Keysight InfiniiVision M9241/42/43A PXIe Oscilloscopes

SCPI Programmer's Guide

KEYSIGHT TECHNOLOGIES

Notices

Revision

Version 07.21.0002

Edition

October 8, 2018

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 SCPI programming with the M9241/42/43A PXIe oscilloscopes:

Table 1 InfiniiVision M9241/42/43A PXIe Oscilloscope Models, Bandwidths, Sample Rates

Bandwidth	200 MHz	500 MHz	1 GHz
Sample Rate	5 GSa/s	5 GSa/s	5 GSa/s
2-Channel DSO	M9241A	M9242A	M9243A

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

- Chapter 1, “What's New,” starting on page 35, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, “Setting Up,” starting on page 45, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, “Getting Started,” starting on page 51, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 63, describes these command types and how they affect the oscilloscope and synchronization.
- Chapter 5, “Commands Quick Reference,” starting on page 65, is a brief listing of the M9241/42/43A PXIe oscilloscope commands and syntax.

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

- Chapter 6, “Common (*) Commands,” starting on page 173, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- Chapter 7, “Root (:) Commands,” starting on page 201, describes commands that reside at the root level of the command tree and control many of the basic functions of the oscilloscope.
- Chapter 8, “:ACQuire Commands,” starting on page 241, describes commands for setting the parameters used when acquiring and storing data.
- Chapter 9, “:CALibrate Commands,” starting on page 259, describes utility commands for determining the state of the calibration factor protection button.
- Chapter 10, “:CHANnel<n> Commands,” starting on page 271, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- Chapter 11, “:COUNter Commands,” starting on page 295, describes commands that control the counter analysis feature.

- Chapter 12, ":DEMO Commands," starting on page 307, describes commands that control the education kit demonstration signals that can be output on the oscilloscope's Demo 1 and Demo 2 terminals.
- Chapter 13, ":DISPlay Commands," starting on page 311, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- Chapter 14, ":DVM Commands," starting on page 333, describes commands that control the digital voltmeter analysis feature.
- Chapter 15, ":EXTernal Trigger Commands," starting on page 339, describes commands that control the input characteristics of the external trigger input.
- Chapter 16, ":FRANalysis Commands," starting on page 345, describes commands that control oscilloscope functions associated with the Frequency Response Analysis (FRA) feature, which is available is with the license-enabled built-in waveform generator).
- Chapter 17, ":FUNCtion<m> Commands," starting on page 361, describes commands that control math waveforms.
- Chapter 18, ":HCOPY Commands," starting on page 403, describes commands that set and query the selection of hardcopy device and formatting options.
- Chapter 19, ":LISTer Commands," starting on page 407, describes commands that turn on/off the Lister display for decoded serial data and get the Lister data.
- Chapter 20, ":MARKer Commands," starting on page 411, describes commands that set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
- Chapter 21, ":MEASure Commands," starting on page 433, describes commands that select automatic measurements (and control markers).
- Chapter 22, ":MEASure Power Commands," starting on page 519, describes measurement commands that are available when the DSOX4PWR power measurements and analysis application is licensed and enabled.
- Chapter 23, ":MTESt Commands," starting on page 543, describes commands that control the mask test features provided with Option LMT.
- Chapter 24, ":POWer Commands," starting on page 577, describes commands that control the DSOX4PWR power measurement application.
- Chapter 25, ":RECall Commands," starting on page 681, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- Chapter 26, ":SAVE Commands," starting on page 691, describes commands that save oscilloscope setups, screen images, and data.
- Chapter 27, ":SBUS<n> Commands," starting on page 721, describes commands that control oscilloscope functions associated with the serial decode bus and serial triggering.

- Chapter 28, ":SEARch Commands," starting on page 933, describes commands that control oscilloscope functions associated with searching for waveform events.
- Chapter 29, ":SYSTem Commands," starting on page 1011, describes commands that control basic system functions of the oscilloscope.
- Chapter 30, ":TIMebase Commands," starting on page 1039, describes commands that control all horizontal sweep functions.
- Chapter 31, ":TRIGger Commands," starting on page 1053, describes commands that control the trigger modes and parameters for each trigger type.
- Chapter 32, ":WAVeform Commands," starting on page 1157, describes commands that provide access to waveform data.
- Chapter 33, ":WGEN<w> Commands," starting on page 1193, describes commands that control waveform generator (Option WGN) functions and parameters.
- Chapter 34, ":WMEMory<r> Commands," starting on page 1239, describes commands that control reference waveforms.
- Chapter 35, "Obsolete and Discontinued Commands," starting on page 1249, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 36, "Error Messages," starting on page 1293, lists the instrument error messages that can occur while programming the oscilloscope.

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

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

- Chapter 37, "Status Reporting," starting on page 1301, describes the oscilloscope's status registers and how to check the status of the instrument.
- Chapter 38, "Synchronizing Acquisitions," starting on page 1333, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- Chapter 39, "More About Oscilloscope Commands," starting on page 1353, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 40, "Programming Examples," starting on page 1363.

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 82350B GPIB interface).

- For information on oscilloscope front-panel operation, see the *Soft Front Panel 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/manuals/M9241A

Contents

22 :MEASure Power Commands

23 :MTESt Commands

25 :RECall Commands

26 :SAVE Commands

27 :SBUS<n> Commands

28 :SEARch Commands

33 :WGEN<w> Commands

34 :WMEMory<r> Commands

35 Obsolete and Discontinued Commands

1 What's New

What's New in Version 7.21

New features in version 7.21 of the InfiniiVision M9241/42/43A PXIe oscilloscope software are:

- USB PD (Power Delivery) serial decode and triggering option.

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

New Commands

Command	Description
:DISPlay:MESSage:CLEar (see page 329)	Removes all user messages that are currently on screen.
:FRANalysis:FREQuency:SINGle (see page 350)	Sets the single frequency value.
:FRANalysis:TRACe (see page 357)	Specifies whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.
:POWer:CLResponse:DATA:GMARgin? (see page 588)	Returns the gain margin in dB.
:POWer:CLResponse:DATA:GMARgin:FREQuency? (see page 589)	Returns the 0° phase crossover frequency in Hz.
:POWer:CLResponse:DATA:PMARgin? (see page 590)	Returns the phase margin in degrees.
:POWer:CLResponse:DATA:PMARgin:FREQuency? (see page 591)	Returns the 0 dB gain crossover frequency in Hz.
:POWer:CLResponse:FREQuency:SINGle (see page 593)	Sets the single frequency value.
:POWer:CLResponse:TRACe (see page 599)	Specifies whether to include gain, phase, both gain and phase, or neither in the control loop response analysis results.
:POWer:PSRR:FREQuency:SINGle (see page 638)	Sets the single frequency value.
:POWer:PSRR:TRACe (see page 642)	Specifies whether to include gain (PSRR) data in the power supply rejection ratio analysis results.
:SBUS<n>:A429:BAUDrate (see page 730)	Specifies the user-defined baud rate.
:SBUS<n>:USBPd Commands (see page 921)	Commands for using the USB PD triggering and serial decode feature.

Changed Commands

Command	Differences
:RECall:WMEMory<r>:[STARt] (see page 690)	There is now a <data> option for recalling waveform data from the controller PC.
:SBUS<n>:A429:SPEed (see page 737)	The USER option is now available to select a user-defined baud rate.
:SBUS<n>:MANChester:BAUDrate (see page 831)	The minimum baud rate is changed from "2000" to "500".
:SBUS<n>:MODE (see page 725)	The USBPd mode is now available with the USB PD serial decode and triggering license.
:SBUS<n>:SENT:PPULse (see page 876)	The pause mode now, in addition to supporting pause pulses off and on, supports SENT SPC (Short PWM Code) where message events are triggered by the master.
:TIMebase:MODE (see page 1041)	The XY and ROLL modes are now available.

What's New in Version 7.10

New features in version 7.10 of the InfiniiVision M9241/42/43A PXIe oscilloscope software are:

- Adaptive persistence option.
- Random trigger holdoff mode.

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

New Commands

Command	Description
:DEMO Commands (see page 307)	Commands for outputting demo signals.
:DISPlay:GRATicule:ALABels (see page 322)	Turns graticule (grid) axis labels on or off.
:DISPlay:GRATicule:INTensity (see page 323)	Sets the graticule (grid) intensity.
:DISPlay:GRATicule:TYPE (see page 324)	Sets the graticule (grid) type.
:FRANalysis:FREQuency:MODE (see page 349)	Lets you select between the normal swept frequency response analysis or analysis at a single frequency, which can be useful when debugging.
:FRANalysis:PPDecade (see page 353)	Specifies the number of points per decade in the frequency response analysis.
:FRANalysis:WGEN:VOLTage:PROFile (see page 360)	Enables or disables the ability to specify amplitude ramping within different decades.
:HCOPY:SDUMp:DATA? (see page 404)	Reads screen image data.
:HCOPY:SDUMp:FORMat (see page 405)	Specifies the format for screen image data: 24-bit PNG, 24-bit BMP, or 8-bit BMP8bit.
:MEASure:DELay:DEFine (see page 462)	Defines slope directions and edge numbers for the :MEASure:DELay command.
:SBUS<n>:MANChester Commands (see page 828)	Commands for using the Manchester triggering and serial decode feature.
:SBUS<n>:NRZ Commands (see page 847)	Commands for using the NRZ triggering and serial decode feature.
:TRIGger:HOLDoff:MAXimum (see page 1060)	When the random trigger holdoff mode is enabled, this command specifies the maximum trigger holdoff time.
:TRIGger:HOLDoff:MINimum (see page 1061)	When the random trigger holdoff mode is enabled, this command specifies the minimum trigger holdoff time.

Command	Description
:TRIGger:HOLDoff:RANDom (see page 1062)	Enables or disables the random trigger holdoff mode.
:TRIGger:NFC:RPOLarity (see page 1098)	Enables or disables triggering on signals with "reverse" polarity.
:TRIGger:PXI:SYNC (see page 1122)	Enables or disables the sync mode (when setting up multiple M924xA oscilloscope module triggers).

Changed Commands

Command	Differences
:CALibrate:OUTPut (see page 263)	The TSOurce option can now be selected to output the raw trigger signal from the oscilloscope's trigger circuit to Trig Out.
:CHANnel<n>:PROBe:HEAD:TYPE (see page 283)	The DSMA and DSMA6 probe head types can now be selected.
:DISPlay:PERSistence (see page 330)	The ADAPtive persistence option has been added.
:FRANalysis:WGEN:VOLTage (see page 359)	The <range> option has been added to specify the initial ramp amplitude at a frequency setting. When the amplitude profile setting is on, amplitudes ramp between the settings specified for individual frequencies.
:FUNCtion<m>[:FFT]:WINDow (see page 383)	The BARTlett window is now available.
:MEASure:DELay (see page 460)	You can now specify an <edge_select_mode> with the command and query.
:OPERegister:CONDition (see page 223)	Bit 4 in the Operation Status Condition Register now shows whether the remote user interface is enabled.
:OPERegister[:EVENt] (see page 226)	Bit 4 in the Operation Status Event Register now indicates when the remote user interface has gone from a disabled state to an enabled state.
:SBUS<n>:MODE (see page 725)	The MANChester and NRZ modes are now available with the M9240NRZA Manchester/NRZ serial decode and triggering license.

Version 7.00 at Introduction

The Keysight InfiniiVision M9241/42/43A PXIe oscilloscopes were introduced with version 7.00 of oscilloscope operating software.

The command set is most closely related to the InfiniiVision 3000T X-Series oscilloscopes (and the 4000 X-Series, 3000 X-Series, 7000A/B Series, 6000 Series, and 54620/54640 Series oscilloscopes before them). For more information, see "Command Differences From 3000T X-Series Oscilloscopes" on page 41.

Command Differences From 3000T X-Series Oscilloscopes

The Keysight InfiniiVision M9241/42/43A PXIe oscilloscopes command set is most closely related to the InfiniiVision 3000T X-Series oscilloscopes (and the 4000 X-Series, 3000 X-Series, 7000A/B Series, 6000 Series, and 54620/54640 Series oscilloscopes before them).

The main differences between the version 7.00 programming command set for the InfiniiVision M9241/42/43A PXIe oscilloscopes and the 4.08 programming command set for the InfiniiVision 3000T X-Series oscilloscopes are related to:

- Two analog channels in a module (3000T X-Series oscilloscopes can have two or four analog channels).
- No digital channels (or pods, buses, or BTIMing or BSTate math functions).
- No demo signals.
- Adds 10 MHz REF connector (similar to 4000 X-Series oscilloscopes).
- Adds precision measurements and math functions feature.
- Adds commands for setting up coordinated triggers between multiple PXIe oscilloscope modules in a chassis.
- No USB triggering.
- No XY or Roll timebase modes.
- Transparent backgrounds for dialog boxes are not available.
- No dedicated FFT function, but you can choose FFT math function operations.
- Adds FFT Phase math function (to previous FFT Magnitude math function).
- FFT detectors are added.
- FFT measurements are added: Adjacent Channel Power Ratio (ACPR), Channel Power, Occupied Bandwidth, and Total Harmonic Distortion.
- No LINE source for edge trigger.
- No external input on pattern trigger (just the two analog channels).
- Single-lane decoding supported for IIC and UART/RS-232.
- No support for I2S, SPI, or FlexRay serial decode and triggering.
- Internal file system locations are now on the PXIe chassis controller PC (instead of /Agilent Flash/).
- Frequency Response Analysis feature is added.
- No printer setup (:HARDcopy comand) from within the remote interface.
- No EXTernal source allowed for serial bus sources (as compared to DSO models of the 3000T X-Series).
- System dates and times are set by the PXIe chassis controller PC.

