_string(":ACQuire:MODE?");
  printf("Acquire mode: %s\n", str_result);

  /* Or, configure by loading a previously saved setup.
  * ------------------------------------------------------------ */

  /* Read setup string from file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
  num_values = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
  fclose (fp);
  printf("Read setup string (%d bytes) from file ", num_values);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Restore setup string. */
  num_values = do_command_ieeeblock(":SYSTem:SETup", num_values);
  printf("Restored setup string (%d bytes).\n", num_values);

  /* Set the desired number of waveform points,
   * and capture an acquisition. */
  do_command(":ACQuire:POINts 32000");
  do_command(":DIGitize");
}

/* 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 y_increment;
  double y_origin;

  FILE *fp;
  int num_values;   /* Number of bytes returned from instrument. */
  int i;

  /* Make 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_values = do_query_ieeeblock(":DISPlay:DATA? PNG");
printf("Screen image bytes: %d\n", num_values);

/* Write screen image bytes to file. */
fp = fopen ("c:\\scope\\data\\screen.png", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote screen image (%d bytes) to ", num_values);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * -------------------------------------------------------------- */

/* Get the waveform type. */
do_query_string(":WAVeform:TYPE?");
printf("Waveform type: %s\n", str_result);

/* Get the number of waveform points. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points: %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: */
do_command(":WAVeform:FORMat WORD");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_number(":WAVeform:XINCrement?");
x_increment = num_result;
printf("Waveform X increment: %e\n", x_increment);

do_query_number(":WAVeform:XORigin?");
x_origin = num_result;
printf("Waveform X origin: %e\n", x_origin);

do_query_number(":WAVeform:YINCrement?");
y_increment = num_result;
printf("Waveform Y increment: %e\n", y_increment);

do_query_number(":WAVeform:YORigin?");
y_origin = num_result;
printf("Waveform Y origin: %e\n", y_origin);

/* Read waveform data. */
num_values = do_query_ieeeblock_words(":WAVeform:DATA?");
printf("Number of data values: %d\n", num_values);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");
```

```
  /* Output waveform data in CSV format. */
  for (i = 0; i < num_values - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      ((float)ieeeblock_data_words[i] * 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);
}

/* Query for an IEEE definite-length block word data result.
 * --------------------------------------------------------------- */
int do_query_ieeeblock_words(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  data_length = IEEEBLOCK_SPACE;
  iscanf(id, "%#wb", &data_length, ieeeblock_data_words);