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

New Commands

Command	Description
:ACQuire:RSIGnal (see page 250)	There is a 10 MHz REF connector on the M9241/42/43A PXIe oscilloscopes.
:FRANalysis Commands (see page 345)	Commands for using the Frequency Response Analysis feature.
:FUNCtion<m>[:FFT]:BSIZe? (see page 370)	Returns the Bin Size setting for the FFT.
:FUNCtion<m>[:FFT]:DETection :POINts (see page 372)	Specifies the maximum number of points that the FFT detector should decimate to.
:FUNCtion<m>[:FFT]:DETection :TYPE (see page 373)	Sets the FFT detector decimation type.
:FUNCtion<m>[:FFT]:RBWidth? (see page 378)	Returns the Resolution Bandwidth setting for the FFT.
:FUNCtion<m>[:FFT]:READout<n> (see page 379)	For the FFT, selects from these types of readouts: Resolution Bandwidth, Bin Size, or Sample Rate.
:FUNCtion<m>[:FFT]:SRATe? (see page 381)	Returns the Sample Rate setting for the FFT.
:MEASure:FFT:ACPR (see page 471)	Installs an FFT analysis Adjacent Channel Power Ratio (ACPR) measurement on screen or returns the measured value.
:MEASure:FFT:CPOWer (see page 472)	Installs an FFT analysis Channel Power measurement on screen or returns the measured value.
:MEASure:FFT:OBW (see page 473)	Installs an FFT analysis Occupied Bandwidth measurement on screen or returns the measured value.
:MEASure:FFT:THD (see page 474)	Installs an FFT analysis Total Harmonic Distortion measurement on screen or returns the measured value.
:SYSTem:GUI:SHOW (see page 1017)	Shows or hides the Soft Front Panel (SFP) user interface.
:SYSTem:PRECision (see page 1021)	Enables or disables the precision measurements and math functions feature.
:SYSTem:PRECision:LENGth (see page 1022)	Specifies the length of the precision analysis record.
:TIMebase:REFClock (see page 1044)	There is a 10 MHz REF connector on the M9241/42/43A PXIe oscilloscopes.
:TRIGger:PXI Commands (see page 1116)	Commands for setting up coordinated triggers between multiple PXIe oscilloscope modules in a chassis.

Changed Commands

Command	Differences From InfiniiVision 3000T X-Series Oscilloscopes
:CALibrate:OUTPut (see page 263)	The OFF option is now available.
:DISPlay:ANNotation<n>:BACKground (see page 315)	The TRANsparent option is not available.
:DISPlay:DATA? (see page 321)	The GRAYscale palette option is not available. Also, the background color invert is not available because there is no :HARDcopy:INKSaver command.
:FUNCtion<m>:OPERation (see page 390)	The BTIMing and BSTate operations are not available.
:SBUS<n>:CAN:SOURce (see page 761)	The EXTernal trigger input source is not available.
:SBUS<n>:IIC[:SOURce]:CLOCk (see page 794)	The EXTernal trigger input source is not available.
:SBUS<n>:IIC[:SOURce]:DATA? (see page 795)	The EXTernal trigger input source is not available.
:SBUS<n>:LIN:SOURce (see page 808)	The EXTernal trigger input source is not available.
:SBUS<n>:MODE (see page 725)	The FLEXray, I2S, and SPI modes are not available.
:SBUS<n>:SENT:SOURce (see page 889)	The EXTernal trigger input source is not available.
:SBUS<n>:UART:SOURce:RX (see page 912)	The EXTernal trigger input source is not available.
:SBUS<n>:UART:SOURce:TX (see page 913)	The EXTernal trigger input source is not available.
:SYSTem:DATE (see page 1014)	The command for setting the date is no longer available, but the query for getting the date is still available.
:SYSTem:TIME (see page 1036)	The command for setting the time is no longer available, but the query for getting the time is still available.
:SYSTem:RLOGger:TRANsparent (see page 1032)	OFF is the only valid option.
:TIMebase:MODE (see page 1041)	The XY and ROLL modes are not available.
:TRIGger:MODE (see page 1066)	The USB mode is not available.
:TRIGger[:EDGE]:SOURce (see page 1086)	The LINE source is not available.

Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:BUS Commands	none	There are no digital channels in the M9241/42/43A PXIe oscilloscopes.
:DEMO Commands	none	There are no demo signals output by the M9241/42/43A PXIe oscilloscopes.
:DIGital Commands	none	There are no digital channels in the M9241/42/43A PXIe oscilloscopes.
:FFT Commands	:FUNCtion<m>:OPERation (see page 390) FFT	There is no dedicated FFT function, but you can choose FFT math function operations.
:FUNCtion<m>:BUS:CLOCk :FUNCtion<m>:BUS:SLOPe :FUNCtion<m>:BUS:YINCrement :FUNCtion<m>:BUS:YORigin :FUNCtion<m>:BUS:YUNits	none	There are no digital channels in the M9241/42/43A PXIe oscilloscopes.
:FUNCtion<m>:TRENd:MEASurement	:FUNCtion<m>:TRENd:NMEasurement (see page 401)	The current command specifies the number of an installed measurement instead of a measurement name.
:HARDcopy Commands	none	Printing takes place on the PXIe chassis controller PC.
:MTESt:RMODe:FACTion:PRINt	none	Printing takes place on the PXIe chassis controller PC.
:POD Commands	none	There are no digital channels in the M9241/42/43A PXIe oscilloscopes.
:PRINt	none	Printing takes place on the PXIe chassis controller PC.
:SYSTem:DATE	none	The date and time are set on the PXIe chassis controller PC.
:SYSTem:TIME	none	The date and time are set on the PXIe chassis controller PC.
:TRIGger:USB Commands	none	These commands are not available.

2 Setting Up

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

Step 1. Install Keysight IO Libraries Suite software

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

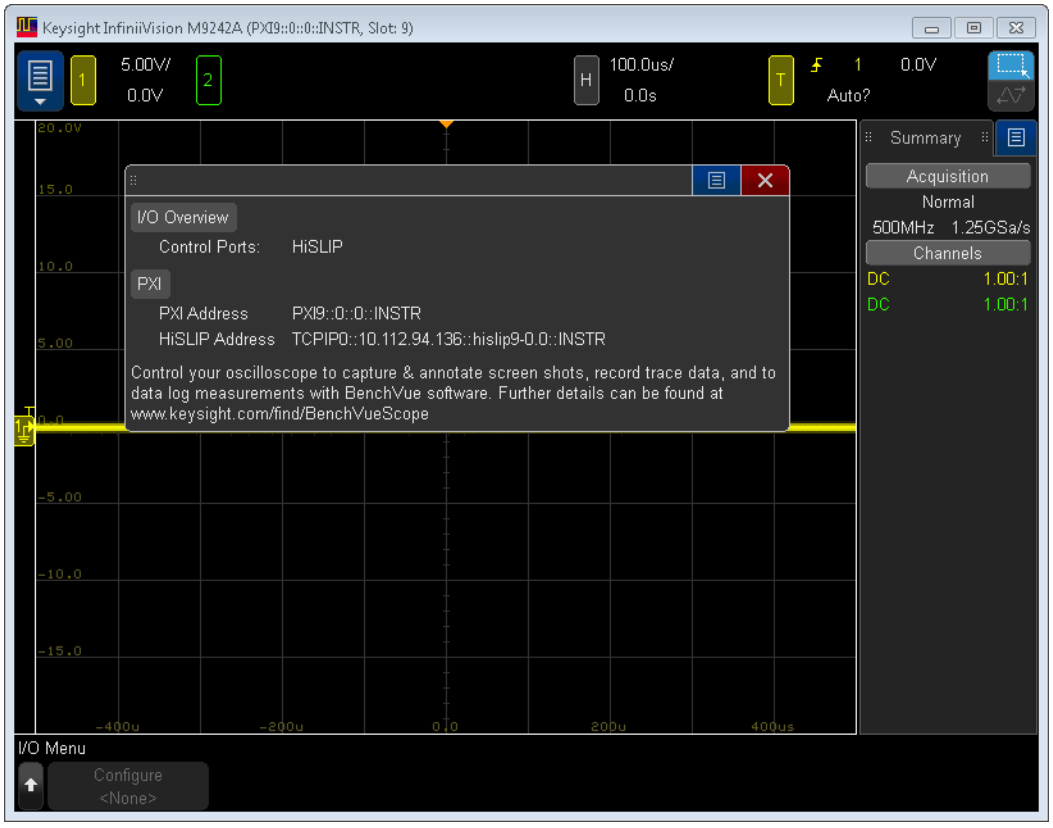
Step 2. Connect and set up the LAN interface

The PXI chassis' LAN interface is used for SCPI programming of M9241/42/43A PXIe oscilloscopes.

Refer to the PXI chassis' documentation for connecting the chassis to a LAN and setting it up.

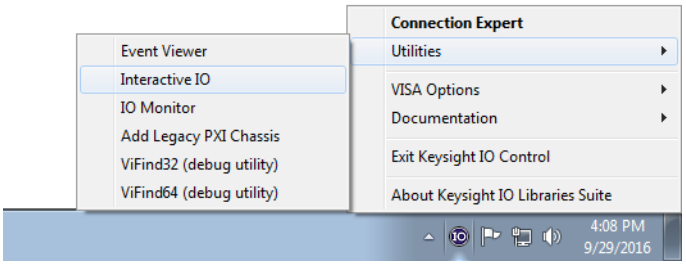
Once the PXI chassis is set up on the LAN, you can control M9241/42/43A PXIe oscilloscopes using their HiSLIP control port. To get an oscilloscope's HiSLIP (VISA) address:

1. In the oscilloscope's Front Panel graphical user interface, choose **(Menu) > Utilities > I/O Menu**.
2. In the dialog box that appears, take note of the HiSLIP address.



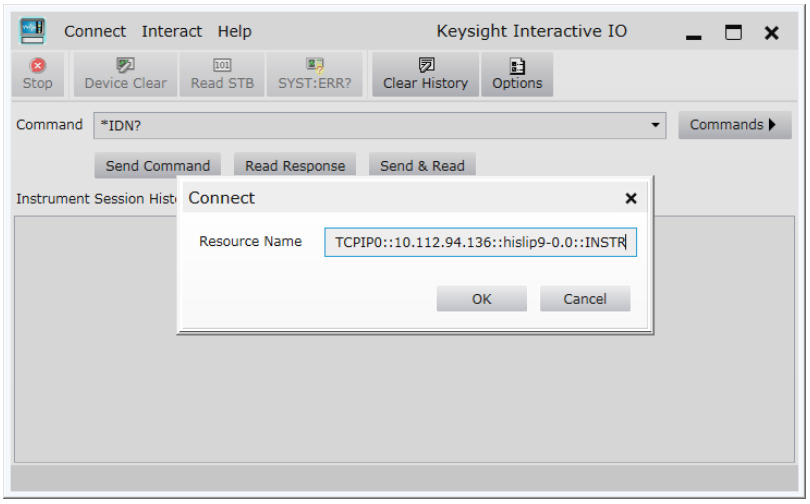
Step 3. Verify the oscilloscope connection

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

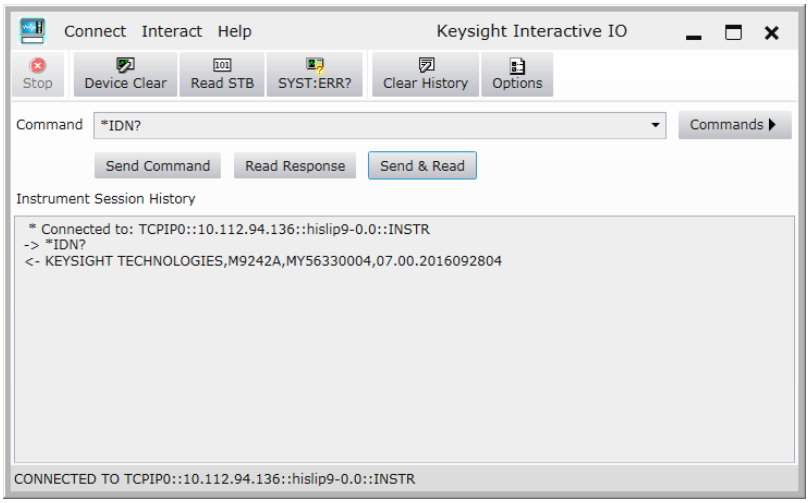


2 In the Keysight Interactive IO application, choose **Connect > Connect...**.

3 In the Connect dialog box, enter the oscilloscope's HiSLIP address into the **Resource Name** field and click **OK**.



4 Enter commands in the **Command** field and press **Send Command**, **Read Response**, or **Send & Read**.



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

3 Getting Started

This chapter gives you an overview of programming the M9241/42/43A PXIe oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

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

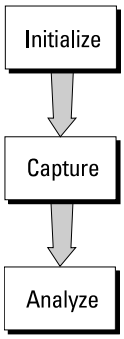
NOTE

Language for Program Examples

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

Basic Oscilloscope Program Structure

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



Initializing

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

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

Capturing Data

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

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

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

Analyzing Captured Data

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

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

Programming the Oscilloscope

Referencing the IO Library

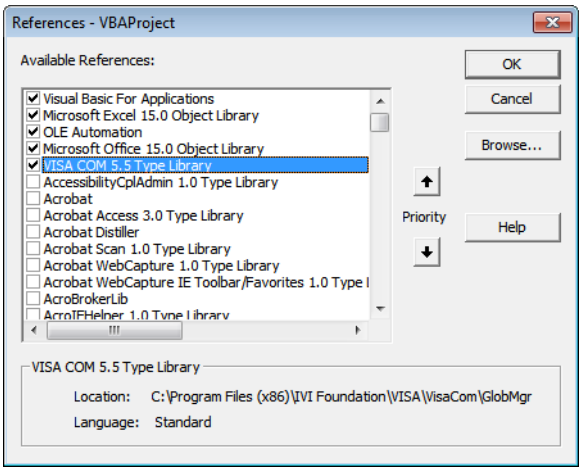
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

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

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

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

2 In the References dialog, check the "VISA COM 5.5 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 5.5 Type Library".