  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? STRing\n", "%t", str_err_val);
  while(strncmp(str_err_val, "0,", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    ipromptf(id, ":SYSTem:ERRor? STRing\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);
  }
}
```

SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

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. Add the sicl32.bas file to your project:
   a. Choose **File > Import File...**.
   b. Navigate to the header file, sicl32.bas (installed with Keysight IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.
4. Choose **Insert > Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the SICL address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Keysight SICL Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight Infiniium Series oscilloscope.
' -------------------------------------------------------------------

Option Explicit

Public id As Integer   ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -------------------------------------------------------------------

Sub Main()

  On Error GoTo ErrorHandler
```

```
  ' Open a device session using the SICL_ADDRESS.
  id = iopen("lan,4880;hislip[141.121.237.226]:hislip0")
  Call itimeout(id, 15000)

  ' Clear the interface.
  Call iclear(id)

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  Call iclose(id)

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' -------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo ErrorHandler

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

  ' Load the default setup.
  DoCommand "*RST"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Capture the waveform.
```

```
' -------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo ErrorHandler

  ' Set probe attenuation factor.
  'DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically configure oscilloscope.
  ' -----------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode.
  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:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' -----------------------------------------------------------------
  Dim lngSetupStringSize As Long
  lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
  Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

  ' Output setup string to a file:
  Dim strPath As String
  strPath = "c:\scope\config\setup.dat"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If

  ' Open file for output.
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.

  ' Change settings with individual commands:
```

```
  ' --------------------------------------------------------------------

  ' Set vertical scale and offset.
  DoCommand ":CHANnel1:SCALe 0.1"
  Debug.Print "Channel 1 vertical scale: " + _
      DoQueryString(":CHANnel1:SCALe?")

  DoCommand ":CHANnel1:OFFSet 0.0"
  Debug.Print "Channel 1 vertical offset: " + _
      DoQueryString(":CHANnel1:OFFSet?")

  ' Set horizontal scale and position.
  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 mode.
  DoCommand ":ACQuire:MODE RTIMe"
  Debug.Print "Acquire mode: " + _
      DoQueryString(":ACQuire:MODE?")

  ' Or, configure by loading a previously saved setup.
  ' --------------------------------------------------------------------
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As hFile   ' Open file for input.
  Dim lngSetupFileSize As Long
  lngSetupFileSize = LOF(hFile)  ' Length of file.
  Get hFile, , byteArray   ' Read data.
  Close hFile   ' Close file.
  ' Write setup string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  Dim lngRestored As Long
  lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' --------------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Analyze the captured waveform.
' ----------------------------------------------------------------------
```

```
Private Sub Analyze()

  On Error GoTo ErrorHandler

  ' Make measurements.
  ' -------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"

  ' Download the screen image.
  ' -------------------------------------------------------------------

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' 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
  Dim lngI As Long
  ' Skip past header.
  For lngI = CInt(Chr(byteArray(1))) + 2 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
  MsgBox "Screen image written to " + strPath

  ' Download waveform data.
  ' -------------------------------------------------------------------

  ' Get the waveform type.
  Debug.Print "Waveform type: " + _
      DoQueryString(":WAVeform:TYPE?")

  ' Get the number of waveform points.
  Debug.Print "Waveform points: " + _
      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:
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double

dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment: " + _
    Format(dblYIncrement, "Scientific")

dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin: " + _
    FormatNumber(dblYOrigin, 0)

' Get the waveform data
DoCommand ":WAVeform:STReaming OFF"
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(lngNumBytes - CInt(Chr(byteArray(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 byteUnsigned As Byte

' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngNumBytes - 2
  byteUnsigned = byteArray(lngI)
  ' Oscilloscope BYTE format sends signed bytes. VBA Byte is
  ' interpreted as unsigned, so convert the bits to signed value.
  lngDataValue = byteUnsigned - ((byteUnsigned And &H80) * 2)

  ' Write time value, voltage value.
```

```
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber((lngDataValue * dblYIncrement) + dblYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo ErrorHandler

  Call ivprintf(id, command + vbLf)

  CheckInstrumentErrors

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  On Error GoTo ErrorHandler

  ' Send command part.
  Call ivprintf(id, command + " ")

  ' Write definite-length block bytes.
  Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)

  ' retCount is now actual number of bytes written.
  DoCommandIEEEBlock = retCount

  CheckInstrumentErrors

  Exit Function

ErrorHandler:
```

```
  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryString(query As String) As String

  Dim actual As Long

  On Error GoTo ErrorHandler

  Dim strResult As String * 200

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%200t", strResult)
  DoQueryString = strResult

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumber(query As String) As Double

  On Error GoTo ErrorHandler

  Dim dblResult As Double

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%lf" + vbLf, dblResult)
  DoQueryNumber = dblResult

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumbers(query As String) As Double()

  On Error GoTo ErrorHandler

  Dim dblResults(10) As Double

  Call ivprintf(id, query + vbLf)
```

```
  Call ivscanf(id, "%,10lf" + vbLf, dblResults)
  DoQueryNumbers = dblResults

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  On Error GoTo ErrorHandler

  ' Send query.
  Call ivprintf(id, query + vbLf)

  ' Read definite-length block bytes.
  Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)

  ' Get number of block length digits.
  Dim intLengthDigits As Integer
  intLengthDigits = CInt(Chr(byteArray(1)))

  ' Get block length from those digits.
  Dim strBlockLength As String
  strBlockLength = ""
  Dim i As Integer
  For i = 2 To intLengthDigits + 1
    strBlockLength = strBlockLength + Chr(byteArray(i))
  Next

  ' Return number of bytes in block plus header.
  DoQueryIEEEBlock_Bytes = CLng(strBlockLength) + intLengthDigits + 2

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String
```

```
  Call ivprintf(id, ":SYSTem:ERRor? STRing" + vbLf) ' Query any errors d
ata.
  Call ivscanf(id, "%200t", strErrVal)   ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    Call ivprintf(id, ":SYSTem:ERRor? STRing" + vbLf) ' Request error me
ssage.
    Call ivscanf(id, "%200t", strErrVal)   ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    Call iflush(id, I_BUF_READ Or I_BUF_WRITE)

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub
```

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.