3 Click **OK**.

Opening the Oscilloscope Connection via the IO Library

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

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

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

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

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

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

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

The ":DISPLAY:LABEL ON" in the above example is called a *program message*. Program messages are explained in more detail in "Program Message Syntax" on page 1355.

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 6, "Common (*) Commands," starting on page 173.

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

Using :AUToscale to Automate Oscilloscope Setup

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

The syntax for the autoscale command is:

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

Using Other Oscilloscope Setup Commands

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

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

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

Example Oscilloscope Setup Code

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

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear
myScope.IO.Timeout = 10000   ' Set interface timeout to 10 seconds.

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

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

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

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

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

Capturing Data with the :DIGitize Command

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

NOTE

Ensure New Data is Collected

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

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

NOTE

Set :TIMebase:MODE to MAIN when using :DIGitize

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

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

The following program example shows a typical setup:

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

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

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

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

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

For more information, see the waveform subsystem commands and corresponding program code examples in Chapter 32, ":WAVeform Commands," starting on page 1157.

NOTE

Aborting a Digitize Operation Over the Programming Interface

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

Reading Query Responses from the Oscilloscope

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the interface to the designated listener (typically a controller).

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

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

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

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

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

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

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

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

Reading Query Results into String Variables

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

NOTE

Express String Variables Using Exact Syntax

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

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

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

After running this program, the controller displays:

Range (string): +40.0E+00

Reading Query Results into Numeric Variables

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

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

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

Definite-length block query response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 1000 bytes of data, the syntax would be:

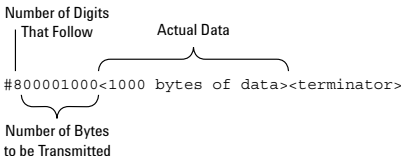


Figure 1 Definite-length block response data

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

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

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

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

Sending Multiple Queries and Reading Results

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

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

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

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

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

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

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

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

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

Checking Instrument Status

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

For more information, see Chapter 37, “Status Reporting,” starting on page 1301 which explains how to check the status of the instrument.

4 Sequential (Blocking) vs. Overlapped Commands

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

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

  These oscilloscope commands and queries are sequential (blocking):

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

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

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

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

Pausing Execution Between Overlapped Commands

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

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

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

Pausing Programs Until Sequential (Blocking) Commands are Complete

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

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

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

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

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

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

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

Using Device Clear to Abort a Sequential (Blocking) Command

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

See Also

- "*OPC (Operation Complete)" on page 185
- "*ESR (Standard Event Status Register)" on page 181
- Chapter 38, “Synchronizing Acquisitions,” starting on page 1333

5 Commands Quick Reference

Command Summary

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 178)	n/a	n/a
*ESE <mask> (see page 179)	*ESE? (see page 179)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables --- ------ ---- ---------- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 181)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 181)	KEYSIGHT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 184)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 185)	*OPC? (see page 185)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 186)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <reserved>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Frequency Response Analysis>, <Power Measurements>, <RS-232/UART Serial>, <Segmented Memory>, <Mask Test>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Educator's Kit>, <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>, <Advanced Math>, <reserved>, <reserved>, <reserved>, <reserved>, <Digital Voltmeter/Counter>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Remote Command Logging>, <reserved>, <SENT Serial>, <CAN FD Serial>, <CXPI Serial>, <NFC Trigger>, <reserved>, <reserved>, <reserved>, <Manchester/NRZ Serial>, <USB PD Serial>

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 186) (cont'd)	<All field> ::= {0 | All} <reserved> ::= 0 <Memory> ::= {0 | MEMUP} <Low Speed Serial> ::= {0 | EMBD} <Automotive Serial> ::= {0 | AUTO} <Frequency Response Analysis> ::= {0 | FRA} <Power Measurements> ::= {0 | PWR} <RS-232/UART Serial> ::= {0 | COMP} <Segmented Memory> ::= {0 | SGM} <Mask Test> ::= {0 | MASK} <Educator's Kit> ::= {0 | EDK} <Waveform Generator> ::= {0 | WAVEGEN} <MIL-1553/ARINC 429 Serial> ::= {0 | AERO} <Extended Video> ::= {0 | VID} <Advanced Math> ::= {0 | ADVMATH} <Digital Voltmeter/Counter> ::= {0 | DVMCTR} <Remote Command Logging> ::= {0 | RML} <SENT Serial> ::= {0 | SENSOR} <CAN FD Serial> ::= {0 | CANFD} <CXPI Serial> ::= {0 | CXPI} <NFC Trigger> ::= {0 | NFC} <Manchester/NRZ Serial> ::= {0 | NRZ} <USB PD Serial> ::= {0 | USBPD}
*RCL <value> (see page 188)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 189)	n/a	See *RST (Reset) (see page 189)
*SAV <value> (see page 192)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*SRE <mask> (see page 193)	*SRE? (see page 194)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables --- ------ ---- ---------- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger
n/a	*STB? (see page 195)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following: Bit Weight Name "1" Indicates --- ------ ---- --------------- 7 128 OPER Operation status condition occurred. 6 64 RQS/ MSS Instrument is requesting service. 5 32 ESB Enabled event status condition occurred. 4 16 MAV Message available. 3 8 ---- (Not used.) 2 4 MSG Message displayed. 1 2 USR User event condition occurred. 0 1 TRG A trigger occurred.
*TRG (see page 197)	n/a	n/a
n/a	*TST? (see page 198)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 199)	n/a	n/a

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see page 204)	{0 | 1}; an integer in NR1 format
:AUTOscale [<source>[,..,<source>]] (see page 205)	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format
:AUTOscale:AMODE <value> (see page 207)	:AUTOscale:AMODE? (see page 207)	<value> ::= {NORMal | CURRent}}
:AUTOscale:CHANnels <value> (see page 208)	:AUTOscale:CHANnels? (see page 208)	<value> ::= {ALL | DISPlayed}}
:AUTOscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 209)	:AUTOscale:FDEBug? (see page 209)	{0 | 1}
:BLANk [<source>] (see page 210)	n/a	<source> ::= {CHANnel<n>} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:DIGitize [<source>[,..,<source>]] (see page 211)	n/a	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:HWEenable <n> (see page 213)	:HWEenable? (see page 213)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDition? (see page 215)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENt]? (see page 216)	<n> ::= 16-bit integer in NR1 format
:MTEenable <n> (see page 217)	:MTEenable? (see page 217)	<n> ::= 16-bit integer in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MTERegister[:EVENt]? (see page 219)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 221)	:OPEE? (see page 222)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister:CONDition? (see page 223)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister[:EVENt]? (see page 226)	<n> ::= 15-bit integer in NR1 format
:OVLenable <mask> (see page 229)	:OVLenable? (see page 230)	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input --- ------ ---------- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see page 231)	<value> ::= integer in NR1 format. See OVLenable for <value>
:RUN (see page 233)	n/a	n/a
n/a	:SERial (see page 234)	<return value> ::= unquoted string containing serial number
:SINGle (see page 235)	n/a	n/a
n/a	:STATus? <display> (see page 236)	{0 | 1} <display> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 237)	n/a	n/a

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 238)	{0 | 1}
:VIEW <source> (see page 239)	n/a	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see page 244)	{1 | 0}
:ACQuire:COMPlete <complete> (see page 245)	:ACQuire:COMPlete? (see page 245)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 246)	:ACQuire:COUNt? (see page 246)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALias <mode> (see page 247)	:ACQuire:DAALias? (see page 247)	<mode> ::= {DISable | AUTO}
:ACQuire:MODE <mode> (see page 248)	:ACQuire:MODE? (see page 248)	<mode> ::= {RTIMe | ETIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 249)	<# points> ::= an integer in NR1 format
:ACQuire:RSIGnal <ref_signal_mode> (see page 250)	:ACQuire:RSIGnal? (see page 250)	<ref_signal_mode> ::= {OFF | OUT | IN}
:ACQuire:SEGMented:ANALyze (see page 251)	n/a	n/a (with SGM license)
:ACQuire:SEGMented:COUNt <count> (see page 252)	:ACQuire:SEGMented:COUNt? (see page 252)	<count> ::= an integer from 2 to 1000 in NR1 format (with SGM license)
:ACQuire:SEGMented:INDex <index> (see page 253)	:ACQuire:SEGMented:INDex? (see page 253)	<index> ::= an integer from 1 to 1000 in NR1 format (with SGM license)

Table 4 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:ACQuire:SRATe? (see page 256)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 257)	:ACQuire:TYPE? (see page 257)	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Table 5 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 261)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 262)	:CALibrate:LABel? (see page 262)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 263)	:CALibrate:OUTPut? (see page 264)	<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | WGEN2 | TSOurce} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.
n/a	:CALibrate:PROTected? (see page 265)	{"PROTected" | "UNPRotected"}
:CALibrate:STARt (see page 266)	n/a	n/a
n/a	:CALibrate:STATus? (see page 267)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see page 268)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 269)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 6 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 275)	:CHANnel<n>:BWLimit? (see page 275)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 276)	:CHANnel<n>:COUPling? (see page 276)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 277)	:CHANnel<n>:DISPlay? (see page 277)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 278)	:CHANnel<n>:IMPedance? (see page 278)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 279)	:CHANnel<n>:INVert? (see page 279)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 280)	:CHANnel<n>:LABel? (see page 280)	<string> ::= any series of 32 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 281)	:CHANnel<n>:OFFSet? (see page 281)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 282)	:CHANnel<n>:PROBe? (see page 282)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see page 283)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 283)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 284)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format

Table 6 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:MMODel <value> (see page 285)	:CHANnel<n>:PROBe:MMODel? (see page 285)	<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246 | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:RSENse <value> (see page 286)	:CHANnel<n>:PROBe:RSENse? (see page 286)	<value> ::= Ohms in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 287)	:CHANnel<n>:PROBe:SKEW? (see page 287)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 288)	:CHANnel<n>:PROBe:STYPe? (see page 288)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:ZOOM {{0 | OFF} | {1 | ON}} (see page 289)	:CHANnel<n>:PROBe:ZOOM? (see page 289)	<setting> ::= {0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 290)	:CHANnel<n>:PROTection? (see page 290)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 291)	:CHANnel<n>:RANGe? (see page 291)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 292)	:CHANnel<n>:SCALe? (see page 292)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format

Table 6 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:UNITs <units> (see page 293)	:CHANnel<n>:UNITs? (see page 293)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 294)	:CHANnel<n>:VERNier? (see page 294)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Table 7 :COUNter Commands Summary

Command	Query	Options and Query Returns
n/a	:COUNter:CURRent? (see page 297)	<value> ::= current counter value in NR3 format
:COUNter:ENABle {{0 | OFF} | {1 | ON}} (see page 298)	:COUNter:ENABle? (see page 298)	{0 | 1}
:COUNter:MODE <mode> (see page 299)	:COUNter:MODE (see page 299)	<mode> ::= {FREQuency | PERiod | TOTalize}
:COUNter:NDIGits <value> (see page 300)	:COUNter:NDIGits (see page 300)	<value> ::= 3 to 8 in NR1 format
:COUNter:SOURce <source> (see page 301)	:COUNter:SOURce? (see page 301)	<source> ::= {CHANnel<n> | TQEVent} <n> ::= 1 to (# analog channels) in NR1 format
:COUNter:TOTalize:CLEar (see page 302)	n/a	n/a
:COUNter:TOTalize:GATE:ENABle {{0 | OFF} | {1 | ON}} (see page 303)	:COUNter:TOTalize:GATE:ENABle? (see page 303)	{0 | 1}
:COUNter:TOTalize:GATE:POLarity <polarity> (see page 304)	:COUNter:TOTalize:GATE:POLarity? (see page 304)	<polarity> ::= {{NEGative | FALLing} | {POSitive | RISing}}
:COUNter:TOTalize:GATE:SOURce <source> (see page 305)	:COUNter:TOTalize:GATE:SOURce? (see page 305)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:COUNter:TOTalize:SLOPe <slope> (see page 306)	:COUNter:TOTalize:SLOPe? (see page 306)	<slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}

Table 8 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCtion <signal> (see page 308)	:DEMO:FUNCtion? (see page 309)	<signal> ::= {SINusoid | NOISy | RINGing | SINGle | CLK | GLITch | BURSt | RUNT | TRANsition | RFBurst | LFSine | FMBurst | NFC | CXPI | ARINc | MANChester | MIL | NMONotonic | HARMonics | COUPling | KEYSight}
:DEMO:OUTPut {{0 | OFF} | {1 | ON}} (see page 310)	:DEMO:OUTPut? (see page 310)	{0 | 1}

Table 9 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation<n> {{0 | OFF} | {1 | ON}} (see page 314)	:DISPlay:ANNotation<n>? (see page 314)	{0 | 1} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:BACKground <mode> (see page 315)	:DISPlay:ANNotation<n>:BACKground? (see page 315)	<mode> ::= {OPAQue | INVerted} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:COLor <color> (see page 316)	:DISPlay:ANNotation<n>:COLor? (see page 316)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:TEXT <string> (see page 317)	:DISPlay:ANNotation<n>:TEXT? (see page 317)	<string> ::= quoted ASCII string (up to 254 characters) <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:X1Position <value> (see page 318)	:DISPlay:ANNotation<n>:X1Position? (see page 318)	<value> ::= an integer from 0 to (800 - width of annotation) in NR1 format. <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:Y1Position <value> (see page 319)	:DISPlay:ANNotation<n>:Y1Position? (see page 319)	<value> ::= an integer from 0 to (480 - height of annotation) in NR1 format. <n> ::= an integer from 1 to 4 in NR1 format.

Table 9 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:CLEar (see page 320)	n/a	n/a
n/a	:DISPlay:DATA? [<format>][,][<palette>] (see page 321)	<format> ::= {BMP | BMP8bit | PNG} <palette> ::= {COLor | GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}} (see page 322)	:DISPlay:GRATicule:ALABels? (see page 322)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:INTensity <value> (see page 323)	:DISPlay:GRATicule:INTensity? (see page 323)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:GRATicule:TYPE <type> (see page 324)	:DISPlay:GRATicule:TYPE? (see page 324)	<type> ::= {FULL | MVOLt | IRE}
:DISPlay:INTensity:WAVeform <value> (see page 325)	:DISPlay:INTensity:WAVeform? (see page 325)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 326)	:DISPlay:LABel? (see page 326)	{0 | 1}
:DISPlay:LABList <binary block> (see page 327)	:DISPlay:LABList? (see page 327)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:MENU <menu> (see page 328)	n/a	<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}
:DISPlay:MESSage:CLEar (see page 329)	n/a	n/a
:DISPlay:PERSistence <value> (see page 330)	:DISPlay:PERSistence? (see page 330)	<value> ::= {MINimum | INFinite | <time> | ADAPtive} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:SIDebar <sidebar> (see page 331)	n/a	<sidebar> ::= {SUMMary | CURSors | MEASurements | DVM | NAVigate | CONTrols | EVENts | COUNter}
:DISPlay:VECTors {1 | ON} (see page 332)	:DISPlay:VECTors? (see page 332)	1

Table 10 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARANge {{0 | OFF} | {1 | ON}} (see page 334)	:DVM:ARANge? (see page 334)	{0 | 1}
n/a	:DVM:CURRent? (see page 335)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABle {{0 | OFF} | {1 | ON}} (see page 336)	:DVM:ENABle? (see page 336)	{0 | 1}
:DVM:MODE <mode> (see page 337)	:DVM:MODE? (see page 337)	<dvm_mode> ::= {ACRMs | DC | DCRMs}
:DVM:SOURce <source> (see page 338)	:DVM:SOURce? (see page 338)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

Table 11 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 340)	:EXTernal:BWLimit? (see page 340)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 341)	:EXTernal:PROBe? (see page 341)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 342)	:EXTernal:RANGe? (see page 342)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 343)	:EXTernal:UNITs? (see page 343)	<units> ::= {VOLT | AMPere}

Table 12 :FRANalysis Commands Summary

Command	Query	Options and Query Returns
n/a	:FRANalysis:DATA? [SWEep | SINGle] (see page 347)	<binary_block> ::= comma-separated data with newlines at the end of each row
:FRANalysis:ENABle {{0 | OFF} | {1 | ON}} (see page 348)	:FRANalysis:ENABle? (see page 348)	{0 | 1}

Table 12 :FRANalysis Commands Summary (continued)

Command	Query	Options and Query Returns
:FRANalysis:FREQuency :MODE <setting> (see page 349)	:FRANalysis:FREQuency :MODE? (see page 349)	<setting> ::= {SWEep | SINGle}
:FRANalysis:FREQuency :SINGle <value>[suffix] (see page 350)	:FRANalysis:FREQuency :SINGle? (see page 350)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STARt <value>[suffix] (see page 351)	:FRANalysis:FREQuency :STARt? (see page 351)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STOP <value>[suffix] (see page 352)	:FRANalysis:FREQuency :STOP? (see page 352)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:PPDecade <value> (see page 353)	:FRANalysis:PPDecade? (see page 353)	<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:FRANalysis:RUN (see page 354)	n/a	n/a
:FRANalysis:SOURce:INPut <source> (see page 355)	:FRANalysis:SOURce:INPut? (see page 355)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:SOURce:OUTPut <source> (see page 356)	:FRANalysis:SOURce:OUTPut? (see page 356)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:TRACe <selection> (see page 357)	:FRANalysis:TRACe? (see page 357)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
:FRANalysis:WGEN:LOAD <impedance> (see page 358)	:FRANalysis:WGEN:LOAD? (see page 358)	<impedance> ::= {ONEMeg | FIFTy}

Table 12 :FRANalysis Commands Summary (continued)

Command	Query	Options and Query Returns
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>] (see page 359)	:FRANalysis:WGEN:VOLTage? [<range>] (see page 359)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 360)	:FRANalysis:WGEN:VOLTage:PROFile? (see page 360)	{0 | 1}

Table 13 :FUNCtion<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCtion<m>:AVERage:COUNt <count> (see page 367)	:FUNCtion<m>:AVERage:COUNt? (see page 367)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:CLEar (see page 368)	n/a	n/a
:FUNCtion<m>:DISPlay {{0 | OFF} | {1 | ON}} (see page 369)	:FUNCtion<m>:DISPlay? (see page 369)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:BSIZe? (see page 370)	<bin_size> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:CENTer <frequency> (see page 371)	:FUNCtion<m>[:FFT]:CENTer? (see page 371)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets> (see page 372)	:FUNCtion<m>[:FFT]:DETection:POINts? (see page 372)	<number_of_buckets> ::= an integer in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:TYPE <type> (see page 373)	:FUNCtion<m>[:FFT]:DETection:TYPE? (see page 373)	<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage} <m> ::= 1 to (# math functions) in NR1 format

Table 13 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency> (see page 374)	:FUNCtion<m>[:FFT]:FREQuency:STARt? (see page 374)	<frequency> ::= the start frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency> (see page 375)	:FUNCtion<m>[:FFT]:FREQuency:STOP? (see page 375)	<frequency> ::= the stop frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:GATE <gating> (see page 376)	:FUNCtion<m>[:FFT]:GATE? (see page 376)	<gating> ::= {NONE | ZOOM} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point> (see page 377)	:FUNCtion<m>[:FFT]:PHASe:REFerence? (see page 377)	<ref_point> ::= {TRIGger | DISPlay} <m> ::= 1-4 in NR1 format
n/a	:FUNCtion<m>[:FFT]:RBWidth? (see page 378)	<resolution_bw> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:READout<n> <readout_type> (see page 379)	:FUNCtion<m>[:FFT]:READout<n>? (see page 379)	<readout_type> ::= {SRATe | BSIZe | RBWidth} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1-2 in NR1 format, 2 is for dedicated FFT function
:FUNCtion<m>[:FFT]:SPAN (see page 380)	:FUNCtion<m>[:FFT]:SPAN? (see page 380)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:SRATe? (see page 381)	<sample_rate> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format

Table 13 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:VTYPe <units> (see page 382)	:FUNCtion<m>[:FFT]:VTYPe? (see page 382)	<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation <units> ::= {DEGRees | RADians} for the FFTPhase operation <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:WINDow <window> (see page 383)	:FUNCtion<m>[:FFT]:WINDow? (see page 383)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq> (see page 384)	:FUNCtion<m>:FREQuency:HIGHpass? (see page 384)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq> (see page 385)	:FUNCtion<m>:FREQuency:LOWPass? (see page 385)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IOFFset <input_offset> (see page 386)	:FUNCtion<m>:INTegrate:IOFFset? (see page 386)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:GAIN <value> (see page 387)	:FUNCtion<m>:LINear:GAIN? (see page 387)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:OFFSet <value> (see page 388)	:FUNCtion<m>:LINear:OFFSet? (see page 388)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:OFFSet <offset> (see page 389)	:FUNCtion<m>:OFFSet? (see page 389)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format

Table 13 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:OPERation <operation> (see page 390)	:FUNCtion<m>:OPERation? (see page 392)	<operation> ::= {ADD | SUBTract | MULTiply | DIVide | INTegrate | DIFF | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | AVERage | LINear | MAXimum | MINimum | PEAK | MAXHold | MINHold | TRENd} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:RANGe <range> (see page 394)	:FUNCtion<m>:RANGe? (see page 394)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:REFerence <level> (see page 395)	:FUNCtion<m>:REFerence? (see page 395)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SCALe <scale value>[<suffix>] (see page 396)	:FUNCtion<m>:SCALe? (see page 396)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SMOoth:POINts <points> (see page 397)	:FUNCtion<m>:SMOoth:POINts? (see page 397)	<points> ::= odd integer in NR1 format

Table 13 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:SOURce1 <source> (see page 398)	:FUNCtion<m>:SOURce1? (see page 398)	<source> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>} <n> ::= 1 to (# analog channels) in NR1 format <c> ::= {1}, must be lower than <m> <r> ::= 1 to (# ref waveforms) in NR1 format <b> ::= {1 | 2} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SOURce2 <source> (see page 400)	:FUNCtion<m>:SOURce2? (see page 400)	<source> ::= {CHANnel<n> | WMEMory<r> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:TRENd:NMEasurement MEAS<n> (see page 401)	:FUNCtion<m>:TRENd:NMEasurement? (see page 401)	<n> ::= # of installed measurement, from 1 to 8 <m> ::= 1 to (# math functions) in NR1 format

Table 14 :HCOPY Commands Summary

Command	Query	Options and Query Returns
n/a	:HCOPY:SDUMp:DATA? (see page 404)	<display_data> ::= binary block data in IEEE-488.2 # format.
:HCOPY:SDUMp:FORMat <format> (see page 405)	:HCOPY:SDUMp:FORMat? (see page 405)	<format> ::= {BMP | BMP8bit | PNG}

Table 15 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 408)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2} | ALL} (see page 409)	:LISTer:DISPlay? (see page 409)	{OFF | SBUS1 | SBUS2 | ALL}
:LISTer:REFerence <time_ref> (see page 410)	:LISTer:REFerence? (see page 410)	<time_ref> ::= {TRIGger | PREVious}

Table 16 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 414)	<return_value> ::= •Y/•X value in NR3 format
:MARKer:MODE <mode> (see page 415)	:MARKer:MODE? (see page 415)	<mode> ::= {OFF | MEASurement | MANual | WAVeform | BINary | HEX}
:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}} (see page 416)	:MARKer:X1:DISPlay? (see page 416)	<setting> ::= {0 | 1}
:MARKer:X1Position <position>[suffix] (see page 417)	:MARKer:X1Position? (see page 417)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 418)	:MARKer:X1Y1source? (see page 418)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}} (see page 419)	:MARKer:X2:DISPlay? (see page 419)	<setting> ::= {0 | 1}

Table 16 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X2Position <position>[suffix] (see page 420)	:MARKer:X2Position? (see page 420)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 421)	:MARKer:X2Y2source? (see page 421)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 422)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 423)	:MARKer:XUNits? (see page 423)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}
:MARKer:XUNits:USE (see page 424)	n/a	n/a
:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}} (see page 425)	:MARKer:Y1:DISPlay? (see page 425)	<setting> ::= {0 | 1}
:MARKer:Y1Position <position>[suffix] (see page 426)	:MARKer:Y1Position? (see page 426)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}} (see page 427)	:MARKer:Y2:DISPlay? (see page 427)	<setting> ::= {0 | 1}
:MARKer:Y2Position <position>[suffix] (see page 428)	:MARKer:Y2Position? (see page 428)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format

Table 16 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MARKer:YDELta? (see page 429)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 430)	:MARKer:YUNits? (see page 430)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 431)	n/a	n/a