53 Reference

HDF5 Example

Here is an example of a captured HDF5 file.

```
Channel 1(6576)
Group Size = 1
Number of Attributes = 17
Waveform Type = 1
Start = 1
NumPoints = 1000000
NumSegments = 0
Count = 1
XDispRange = 1.0E-6
XDispOrigin = -5.0E-7
XInc = 5.0E-11
XOrg = -2.4999999E-5
XUnits = Second
YDispRange = 8.0
YDispOrigin = 0.0
YInc = 1.327218738E-4
YOrg = 0.11645629362732
YUnits = Volt
MinBandwidth = 0.0
MaxBandwidth = 6.0E9
```

CSV and TSV Header Format

Revision Always 0 (zero).

Type How the waveform was acquired: normal, raw, interpolate, average, or versus. When this field is read back into the scope, all modes, except versus, are converted to raw. The default value is raw.

Start Starting point in the waveform of the first data point in the file. This is usually zero.

Points The number of points in the waveform record. The number of points is set by the Memory Depth control. The default value is 1.

Count or Segments For count, it is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. Count is ignored when it is read back into the scope. The default value is 0.

Segments is used instead of Count when the data is acquired using the Segmented acquisition mode. This number is the total number of segments that were acquired.

XDispRange The number of X display range columns (n) depends on the number of sources being stored. The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.

XDispOrg The number of X display origin columns (n) depends on the number of sources being stored. The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

XInc The number of X increment columns (n) depends on the number of sources being store. The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.

XOrg The number of X origin columns (n) depends on the number of sources being store. The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

XUnits The number of X units columns (n) depends on the number of sources being store. The X units is the unit of measure for each time value of the acquired data.

YDispRange The number of Y display range columns (n) depends on the number of sources being store. The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero then no data has been acquired.

YDispOrg The number of Y display origin columns (n) depends on the number of sources being store. The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero then no data has been acquired.

YInc The number of Y increment columns (n) depends on the number of sources being store. The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero then no data has been acquired.

YOrg The number of Y origin columns (n) depends on the number of sources being store. The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero then no data has been acquired.

YUnits The number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform.

Frame A string containing the model number and serial number of the scope in the format of MODEL#:SERIAL#.

Date The date when the waveform was acquired. The default value is 27 DEC 1996.

Time The time when the waveform was acquired. The default value is 01:00:00:00.

Max bandwidth An estimation of the maximum bandwidth of the waveform. The default value is 0.

Min bandwidth An estimation of the minimum bandwidth of the waveform. The default value is 0.

Time Tags The Time Tags only occur when the data was acquired using the Segmented acquisition mode with time tags enabled and the file format is YValues. The number of columns depends on the number of Segments being saved.

Data The data values follow this header entry.

BIN Header Format

- "File Header" on page 2163
- "Waveform Header" on page 2163
- "Waveform Data Header" on page 2165
- "Example Program for Reading Binary Data" on page 2166

File Header

There is only one file header in a binary file. The file header consists of the following information.

Cookie Two byte characters, AG, which indicates that the file is in the Keysight Binary Data file format.

Version Two bytes which represent the file version.

File Size An integer (4 byte signed) which is the number of bytes that are in the file.

Number of Waveforms An integer (4 byte signed) which is the number of waveforms that are stored in the file.

Waveform Header

The waveform header contains information about the type of waveform data that is stored following the waveform data header which is located after each waveform header. Because it is possible to store more than one waveform in the file, there will be a waveform header and a waveform data header for each waveform.

Header Size An integer (4 byte signed) which is the number of bytes in the header.

Waveform Type An integer (4 byte signed) which is the type of waveform that is stored in the file. The follow shows what each value means.

0 = Unknown

1 = Normal

2 = Peak Detect

3 = Average

4 = Horizontal Histogram

5 = Vertical Histogram

6 = Logic

Number of Waveform Buffers An integer (4 byte signed) which is the number of waveform buffers required to read the data. This value is one except for peak detect data and digital data.

Points An integer (4 byte signed) that is the number of waveform points in the data.

Count An integer (4 byte signed) which is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. The default value is 0.

X Display Range A float (4 bytes) which is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.

X Display Origin A double (8 bytes) which is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

X Increment A double (8 bytes) which is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.

X Origin A double (8 bytes) which is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

X Units An integer (4 byte signed) which is the number of X units columns (n) depends on the number of sources being stored. The X units is the unit of measure for each time value of the acquired data. X unit definitions are:

0 = Unknown

1 = Volt

2 = Second

3 = Constant

4 = Amp

5 = Decibel

6 = Hz

Y Units An integer (4 byte signed) which is the number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform. Y units definitions are:

0 = Unknown

1 = Volt

2 = Second

3 = Constant

4 = Amp

5 = Decibel

6 = Hz

Date A 16 character array which is the date when the waveform was acquired. The default value is 27 DEC 1996.

Time A 16 character array which is the time when the waveform was acquired. The default value is 01:00:00:00.

Frame A 24 character array which is the model number and serial number of the scope in the format of MODEL#:SERIAL#.

Waveform Label A 16 character array which is the waveform label.

Time Tags A double (8 bytes) which is the time tag value of the segment being saved.

Segment Index An unsigned integer (4 byte signed) which is the segment index of the data that follows the waveform data header.

Waveform Data Header

The waveform data header consists of information about the waveform data points that are stored immediately after the waveform data header.

Waveform Data Header Size An integer (4 byte signed) which is the size of the waveform data header.

Buffer Type A short (2 byte signed) which is the type of waveform data that is stored in the file. The following shows what each value means.

0 = Unknown data

1 = Normal 32 bit float data

2 = Maximum float data

3 = Minimum float data

4 = Time float data

5 = Counts 32 bit float data

6 = Digital unsigned 8 bit char data

Bytes Per Point A short (2 byte signed) which is the number of bytes per data point.

Buffer Size An integer (4 byte signed) which is the size of the buffer required to hold the data bytes.