Table 17 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 451)	n/a	n/a
:MEASure:AREa [<interval>][,<source>] (see page 452)	:MEASure:AREa? [<interval>][,<source>] (see page 452)	<interval> ::= {CYCLe | DISPlay} \<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BRATe [<source>] (see page 453)	:MEASure:BRATe? [<source>] (see page 453)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= bit rate in Hz, NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:BWIDth [<source>] (see page 454)	:MEASure:BWIDth? [<source>] (see page 454)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 455)	n/a	n/a
:MEASure:COUNter [<source>] (see page 456)	:MEASure:COUNter? [<source>] (see page 456)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 457)	:MEASure:DEFine? DELay (see page 459)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 457)	:MEASure:DEFine? THResholds (see page 459)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent | ABSolute}

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DELay [<source1>] [,<source2>] (see page 460)	:MEASure:DELay? [<source1>] [,<source2>] (see page 460)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>, <source1_edge_threshold>, <source2_edge_slope>, <source2_edge_number>, <source2_edge_threshold> (see page 462)	:MEASure:DELay:DEFine? (see page 462)	<source1_edge_slope>, <source2_edge_slope> ::= {RISing | FALLing} <source1_edge_number>, <source2_edge_number> ::= 0 to 1000 in NR1 format <source1_edge_threshold>, <source2_edge_threshold> ::= MIDDle
:MEASure:DUAL:CHARge [<interval>] [,<source1>] [,<source2>] (see page 463)	:MEASure:DUAL:CHARge? [<interval>] [,<source1>] [,<source2>] (see page 463)	<interval> ::= {CYCLe | DISPlay} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= area in Amp-hours, NR3 format
:MEASure:DUAL:VAMPlitude [<source1>] [,<source2>] (see page 464)	:MEASure:DUAL:VAMPlitude? [<source1>] [,<source2>] (see page 464)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUAL:VAVerage [<interval>] [,<source1>] [,<source2>] (see page 465)	:MEASure:DUAL:VAVerage? [<interval>] [,<source1>] [,<source2>] (see page 465)	<interval> ::= {CYCLe | DISPlay} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:DUAL:VBASe [<source1>] [,<source2>] (see page 466)	:MEASure:DUAL:VBASe? [<source1>] [,<source2>] (see page 466)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:DUAL:VPP [<source1>] [,<source2>] (see page 467)	:MEASure:DUAL:VPP? [<source1>] [,<source2>] (see page 467)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:DUAL:VRMS [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 468)	:MEASure:DUAL:VRMS? [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 468)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated RMS voltage in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 469)	:MEASure:DUTYcycle? [<source>] (see page 469)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 470)	:MEASure:FALLtime? [<source>] (see page 470)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FFT:ACPR <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 471)	:MEASure:FFT:ACPR? <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 471)	<chan_width> ::= width of main range and sideband channels, Hz in NR3 format <chan_spacing> ::= spacing between main range and sideband channels, Hz in NR3 format <chan> ::= {CENTer | HIGH<sb> | LOW<sb>} <sb> ::= sideband 1 to 5 <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= adjacent channel power ratio, dBV in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:CPOWer [<source>] (see page 472)	:MEASure:FFT:CPOWer? [<source>] (see page 472)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= spectral channel power, dBV in NR3 format
:MEASure:FFT:OBW <percentage>[,<source>] (see page 473)	:MEASure:FFT:OBW? <percentage>[,<source>] (see page 473)	<percentage> ::= percent of spectral power occupied bandwidth is measured for in NR3 format <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= occupied bandwidth, Hz in NR3 format
:MEASure:FFT:THD [<source>] (see page 474)	:MEASure:FFT:THD? [<source>] (see page 474)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= total harmonic distortion ratio percent in NR3 format
:MEASure:FREQuency [<source>] (see page 475)	:MEASure:FREQuency? [<source>] (see page 475)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= frequency in Hertz in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NDUTy [<source>] (see page 476)	:MEASure:NDUTy? [<source>] (see page 476)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NEDGes [<source>] (see page 477)	:MEASure:NEDGes? [<source>] (see page 477)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling edge count in NR3 format
:MEASure:NPULses [<source>] (see page 478)	:MEASure:NPULses? [<source>] (see page 478)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling pulse count in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 479)	:MEASure:NWIDth? [<source>] (see page 479)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 480)	:MEASure:OVERshoot? [<source>] (see page 480)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PEDGes [<source>] (see page 482)	:MEASure:PEDGes? [<source>] (see page 482)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising edge count in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PERiod [<source>] (see page 483)	:MEASure:PERiod? [<source>] (see page 483)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 484)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 484)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PPULses [<source>] (see page 485)	:MEASure:PPULses? [<source>] (see page 485)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising pulse count in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PREShoot [<source>] (see page 486)	:MEASure:PREShoot? [<source>] (see page 486)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 487)	:MEASure:PWIDth? [<source>] (see page 487)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 488)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 491)	:MEASure:RISetime? [<source>] (see page 491)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SDEViation [<source>] (see page 492)	:MEASure:SDEViation? [<source>] (see page 492)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 | OFF} | {1 | ON}} (see page 493)	:MEASure:SHOW? (see page 493)	{0 | 1}
:MEASure:SOURce <source1> [,<source2>] (see page 494)	:MEASure:SOURce? (see page 494)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 496)	:MEASure:STATistics? (see page 496)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 497)	:MEASure:STATistics:DISPlay? (see page 497)	{0 | 1}
:MEASure:STATistics:INCRement (see page 498)	n/a	n/a
:MEASure:STATistics:MCOunt <setting> (see page 499)	:MEASure:STATistics:MCOunt? (see page 499)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:STATistics:RESet (see page 500)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 501)	:MEASure:STATistics:RSDeviation? (see page 501)	{0 | 1}
n/a	:MEASure:TEDGe? <slope><occurrence>[,<source>] (see page 502)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>,[<slope>]<occurrence>[,<source>] (see page 504)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VAMPlitude [<source>] (see page 506)	:MEASure:VAMPlitude? [<source>] (see page 506)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<interval>][,<source>] (see page 507)	:MEASure:VAVerage? [<interval>][,<source>] (see page 507)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 508)	:MEASure:VBASe? [<source>] (see page 508)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 509)	:MEASure:VMAX? [<source>] (see page 509)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 510)	:MEASure:VMIN? [<source>] (see page 510)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 511)	:MEASure:VPP? [<source>] (see page 511)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRATio [<interval>][,<source 1>] [,<source2>] (see page 512)	:MEASure:VRATio? [<interval>][,<source 1>] [,<source2>] (see page 512)	<interval> ::= {CYCLe | DISPlay} \<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>] [,<type>][,<source>] (see page 513)	:MEASure:VRMS? [<interval>] [,<type>][,<source>] (see page 513)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 514)	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 515)	:MEASure:VTOP? [<source>] (see page 515)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 516)	:MEASure:WINDow? (see page 516)	<type> ::= {MAIN | ZOOM | AUTO | GATE}
:MEASure:XMAX [<source>] (see page 517)	:MEASure:XMAX? [<source>] (see page 517)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 518)	:MEASure:XMIN? [<source>] (see page 518)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format

Table 18 :MEASure Power Commands Summary

Command	Query	Options and Query Returns
:MEASure:ANGLe [<source1>][,<source2>] (see page 524)	:MEASure:ANGLe? [<source1>][,<source2>] (see page 524)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the power phase angle in degrees in NR3 format
:MEASure:APParent [<source1>][,<source2>] (see page 525)	:MEASure:APParent? [<source1>][,<source2>] (see page 525)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the apparent power value in NR3 format
:MEASure:CPLoss [<source1>][,<source2>] (see page 526)	:MEASure:CPLoss? [<source1>][,<source2>] (see page 526)	<source1>, <source2> <source1> ::= {FUNCtion<m> | MATH<m>} <source2> ::= {CHANnel<n>} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the switching loss per cycle watts value in NR3 format
:MEASure:CRESt [<source>] (see page 527)	:MEASure:CRESt? [<source>] (see page 527)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= the crest factor value in NR3 format
:MEASure:EFFiciency (see page 528)	:MEASure:EFFiciency? (see page 528)	<return_value> ::= percent value in NR3 format

Table 18 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:ELOSs [<source>] (see page 529)	:MEASure:ELOSs? [<source>] (see page 529)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the energy loss value in NR3 format
:MEASure:FACTor [<source1>][,<source2>] (see page 530)	:MEASure:FACTor? [<source1>][,<source2>] (see page 530)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the power factor value in NR3 format
:MEASure:IPOWer (see page 531)	:MEASure:IPOWer? (see page 531)	<return_value> ::= the input power value in NR3 format
:MEASure:OFFTime [<source1>][,<source2>] (see page 532)	:MEASure:OFFTime? [<source1>][,<source2>] (see page 532)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the time in seconds in NR3 format
:MEASure:ONTime [<source1>][,<source2>] (see page 533)	:MEASure:ONTime? [<source1>][,<source2>] (see page 533)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the time in seconds in NR3 format
:MEASure:OPOWer (see page 534)	:MEASure:OPOWer? (see page 534)	<return_value> ::= the output power value in NR3 format

Table 18 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PCURrent [<source>] (see page 535)	:MEASure:PCURrent? [<source>] (see page 535)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the peak current value in NR3 format
:MEASure:PLOSs [<source>] (see page 536)	:MEASure:PLOSs? [<source>] (see page 536)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the power loss value in NR3 format
:MEASure:RDSon [<source1>][,<source2>] (see page 537)	:MEASure:RDSon? [<source1>][,<source2>] (see page 537)	<source1>, <source2> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the Rds(on) value in NR3 format
:MEASure:REACtive [<source1>][,<source2>] (see page 538)	:MEASure:REACtive? [<source1>][,<source2>] (see page 538)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the reactive power value in NR3 format

Table 18 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:REAL [<source>] (see page 539)	:MEASure:REAL? [<source>] (see page 539)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= the real power value in NR3 format
:MEASure:RIPPle [<source>] (see page 540)	:MEASure:RIPPle? [<source>] (see page 540)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the output ripple value in NR3 format

Table 18 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:TRESponse [<source>] (see page 541)	:MEASure:TRESponse? [<source>] (see page 541)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds for the overshoot to settle back into the band in NR3 format
:MEASure:VCESat [<source>] (see page 542)	:MEASure:VCESat? [<source>] (see page 542)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the Vce(sat) value in NR3 format

Table 19 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 548)	:MTESt:ALL? (see page 548)	{0 | 1}
:MTESt:AMASk:CREate (see page 549)	n/a	n/a
:MTESt:AMASk:SOURce <source> (see page 550)	:MTESt:AMASk:SOURce? (see page 550)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 551)	:MTESt:AMASk:UNITs? (see page 551)	<units> ::= {CURRent | DIVisions}

Table 19 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:AMASk:XDELta <value> (see page 552)	:MTESt:AMASk:XDELta? (see page 552)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 553)	:MTESt:AMASk:YDELta? (see page 553)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 554)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 555)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 556)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 557)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 558)	:MTESt:DATA? (see page 558)	<mask> ::= data in IEEE 488.2 # format.
:MTESt:DELete (see page 559)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 560)	:MTESt:ENABle? (see page 560)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 561)	:MTESt:LOCK? (see page 561)	{0 | 1}
:MTESt:RMODe <rmode> (see page 562)	:MTESt:RMODe? (see page 562)	<rmode> ::= {FORever | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 563)	:MTESt:RMODe:FACTion:MEASure? (see page 563)	{0 | 1}
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 564)	:MTESt:RMODe:FACTion:SAVE? (see page 564)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 565)	:MTESt:RMODe:FACTion:STOP? (see page 565)	{0 | 1}
:MTESt:RMODe:SIGMa <level> (see page 566)	:MTESt:RMODe:SIGMa? (see page 566)	<level> ::= from 0.1 to 9.3 in NR3 format

Table 19 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:RMODe:TIME <seconds> (see page 567)	:MTESt:RMODe:TIME? (see page 567)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 568)	:MTESt:RMODe:WAVeforms? (see page 568)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 569)	:MTESt:SCALe:BIND? (see page 569)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 570)	:MTESt:SCALe:X1? (see page 570)	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALe:XDELta <xdelta_value> (see page 571)	:MTESt:SCALe:XDELta? (see page 571)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 572)	:MTESt:SCALe:Y1? (see page 572)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 573)	:MTESt:SCALe:Y2? (see page 573)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 574)	:MTESt:SOURce? (see page 574)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 575)	<title> ::= a string of up to 128 ASCII characters

Table 20 :POWer Commands Summary

Command	Query	Options and Query Returns
n/a	:POWer:CLResponse? (see page 585)	n/a
:POWer:CLResponse:APPLy (see page 586)	n/a	n/a
n/a	:POWer:CLResponse:DATA? [SWEep | SINGle] (see page 587)	<binary_block> ::= comma-separated data with newlines at the end of each row