Example Program for Reading Binary Data

The following is a programming example of reading a Binary Data (.bin) file and converting it to a CSV (.csv) file without a file header.

```
/* bintoascii.c */

/* Reads the binary file format.
   This program demonstrates how to import the Infiniium
   oscilloscope binary file format and how to export it to an
   ascii comma separated file format.
*/
#pragma pack(4)

#include <stdio.h>               /* location of: printf() */
#include <stdlib.h>      /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0" /* Change the IP address
                                              * to the one found in
                                              * the Remote Setup
                                              * dialog box.
                                              */
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Type definitions */
typedef unsigned _int64 UINT64; /* This defines a 64-bit unsigned
                                 * integer for Microsoft platforms.
                                 */

/* Structure and Union definitions */
union DATATYPE
{
   char buffer[MAX_LENGTH];  /* Buffer for reading word format data */
   char byte[MAX_LENGTH];
   unsigned short word[MAX_LENGTH/2];
   UINT64 longlong[MAX_LENGTH/4];
};


typedef struct
{
   char Cookie[2];
   char Version[2];
   int  FileSize;
   int  NumberOfWaveforms;
} FileHeader;

const char COOKIE[2] = {'A', 'G'};
const char VERSION[2] = {'1', '0'};
```

```
#define DATE_TIME_STRING_LENGTH 16
#define FRAME_STRING_LENGTH  24
#define SIGNAL_STRING_LENGTH 16

typedef struct
{
   int    HeaderSize;
   int    WaveformType;
   int    NWaveformBuffers;
   int    Points;
   int    Count;
   float  XDisplayRange;
   double XDisplayOrigin;
   double XIncrement;
   double XOrigin;
   int    XUnits;
   int    YUnits;
   char   Date[DATE_TIME_STRING_LENGTH];
   char   Time[DATE_TIME_STRING_LENGTH];
   char   Frame[FRAME_STRING_LENGTH];
   char   WaveformLabel[SIGNAL_STRING_LENGTH];
   double TimeTag;
   unsigned int SegmentIndex;
} WaveformHeader;

typedef struct
{
   int   HeaderSize;
   short BufferType;
   short BytesPerPoint;
   int   BufferSize;
} WaveformDataHeader;

typedef enum
{
   PB_UNKNOWN,
   PB_NORMAL,
   PB_PEAK_DETECT,
   PB_AVERAGE,
   PB_HORZ_HISTOGRAM,
   PB_VERT_HISTOGRAM,
   PB_LOGIC
} WaveformType;

typedef enum
{
   PB_DATA_UNKNOWN,
   PB_DATA_NORMAL,
   PB_DATA_MAX,
   PB_DATA_MIN,
   PB_DATA_TIME,
   PB_DATA_COUNTS,
   PB_DATA_LOGIC
} DataType;

/* Prototypes */
void GetTimeConversionFactors( WaveformHeader waveformHeader,
```

```
                               double *xInc, double *xOrg );
void OutputNormalWaveform( WaveformHeader waveformHeader );
void OutputPeakDetectWaveform( WaveformHeader waveformHeader );
void OutputHistogramWaveform( WaveformHeader waveformHeader );
void OutputData( FILE *PeakFile,
                 WaveformDataHeader waveformDataHeader );

/* Globals */
double xOrg=0L, xInc=0L;     /* Values necessary to create time data */
union DATATYPE WaveFormData; /* Used to input and output data */
FILE *InputFile = NULL;
FILE *OutputFile;
errno_t err;
char *buffer;
float Volts[MAX_LENGTH];
float MaxVolts[MAX_LENGTH];
float MinVolts[MAX_LENGTH];
UINT64 HistogramData[MAX_LENGTH];

int main( int argc, char **argv )
{
   FileHeader fileHeader;
   WaveformHeader waveformHeader;

   if( argc > 1 )
   {
      InputFile = fopen( argv[1], "rb" );

      if( InputFile )
      {
         OutputFile = fopen( argv[2], "wb" );

         if( OutputFile )
         {
            /* Read the File Header */
            fread( &fileHeader, 1, sizeof( FileHeader ), InputFile );

            /* Make sure that this is a Keysight Binary File */
            if( (fileHeader.Cookie[0] == COOKIE[0]) &&
                (fileHeader.Cookie[1] == COOKIE[1]) )
            {
               fread( &waveformHeader, 1,
                      sizeof( WaveformHeader ), InputFile );

               switch( waveformHeader.WaveformType )
               {
                  case PB_NORMAL:
                  case PB_AVERAGE:
                     OutputNormalWaveform( waveformHeader );
                     break;
                  case PB_PEAK_DETECT:
                     OutputPeakDetectWaveform( waveformHeader );
                     break;
                  case PB_HORZ_HISTOGRAM:
                  case PB_VERT_HISTOGRAM:
                     OutputHistogramWaveform( waveformHeader );
                     break;
```

```
               default:
               case PB_UNKNOWN:
                  printf( "Unknown waveform type: %d\n" );
                  break;
            }
         }
      }
      else
      {
         printf( "Unable to open output file %s\n", OutputFile);
      }
   }
   else
   {
      printf( "Unable to open input file %s\n", argv[1] );
   }

   fclose( InputFile );
   fclose( OutputFile );
}
else
{
   printf( "Usage: bintoascii inputfile outputfile\n" );
}

}

/*********************************************************************
 * Function name:   GetTimeConversionFactors
 *  Parameters:     double xInc which is the time between consecutive
 *                     sample points.