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:POWer:CLResponse:DATA:GMARgin? (see page 588)	<gain_margin> ::= gain margin in dB in NR3 format.
n/a	:POWer:CLResponse:DATA:GMARgin:FREQuency? (see page 589)	<frequency> ::= 0 degrees phase crossover frequency in Hz in NR3 format
n/a	:POWer:CLResponse:DATA:PMARgin? (see page 590)	<phase_margin> ::= phase margin in degrees in NR3 format.
n/a	:POWer:CLResponse:DATA:PMARgin:FREQuency? (see page 591)	<frequency> ::= 0dB gain crossover frequency in Hz in NR3 format.
:POWer:CLResponse:FREQuency:MODE <mode> (see page 592)	:POWer:CLResponse:FREQuency:MODE? (see page 592)	<mode> ::= {SWEep | SINGle}
:POWer:CLResponse:FREQuency:SINGle <value>[suffix] (see page 593)	:POWer:CLResponse:FREQuency:SINGle? (see page 593)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:CLResponse:FREQuency:STARt <value>[suffix] (see page 594)	:POWer:CLResponse:FREQuency:STARt? (see page 594)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:CLResponse:FREQuency:STOP <value>[suffix] (see page 595)	:POWer:CLResponse:FREQuency:STOP? (see page 595)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:CLResponse:PPDecade <pts> (see page 596)	:POWer:CLResponse:PPDecade? (see page 596)	<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:POWer:CLResponse:SOURce:INPut <source> (see page 597)	:POWer:CLResponse:SOURce:INPut? (see page 597)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:CLResponse:SOURce:OUTPut <source> (see page 598)	:POWer:CLResponse:SOURce:OUTPut? (see page 598)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:CLResponse:TRACe <selection> (see page 599)	:POWer:CLResponse:TRACe? (see page 599)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:CLResponse:WGEN:LOAD <impedance> (see page 600)	:POWer:CLResponse:WGEN:LOAD? (see page 600)	<impedance> ::= {ONEMeg | FIFTy}
:POWer:CLResponse:WGEN:VOLTage <amplitude>[,<range>] (see page 601)	:POWer:CLResponse:WGEN:VOLTage? [<range>] (see page 601)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:POWer:CLResponse:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 602)	:POWer:CLResponse:WGEN:VOLTage:PROFile? (see page 602)	{0 | 1}
:POWer:DESKew (see page 603)	n/a	n/a
:POWer:EFFiciency:APPLy (see page 604)	n/a	n/a
:POWer:EFFiciency:TYPE <type> (see page 605)	:POWer:EFFiciency:TYPE? (see page 605)	<type> ::= {DCDC | DCAC | ACDC | ACAC}
:POWer:ENABle {{0 | OFF} | {1 | ON}} (see page 606)	:POWer:ENABle? (see page 606)	{0 | 1}
:POWer:HARMonics:APPLy (see page 607)	n/a	n/a
n/a	:POWer:HARMonics:DATA? (see page 608)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISPlay <display> (see page 609)	:POWer:HARMonics:DISPlay? (see page 609)	<display> ::= {TABLe | BAR | OFF}
n/a	:POWer:HARMonics:FAILcount? (see page 610)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 611)	:POWer:HARMonics:LINE? (see page 611)	<frequency> ::= {F50 | F60 | F400 | AUTO}
n/a	:POWer:HARMonics:POWerfactor? (see page 612)	<value> ::= Class C power factor in NR3 format

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:HARMonics:RPOWer <source> (see page 613)	:POWer:HARMonics:RPOWer? (see page 613)	<source> ::= {MEASured | USER}
:POWer:HARMonics:RPOWer:USER <value> (see page 614)	:POWer:HARMonics:RPOWer:USER? (see page 614)	<value> ::= Watts from 1.0 to 600.0 in NR3 format
n/a	:POWer:HARMonics:RUNCount? (see page 615)	<count> ::= integer in NR1 format
:POWer:HARMonics:STANDard <class> (see page 616)	:POWer:HARMonics:STANDard? (see page 616)	<class> ::= {A | B | C | D}
n/a	:POWer:HARMonics:STATus? (see page 617)	<status> ::= {PASS | FAIL | UNTested}
n/a	:POWer:HARMonics:THD? (see page 618)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLy (see page 619)	n/a	n/a
:POWer:INRush:EXIT (see page 620)	n/a	n/a
:POWer:INRush:NEXT (see page 621)	n/a	n/a
:POWer:ITYPe <type> (see page 622)	:POWer:ITYPe? (see page 622)	<type> ::= {DC | AC}
:POWer:MODulation:APPLy (see page 623)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 624)	:POWer:MODulation:SOURce? (see page 624)	<source> ::= {V | I}
:POWer:MODulation:TYPE <modulation> (see page 625)	:POWer:MODulation:TYPE? (see page 625)	<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDith | NWIDth | DUTYcycle | RISetime | FALLtime}
:POWer:ONOFf:APPLy (see page 626)	n/a	n/a
:POWer:ONOFf:EXIT (see page 627)	n/a	n/a
:POWer:ONOFf:NEXT (see page 628)	n/a	n/a

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:ONOFf:TEST {{0 | OFF} | {1 | ON}} (see page 629)	:POWer:ONOFf:TEST? (see page 629)	{0 | 1}
:POWer:ONOFf:THResholds <type>, <input_thr>, <output_thr> (see page 630)	:POWer:ONOFf:THResholds? <type> (see page 630)	<type> ::= {0 | 1} <input_thr> ::= percent from 0-100 in NR1 format <output_thr> ::= percent from 0-100 in NR1 format
n/a	:POWer:PSRR? (see page 632)	n/a
:POWer:PSRR:APPLy (see page 633)	n/a	n/a
n/a	:POWer:PSRR:DATA? [SWEep | SINGle] (see page 634)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:PSRR:FREQuency:MAXimum <value>[suffix] (see page 635)	:POWer:PSRR:FREQuency:MAXimum? (see page 635)	<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:FREQuency:MINimum <value>[suffix] (see page 636)	:POWer:PSRR:FREQuency:MINimum? (see page 636)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:FREQuency:MODE <mode> (see page 637)	:POWer:PSRR:FREQuency:MODE? (see page 637)	<mode> ::= {SWEep | SINGle}
:POWer:PSRR:FREQuency:SINGle <value>[suffix] (see page 638)	:POWer:PSRR:FREQuency:SINGle? (see page 638)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:PPDecade <pts> (see page 639)	:POWer:PSRR:PPDecade? (see page 639)	<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:POWer:PSRR:SOURce:INPut <source> (see page 640)	:POWer:PSRR:SOURce:INPut? (see page 640)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:PSRR:SOURce:OUTPut <source> (see page 641)	:POWer:PSRR:SOURce:OUTPut? (see page 641)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:TRACe <selection> (see page 642)	:POWer:PSRR:TRACe? (see page 642)	<selection> ::= {NONE | GAIN}
:POWer:PSRR:WGEN:LOAD <impedance> (see page 643)	:POWer:PSRR:WGEN:LOAD? (see page 643)	<impedance> ::= {ONEMeg | FIFTy}
:POWer:PSRR:WGEN:VOLTage <amplitude>[,<range>] (see page 644)	:POWer:PSRR:WGEN:VOLTage? [<range>] (see page 644)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:POWer:PSRR:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 645)	:POWer:PSRR:WGEN:VOLTage:PROFile? (see page 645)	{0 | 1}
:POWer:QUALity:APPLy (see page 646)	n/a	n/a
:POWer:RIPPle:APPLy (see page 647)	n/a	n/a
:POWer:SIGNals:AUTosetup <analysis> (see page 648)	n/a	<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity | SLEW | SWITch | RDSVce}
:POWer:SIGNals:CYCLes:HARMonics <count> (see page 649)	:POWer:SIGNals:CYCLes:HARMonics? (see page 649)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:CYCLes:QUALity <count> (see page 650)	:POWer:SIGNals:CYCLes:QUALity? (see page 650)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURation:EFFiciency <value>[suffix] (see page 651)	:POWer:SIGNals:DURation:EFFiciency? (see page 651)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:MODulation <value>[suffix] (see page 652)	:POWer:SIGNals:DURation:MODulation? (see page 652)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:ONOFf:OFF <value>[suffix] (see page 653)	:POWer:SIGNals:DURation:ONOFf:OFF? (see page 653)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:DURation:ONOFf:ON <value>[suffix] (see page 654)	:POWer:SIGNals:DURation:ONOFf:ON? (see page 654)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:RIPPle <value>[suffix] (see page 655)	:POWer:SIGNals:DURation:RIPPle? (see page 655)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:TRANsient <value>[suffix] (see page 656)	:POWer:SIGNals:DURation:TRANsient? (see page 656)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:IEXPected <value>[suffix] (see page 657)	:POWer:SIGNals:IEXPected? (see page 657)	<value> ::= Expected current value in NR3 format [suffix] ::= {A | mA}
:POWer:SIGNals:OVERshoot <percent> (see page 658)	:POWer:SIGNals:OVERshoot? (see page 658)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V | mV}}
:POWer:SIGNals:VMAXimum:INRush <value>[suffix] (see page 659)	:POWer:SIGNals:VMAXimum:INRush? (see page 659)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VMAXimum:ONOFf:OFF <value>[suffix] (see page 660)	:POWer:SIGNals:VMAXimum:ONOFf:OFF? (see page 660)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VMAXimum:ONOFf:ON <value>[suffix] (see page 661)	:POWer:SIGNals:VMAXimum:ONOFf:ON? (see page 661)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady:ONOFf:OFF <value>[suffix] (see page 662)	:POWer:SIGNals:VSTeady:ONOFf:OFF? (see page 662)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady:ONOFf:ON <value>[suffix] (see page 663)	:POWer:SIGNals:VSTeady:ONOFf:ON? (see page 663)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:VSTeady:TRANsient <value>[suffix] (see page 664)	:POWer:SIGNals:VSTeady:TRANsient? (see page 664)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:SOURce:CURRent<i> <source> (see page 665)	:POWer:SIGNals:SOURce:CURRent<i>? (see page 665)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SIGNals:SOURce:VOLTage<i> <source> (see page 666)	:POWer:SIGNals:SOURce:VOLTage<i>? (see page 666)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SLEW:APPLy (see page 667)	n/a	n/a
:POWer:SLEW:SOURce <source> (see page 668)	:POWer:SLEW:SOURce? (see page 668)	<source> ::= {V | I}
:POWer:SWITch:APPLy (see page 669)	n/a	n/a
:POWer:SWITch:CONDuction <conduction> (see page 670)	:POWer:SWITch:CONDuction? (see page 670)	<conduction> ::= {WAVeform | RDS | VCE}
:POWer:SWITch:IREFerence <percent> (see page 671)	:POWer:SWITch:IREFerence? (see page 671)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 672)	:POWer:SWITch:RDS? (see page 672)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM | mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 673)	:POWer:SWITch:VCE? (see page 673)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V | mV}
:POWer:SWITch:VREFerence <percent> (see page 674)	:POWer:SWITch:VREFerence? (see page 674)	<percent> ::= percent in NR1 format
:POWer:TRANsient:APPLy (see page 675)	n/a	n/a
:POWer:TRANsient:EXIT (see page 676)	n/a	n/a

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:TRANsient:IINitial <value>[suffix] (see page 677)	:POWer:TRANsient:IINitial? (see page 677)	<value> ::= Initial current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:INEW <value>[suffix] (see page 678)	:POWer:TRANsient:INEW? (see page 678)	<value> ::= New current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:NEXT (see page 679)	n/a	n/a

Table 21 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[STARt] [<file_spec>][,<column>][,<wavegen_id>] (see page 683)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <column> ::= Column in CSV file to load. Column number starts from 1. <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= WGEN1
:RECall:DBC[:STARt] [<file_name>] [,<serialbus>] (see page 684)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".dbc". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:FILename <base_name> (see page 685)	:RECall:FILename? (see page 685)	<base_name> ::= quoted ASCII string
:RECall:LDF[:STARt] [<file_name>] [,<serialbus>] (see page 686)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".ldf". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format

Table 21 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:MASK[:STARt] [<file_spec>] (see page 687)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:PWD <path_name> (see page 688)	:RECall:PWD? (see page 688)	<path_name> ::= quoted ASCII string
:RECall:SETup[:STARt] [<file_spec>] (see page 689)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>] (see page 690)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5". <data> ::= binary block data in IEEE 488.2 # format

Table 22 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [<file_spec>][, <wavegen_id>] (see page 695)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= WGEN1
:SAVE:FILename <base_name> (see page 696)	:SAVE:FILename? (see page 696)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 697)	n/a	<file_name> ::= quoted ASCII string

Table 22 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 698)	:SAVE:IMAGe:FACTors? (see page 698)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 699)	:SAVE:IMAGe:FORMat? (see page 699)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 700)	:SAVE:IMAGe:INKSaver? (see page 700)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 701)	:SAVE:IMAGe:PALette? (see page 701)	<palette> ::= {COLor | GRAYscale}
:SAVE:LISTer[:STARt] [<file_name>] (see page 702)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:STARt] [<file_spec>] (see page 703)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:MULTi[:STARt] [<file_name>] (see page 704)	n/a	<file_name> ::= quoted ASCII string
:SAVE:POWer[:STARt] [<file_name>] (see page 705)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 706)	:SAVE:PWD? (see page 706)	<path_name> ::= quoted ASCII string
:SAVE:RESults:[STARt] [<file_spec>] (see page 707)	n/a	<file_name> ::= quoted ASCII string
:SAVE:RESults:FORMat: CURSor {{0 | OFF} | {1 | ON}} (see page 708)	:SAVE:RESults:FORMat: CURSor? (see page 708)	{0 | 1}
:SAVE:RESults:FORMat: MASK {{0 | OFF} | {1 | ON}} (see page 709)	:SAVE:RESults:FORMat: MASK? (see page 709)	{0 | 1}