 *                  double xOrg which is the time value of the first
 *                     data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
 ********************************************************************/
void GetTimeConversionFactors( WaveformHeader waveformHeader,
                               double *xInc, double *xOrg )
{

   /* Read values which are used to create time values */

   *xInc = waveformHeader.XIncrement;
   *xOrg = waveformHeader.XOrigin;

}

/*******************************************************************
 * Function name:  OutputNormalWaveform
 *  Parameters:    WaveformHeader *waveformHeader which is a structure
 *                 that contains the waveform header information.
 *   Return value: none
 *   Description: This routine stores the time and voltage information
 *                about the waveform as time and voltage separated by
 *                commas to a file.
 *******************************************************************/
```

```
void OutputNormalWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;
   double Time;

   BytesRead = fread( &waveformDataHeader, 1,
                      sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, MAX_LENGTH, InputFile );
      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
      {
         Time = (j * xInc) + xOrg; /* calculate time */
         j = j + 1;
         fprintf( OutputFile, "%e,%f\n", Time, Volts[i] );
      }
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
   }
}

/**********************************************************************
 * Function name: OutputHistogramWaveform
 *  Parameters:   WaveformHeader *waveformHeader which is a structure
 *                   that contains the waveform header information.
 *  Return value: none
 *  Description:  This routine stores the time and hits information
 *                as time and hits separated by commas to a file.
 *********************************************************************/
void OutputHistogramWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;

   fread( &waveformDataHeader, 1,
          sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) HistogramData, 1, MAX_LENGTH,
                         InputFile );

      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
       {
          fprintf( OutputFile, "%d,%u64l\n", j, HistogramData[i] );
          j = j + 1;
       }
```

```
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
    }
}

/*********************************************************************
 * Function name: OutputData
 *  Parameters:   FILE *PeakFile which is the pointer to the file
 *                   to be written.
 *                WaveformDataHeader waveformDataHeader
 *                   which is a structure that contains the waveform
 *                   header information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputData( FILE *PeakFile, WaveformDataHeader waveformDataHeader )
{
   int done = FALSE;
   size_t BytesRead = 0L;
   int NumberToRead;

   NumberToRead = waveformDataHeader.BufferSize;

   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, NumberToRead, InputFile ) +
                  BytesRead;

      fwrite( Volts, 1, BytesRead, PeakFile );

      if( BytesRead <= NumberToRead )
      {
         done = TRUE;
      }
   }
}

/*********************************************************************
 * Function name: OutputPeakDetectWaveform
 *  Parameters:   WaveformHeader waveformHeader which is a
 *                structure that contains the waveform header
 *                information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputPeakDetectWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
```

```
    size_t BytesRead = 0L;
    double Time;
    FILE *MaxFile;
    FILE *MinFile;

    fread( &waveformDataHeader, 1,
           sizeof( WaveformDataHeader ), InputFile );
    GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );

    MaxFile = fopen( "maxdata.bin", "wb" );
    MinFile = fopen( "mindata.bin", "wb" );

    if( MaxFile && MinFile )
    {
       if( waveformDataHeader.BufferType == PB_DATA_MAX )
       {
          OutputData( MaxFile, waveformDataHeader );
          OutputData( MinFile, waveformDataHeader );
       }
       else
       {
          OutputData( MinFile, waveformDataHeader );
          OutputData( MaxFile, waveformDataHeader );
       }

       fclose( MaxFile );
       fclose( MinFile );

       MaxFile = fopen( "maxdata.bin", "rb" );
       MinFile = fopen( "mindata.bin", "rb" );

       while( !done )
       {
          BytesRead = fread( (char *) MaxVolts, 1, MAX_LENGTH, MaxFile );
          fread( (char *) MinVolts, 1, MAX_LENGTH, MinFile );

          for( i = 0; i < BytesRead/4; i++)
          {
             Time = (j * xInc) + xOrg; /* calculate time */
             j = j + 1;
             fprintf( OutputFile, "%9.5e,%f,%f\n", Time, MinVolts[i],
                      MaxVolts[i] );
          }

          if( BytesRead < MAX_LENGTH )
          {
             done = TRUE;
          }
       }

       fclose( MaxFile );
       fclose( MinFile );
    }
}
```

Index

Symbols

Numerics

A

B

C

D

E

F

G

I

J

M

N

O

P

Q

R

S

T

W

X

Y

Z