Table 22 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:RESults:FORMat:MEASurement {{0 | OFF} | {1 | ON}} (see page 710)	:SAVE:RESults:FORMat:MEASurement? (see page 710)	{0 | 1}
:SAVE:RESults:FORMat:SEARch {{0 | OFF} | {1 | ON}} (see page 711)	:SAVE:RESults:FORMat:SEARch? (see page 711)	{0 | 1}
:SAVE:RESults:FORMat:SEGMented {{0 | OFF} | {1 | ON}} (see page 712)	:SAVE:RESults:FORMat:SEGMented? (see page 712)	{0 | 1}
:SAVE:SETup[:STARt] [<file_spec>] (see page 713)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:STARt] [<file_name>] (see page 714)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 715)	:SAVE:WAVeform:FORMat? (see page 715)	<format> ::= {ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 716)	:SAVE:WAVeform:LENGth? (see page 716)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 717)	:SAVE:WAVeform:LENGth:MAX? (see page 717)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 718)	:SAVE:WAVeform:SEGMented? (see page 718)	<option> ::= {ALL | CURRent}

Table 22 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WMEMory:SOURce <source> (see page 719)	:SAVE:WMEMory:SOURce? (see page 719)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:STARt] [<file_name>] (see page 720)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 23 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 724)	:SBUS<n>:DISPlay? (see page 724)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 725)	:SBUS<n>:MODE? (see page 725)	<mode> ::= {A429 | CAN | CXPI | IIC | LIN | M1553 | MANChester | NRZ | SENT | UART | USBPd}

Table 24 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUTosetup (see page 728)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 729)	:SBUS<n>:A429:BASE? (see page 729)	<base> ::= {BINary | HEX}
:SBUS<n>:A429:BAUDrate <baudrate> (see page 730)	:SBUS<n>:A429:BAUDrate? (see page 730)	<baudrate> ::= integer from 10000 to 1000000
n/a	:SBUS<n>:A429:COUNt:ERRor? (see page 731)	<error_count> ::= integer in NR1 format

Table 24 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:COUNt:RESet (see page 732)	n/a	n/a
n/a	:SBUS<n>:A429:COUNt:WORD? (see page 733)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMat <format> (see page 734)	:SBUS<n>:A429:FORMat? (see page 734)	<format> ::= {LDSDi | LDSSm | LDATa}
:SBUS<n>:A429:SIGNal <signal> (see page 735)	:SBUS<n>:A429:SIGNal? (see page 735)	<signal> ::= {A | B | DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 736)	:SBUS<n>:A429:SOURce? (see page 736)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEed <speed> (see page 737)	:SBUS<n>:A429:SPEed? (see page 737)	<speed> ::= {LOW | HIGH | USER}
:SBUS<n>:A429:TRIGger:LABel <value> (see page 738)	:SBUS<n>:A429:TRIGger:LABel? (see page 738)	<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 or "0xXX" (don't care) <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SBUS<n>:A429:TRIGger:PATTern:DATA <string> (see page 739)	:SBUS<n>:A429:TRIGger:PATTern:DATA? (see page 739)	<string> ::= "nn...n" where n ::= {0 | 1 | X}, length depends on FORMat
:SBUS<n>:A429:TRIGger:PATTern:SDI <string> (see page 740)	:SBUS<n>:A429:TRIGger:PATTern:SDI? (see page 740)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger:PATTern:SSM <string> (see page 741)	:SBUS<n>:A429:TRIGger:PATTern:SSM? (see page 741)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits

Table 24 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :RANGe <min>,<max> (see page 742)	:SBUS<n>:A429:TRIGger :RANGe? (see page 742)	<min> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <max> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SBUS<n>:A429:TRIGger :TYPE <condition> (see page 743)	:SBUS<n>:A429:TRIGger :TYPE? (see page 743)	<condition> ::= {WSTArt | WSTOp | LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors | LRANge | ABITs | AOBits | AZBits}

Table 25 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 748)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 749)	<frame_count> ::= 0 in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 750)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:SPEC? (see page 751)	<spec_error_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 752)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 753)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPlay <type> (see page 754)	:SBUS<n>:CAN:DISPlay? (see page 754)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:CAN:FDSPoint <value> (see page 755)	:SBUS<n>:CAN:FDSPoint? (see page 755)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.

Table 25 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:FDSTandard <std> (see page 756)	:SBUS<n>:CAN:FDSTandard? (see page 756)	<std> ::= {ISO | NISO}
:SBUS<n>:CAN:SAMPlepoint <percent> (see page 757)	:SBUS<n>:CAN:SAMPlepoint? (see page 757)	<percent> ::= 30.0 to 90.0 in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 758)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 758)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 759)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 759)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate> (see page 760)	:SBUS<n>:CAN:SIGNal:FDBaudrate? (see page 760)	<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
:SBUS<n>:CAN:SOURce <source> (see page 761)	:SBUS<n>:CAN:SOURce? (see page 761)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 762)	:SBUS<n>:CAN:TRIGger? (see page 763)	<condition> ::= {SOF | EOF | IDData | DATA | FDData | IDRemote | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive | OVERload | MESSage | MSIGnal | FDMSignal}
:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}} (see page 765)	:SBUS<n>:CAN:TRIGger:IDFilter? (see page 765)	{0 | 1}
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 766)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 766)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc> (see page 767)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? (see page 767)	<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 768)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 768)	<length> ::= integer from 1 to 8 in NR1 format

Table 25 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start> (see page 769)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? (see page 769)	<start> ::= integer between 0 and 63, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 770)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 770)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value> (see page 771)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 771)	<value> ::= {STANdard | EXTended}
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name> (see page 772)	:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? (see page 772)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name> (see page 773)	:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? (see page 773)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data> (see page 774)	:SBUS<n>:CAN:TRIGger:SYMBolic:VALue? (see page 774)	<data> ::= value in NR3 format

Table 26 :SBUS<n>:CXPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:CXPI:BAUDrate <baudrate> (see page 777)	:SBUS<n>:CXPI:BAUDrate? (see page 777)	<baudrate> ::= integer from 9600 to 40000 in 100 b/s increments.
:SBUS<n>:CXPI:PARity {{0 | OFF} | {1 | ON}} (see page 778)	:SBUS<n>:CXPI:PARity? (see page 778)	{0 | 1}
:SBUS<n>:CXPI:SOURce <source> (see page 779)	:SBUS<n>:CXPI:SOURce? (see page 779)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CXPI:TOLerance <percent> (see page 780)	:SBUS<n>:CXPI:TOLerance? (see page 780)	<percent> ::= from 1-30, in NR1 format.

Table 26 :SBUS<n>:CXPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CXPI:TRIGger <mode> (see page 781)	:SBUS<n>:CXPI:TRIGger? (see page 782)	<mode> ::= {SOF | EOF | PTYPe | ID | DATA | LDATa | CRCerror | PARityerror | IBSerror | IFSerror | FRAMingerror | DLENgtherror | SAMPleerror | ALLerrors | SLEepframe | WAKeuppulse}
:SBUS<n>:CXPI:TRIGger :IDFilter {{0 | OFF} | {1 | ON}} (see page 783)	:SBUS<n>:CXPI:TRIGger :IDFilter? (see page 783)	{0 | 1}
:SBUS<n>:CXPI:TRIGger :PTYPe {{0 | OFF} | {1 | ON}} (see page 784)	:SBUS<n>:CXPI:TRIGger :PTYPe? (see page 784)	{0 | 1}
:SBUS<n>:CXPI:TRIGger :PATTern:DATA <string> (see page 785)	:SBUS<n>:CXPI:TRIGger :PATTern:DATA? (see page 785)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:CXPI:TRIGger :PATTern:DATA:LENGth <length> (see page 786)	:SBUS<n>:CXPI:TRIGger :PATTern:DATA:LENGth? (see page 786)	<start> ::= integer between 0 and 12, in NR1 format.
:SBUS<n>:CXPI:TRIGger :PATTern:DATA:STARt <start> (see page 787)	:SBUS<n>:CXPI:TRIGger :PATTern:DATA:STARt? (see page 787)	<start> ::= integer between 0 and 124, in NR1 format.
:SBUS<n>:CXPI:TRIGger :PATTern:ID <string> (see page 788)	:SBUS<n>:CXPI:TRIGger :PATTern:ID? (see page 788)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:CXPI:TRIGger :PATTern:INFO:CT <string> (see page 789)	:SBUS<n>:CXPI:TRIGger :PATTern:INFO:CT? (see page 789)	<string> ::= "nn" where n ::= {0 | 1 | X}

Table 26 :SBUS<n>:CXPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC <dlc> (see page 790)	:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC? (see page 790)	<dlc> ::= integer between -1 (don't care) and 15, in NR1 format, when trigger is in DATA mode. <dlc> ::= integer between -1 (don't care) and 255, in NR1 format, when trigger is in LDATa mode.
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM <string> (see page 791)	:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM? (see page 791)	<string> ::= "nn" where n ::= {0 | 1 | X}

Table 27 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 793)	:SBUS<n>:IIC:ASIZe? (see page 793)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce]:CLOCk <source> (see page 794)	:SBUS<n>:IIC[:SOURce]:CLOCk? (see page 794)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC[:SOURce]:DATA <source> (see page 795)	:SBUS<n>:IIC[:SOURce]:DATA? (see page 795)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value> (see page 796)	:SBUS<n>:IIC:TRIGger:PATTern:ADDRess? (see page 796)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value> (see page 797)	:SBUS<n>:IIC:TRIGger:PATTern:DATA? (see page 797)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value> (see page 798)	:SBUS<n>:IIC:TRIGger:PATTern:DATa2? (see page 798)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

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

Command	Query	Options and Query Returns
:SBUS<n>:IIC:TRIGger:QUALifier <value> (see page 799)	:SBUS<n>:IIC:TRIGger:QUALifier? (see page 799)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 800)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 800)	<type> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk | NACKnowledge | READEprom | READ7 | WRITe7 | R7Data2 | W7Data2 | WRITe10}

Table 28 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:DISPlay <type> (see page 804)	:SBUS<n>:LIN:DISPlay? (see page 804)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 805)	:SBUS<n>:LIN:PARity? (see page 805)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 806)	:SBUS<n>:LIN:SAMPlepoint? (see page 806)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 807)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 807)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 808)	:SBUS<n>:LIN:SOURce? (see page 808)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 809)	:SBUS<n>:LIN:STANdard? (see page 809)	<std> ::= {LIN13 | LIN13NLC | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 810)	:SBUS<n>:LIN:SYNCbreak? (see page 810)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 811)	:SBUS<n>:LIN:TRIGger? (see page 811)	<condition> ::= {SYNCbreak | ID | DATA}

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

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:ID <value> (see page 813)	:SBUS<n>:LIN:TRIGger:ID? (see page 813)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 814)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 814)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 | 1 | X | $} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length> (see page 816)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 816)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 817)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 817)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name> (see page 818)	:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? (see page 818)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name> (see page 819)	:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? (see page 819)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data> (see page 820)	:SBUS<n>:LIN:TRIGger:SYMBolic:VALue? (see page 820)	<data> ::= value in NR3 format

Table 29 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTosetup (see page 822)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 823)	:SBUS<n>:M1553:BASE? (see page 823)	<base> ::= {BINary | HEX}
:SBUS<n>:M1553:SOURce <source> (see page 824)	:SBUS<n>:M1553:SOURce? (see page 824)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGger:PATTern:DATA <string> (see page 825)	:SBUS<n>:M1553:TRIGger:PATTern:DATA? (see page 825)	<string> ::= "nn...n" where n ::= {0 | 1 | X}
:SBUS<n>:M1553:TRIGger:RTA <value> (see page 826)	:SBUS<n>:M1553:TRIGger:RTA? (see page 826)	<value> ::= 5-bit integer in decimal, <nondecimal>, or <string> from 0-31 <nondecimal> ::= #Hnn where n ::= {0,..,9|A,..,F} <string> ::= "0xnn" where n::= {0,..,9|A,..,F}
:SBUS<n>:M1553:TRIGger:TYPE <type> (see page 827)	:SBUS<n>:M1553:TRIGger:TYPE? (see page 827)	<type> ::= {DSTArt | DSTOp | CSTArt | CSTOp | RTA | PERRor | SERRor | MERRor | RTA11}

Table 30 :SBUS<n>:MANChester Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:MANChester:BASE <base> (see page 830)	:SBUS<n>:MANChester:BASE? (see page 830)	<base> ::= {HEX | DECimal | ASCii}
:SBUS<n>:MANChester:BAUDrate <baudrate> (see page 831)	:SBUS<n>:MANChester:BAUDrate? (see page 831)	<baudrate> ::= integer from 500 to 5000000 in 100 b/s increments
:SBUS<n>:MANChester:BITorder <bitorder> (see page 832)	:SBUS<n>:MANChester:BITorder? (see page 832)	<bitorder> ::= {MSBFirst | LSBFirst}
:SBUS<n>:MANChester:DISPlay <format> (see page 833)	:SBUS<n>:MANChester:DISPlay? (see page 833)	<format> ::= {BIT | WORD}

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

Command	Query	Options and Query Returns
:SBUS<n>:MANChester:DSIZe {AUTO | <#words>} (see page 834)	:SBUS<n>:MANChester:DSIZe? (see page 834)	<#words> ::= from 1-255, in NR1 format
:SBUS<n>:MANChester:HSIZe <#bits> (see page 835)	:SBUS<n>:MANChester:HSIZe? (see page 835)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:MANChester:IDLE:BITS <#bits> (see page 836)	:SBUS<n>:MANChester:IDLE:BITS? (see page 836)	<#bits> ::= minimum idle time in terms of bit width, from 1.50 to 32.0, in NR3 format.
:SBUS<n>:MANChester:LOGic <logic> (see page 837)	:SBUS<n>:MANChester:LOGic? (see page 837)	<logic> ::= {FALLing | RISing}
:SBUS<n>:MANChester:SOURce <source> (see page 838)	:SBUS<n>:MANChester:SOURce? (see page 838)	<source> ::= {CHANnel<n>}
:SBUS<n>:MANChester:SSIZe <#bits> (see page 839)	:SBUS<n>:MANChester:SSIZe? (see page 839)	<#bits> ::= from 0-255, in NR1 format
:SBUS<n>:MANChester:STARt <edge#> (see page 840)	:SBUS<n>:MANChester:STARt? (see page 840)	<edge#> ::= from 1-256, in NR1 format
:SBUS<n>:MANChester:TOLerance <percent> (see page 841)	:SBUS<n>:MANChester:TOLerance? (see page 841)	<percent> ::= from 1-30, in NR1 format
:SBUS<n>:MANChester:TRIGger <mode> (see page 842)	:SBUS<n>:MANChester:TRIGger? (see page 842)	<mode> ::= {SOF | VALue | MERRor}
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA <string> (see page 843)	:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA? (see page 843)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh <width> (see page 844)	:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh? (see page 844)	<width> ::= integer from 4 to 128 in NR1 format

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

Command	Query	Options and Query Returns
:SBUS<n>:MANChester:TSIZe <#bits> (see page 845)	:SBUS<n>:MANChester:TSIZe? (see page 845)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:MANChester:WSIZe <#bits> (see page 846)	:SBUS<n>:MANChester:WSIZe? (see page 846)	<#bits> ::= from 2-32, in NR1 format

Table 31 :SBUS<n>:NRZ Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:NRZ:BASE <base> (see page 849)	:SBUS<n>:NRZ:BASE? (see page 849)	<base> ::= {HEX | DECimal | ASCii}
:SBUS<n>:NRZ:BAUDrate <baudrate> (see page 850)	:SBUS<n>:NRZ:BAUDrate? (see page 850)	<baudrate> ::= integer from 5000 to 5000000 in 100 b/s increments
:SBUS<n>:NRZ:BITorder <bitorder> (see page 851)	:SBUS<n>:NRZ:BITorder? (see page 851)	<bitorder> ::= {MSBFirst | LSBFirst}
:SBUS<n>:NRZ:DISPlay <format> (see page 852)	:SBUS<n>:NRZ:DISPlay? (see page 852)	<format> ::= {BIT | WORD}
:SBUS<n>:NRZ:DSIZe <#words> (see page 853)	:SBUS<n>:NRZ:DSIZe? (see page 853)	<#words> ::= from 1-255, in NR1 format
:SBUS<n>:NRZ:FSIZe <#bits> (see page 854)	:SBUS<n>:NRZ:FSIZe? (see page 854)	<#bits> ::= from 2-255, in NR1 format
:SBUS<n>:NRZ:HSIZe <#bits> (see page 855)	:SBUS<n>:NRZ:HSIZe? (see page 855)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:NRZ:IDLE:BITS <#bits> (see page 856)	:SBUS<n>:NRZ:IDLE:BITS? (see page 856)	<#bits> ::= minimum idle time in terms of bit width, from 1.50 to 32.0, in NR3 format.
:SBUS<n>:NRZ:IDLE:STATe <state> (see page 857)	:SBUS<n>:NRZ:IDLE:STATe? (see page 857)	<state> ::= {LOW | HIGH}
:SBUS<n>:NRZ:LOGic <logic> (see page 858)	:SBUS<n>:NRZ:LOGic? (see page 858)	<logic> ::= {HIGH | LOW}
:SBUS<n>:NRZ:SOURce <source> (see page 859)	:SBUS<n>:NRZ:SOURce? (see page 859)	<source> ::= {CHANnel<n>}

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

Command	Query	Options and Query Returns
:SBUS<n>:NRZ:STARt <#bits> (see page 860)	:SBUS<n>:NRZ:STARt? (see page 860)	<#bits> ::= from 0-255, in NR1 format
:SBUS<n>:NRZ:TRIGger <mode> (see page 861)	:SBUS<n>:NRZ:TRIGger? (see page 861)	<mode> ::= {SOF | VALue}
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA <string> (see page 862)	:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA? (see page 862)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh <width> (see page 863)	:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh? (see page 863)	<width> ::= integer from 4 to 128 in NR1 format
:SBUS<n>:NRZ:TSIZe <#bits> (see page 864)	:SBUS<n>:NRZ:TSIZe? (see page 864)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:NRZ:WSIZe <#bits> (see page 865)	:SBUS<n>:NRZ:WSIZe? (see page 865)	<#bits> ::= from 2-32, in NR1 format

Table 32 :SBUS<n>:SENT Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SENT:CLOCk <period> (see page 868)	:SBUS<n>:SENT:CLOCk? (see page 868)	<period> ::= the nominal clock period (tick), from 500 ns to 300 us, in NR3 format.
:SBUS<n>:SENT:CRC <format> (see page 869)	:SBUS<n>:SENT:CRC? (see page 869)	<format> ::= {LEGacy | RECommended}
:SBUS<n>:SENT:DISPlay <base> (see page 870)	:SBUS<n>:SENT:DISPlay? (see page 870)	<base> ::= {HEX | DECimal | SYMBolic}
:SBUS<n>:SENT:FORMat <decode> (see page 872)	:SBUS<n>:SENT:FORMat? (see page 872)	<decode> ::= {NIBBles | FSIGnal | FSSerial | FESerial | SSERial | ESERial}
:SBUS<n>:SENT:IDLE <state> (see page 874)	:SBUS<n>:SENT:IDLE? (see page 874)	<state> ::= {LOW | HIGH}
:SBUS<n>:SENT:LENGth <#_nibbles> (see page 875)	:SBUS<n>:SENT:LENGth? (see page 875)	<#_nibbles> ::= from 1-6, in NR1 format.
:SBUS<n>:SENT:PPULse {{0 | OFF} | {1 | ON} | SPC} (see page 876)	:SBUS<n>:SENT:PPULse? (see page 876)	{0 | 1 | SPC}

Table 32 :SBUS<n>:SENT Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SENT:SIGNal<s>:DISPlay {{0 | OFF} | {1 | ON}} (see page 878)	:SBUS<n>:SENT:SIGNal<s>:DISPlay? (see page 878)	<s> ::= 1-6, in NR1 format. {0 | 1}
:SBUS<n>:SENT:SIGNal<s>:LENGth <length> (see page 879)	:SBUS<n>:SENT:SIGNal<s>:LENGth? (see page 879)	<s> ::= 1-6, in NR1 format. <length> ::= from 1-24, in NR1 format.
:SBUS<n>:SENT:SIGNal<s>:MULTiplier <multiplier> (see page 881)	:SBUS<n>:SENT:SIGNal<s>:MULTiplier? (see page 881)	<s> ::= 1-6, in NR1 format. <multiplier> ::= from 1-24, in NR3 format.
:SBUS<n>:SENT:SIGNal<s>:OFFSet <offset> (see page 883)	:SBUS<n>:SENT:SIGNal<s>:OFFSet? (see page 883)	<s> ::= 1-6, in NR1 format. <offset> ::= from 1-24, in NR3 format.
:SBUS<n>:SENT:SIGNal<s>:ORDer <order> (see page 885)	:SBUS<n>:SENT:SIGNal<s>:ORDer? (see page 885)	<s> ::= 1-6, in NR1 format. <order> ::= {MSNFirst | LSNFirst}
:SBUS<n>:SENT:SIGNal<s>:STARt <position> (see page 887)	:SBUS<n>:SENT:SIGNal<s>:STARt? (see page 887)	<s> ::= 1-6, in NR1 format. <position> ::= from 0-23, in NR1 format.
:SBUS<n>:SENT:SOURce <source> (see page 889)	:SBUS<n>:SENT:SOURce? (see page 889)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SENT:TOLerance <percent> (see page 890)	:SBUS<n>:SENT:TOLerance? (see page 890)	<percent> ::= from 3-30, in NR1 format.
:SBUS<n>:SENT:TRIGger <mode> (see page 891)	:SBUS<n>:SENT:TRIGger? (see page 891)	<mode> ::= {SFCMessage | SSCMessage | FCData | SCMid | SCData | FCCerror | SCCerror | CRCerror | TOLerror | PPERror | SSPerror}
:SBUS<n>:SENT:TRIGger:FAST:DATA <string> (see page 893)	:SBUS<n>:SENT:TRIGger:FAST:DATA? (see page 893)	<string> ::= "nnnn..." where n ::= {0 | 1 | X} <string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}

Table 32 :SBUS<n>:SENT Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SENT:TRIGger :SLOW:DATA <data> (see page 894)	:SBUS<n>:SENT:TRIGger :SLOW:DATA? (see page 894)	<data> ::= when ILENgth = SHORt, from -1 (don't care) to 65535, in NR1 format. <data> ::= when ILENgth = LONG, from -1 (don't care) to 4095, in NR1 format.
:SBUS<n>:SENT:TRIGger :SLOW:ID <id> (see page 896)	:SBUS<n>:SENT:TRIGger :SLOW:ID? (see page 896)	<id> ::= when ILENgth = SHORt, from -1 (don't care) to 15, in NR1 format. <id> ::= when ILENgth = LONG, from -1 (don't care) to 255, in NR1 format.
:SBUS<n>:SENT:TRIGger :SLOW:ILENgth <length> (see page 898)	:SBUS<n>:SENT:TRIGger :SLOW:ILENgth? (see page 898)	<length> ::= {SHORt | LONG}
:SBUS<n>:SENT:TRIGger :TOLerance <percent> (see page 899)	:SBUS<n>:SENT:TRIGger :TOLerance? (see page 899)	<percent> ::= from 1-18, in NR1 format.

Table 33 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 902)	:SBUS<n>:UART:BASE? (see page 902)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 903)	:SBUS<n>:UART:BAUDrate? (see page 903)	<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
:SBUS<n>:UART:BITorder <bitorder> (see page 904)	:SBUS<n>:UART:BITorder? (see page 904)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 905)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 906)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 907)	<frame_count> ::= integer in NR1 format

Table 33 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 908)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 909)	:SBUS<n>:UART:FRAMing? (see page 909)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SBUS<n>:UART:PARity <parity> (see page 910)	:SBUS<n>:UART:PARity? (see page 910)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 911)	:SBUS<n>:UART:POLarity? (see page 911)	<polarity> ::= {HIGH | LOW}
:SBUS<n>:UART:SOURce:RX <source> (see page 912)	:SBUS<n>:UART:SOURce:RX? (see page 912)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:SOURce:TX <source> (see page 913)	:SBUS<n>:UART:SOURce:TX? (see page 913)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 914)	:SBUS<n>:UART:TRIGger:BASE? (see page 914)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 915)	:SBUS<n>:UART:TRIGger:BURSt? (see page 915)	<value> ::= {OFF | 1 to 4096 in NR1 format}

Table 33 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :DATA <value> (see page 916)	:SBUS<n>:UART:TRIGger :DATA? (see page 916)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SBUS<n>:UART:TRIGger :IDLE <time_value> (see page 917)	:SBUS<n>:UART:TRIGger :IDLE? (see page 917)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 918)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 918)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 919)	:SBUS<n>:UART:TRIGger :TYPE? (see page 919)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 920)	:SBUS<n>:UART:WIDTh? (see page 920)	<width> ::= {5 | 6 | 7 | 8 | 9}

Table 34 :SBUS<n>:USBPd Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:USBPd:SOURce <source> (see page 922)	:SBUS<n>:USBPd:SOURce? (see page 922)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:USBPd:TRIGger <mode> (see page 923)	:SBUS<n>:USBPd:TRIGger? (see page 923)	<mode> ::= {PSTart | EOP | SOP | SPRime | SDPRime | SPDebug | SDPDebug | HRST | CRST | CRCerror | PERRor | HEADer}
:SBUS<n>:USBPd:TRIGger:HEADer <type> (see page 924)	:SBUS<n>:USBPd:TRIGger:HEADer? (see page 924)	<type> ::= {CMESsage | DMESsage | EMESsage | VALue}

Table 34 :SBUS<n>:USBPd Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage <type> (see page 926)	:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage? (see page 926)	<type> ::= {GOODcrc | GOTOmin | ACCept | REJect | PING | PSRDy | GSRCap | GSNCap | DRSWap | PRSWap | VCSWap | WAIT | SRST | GSCX | GSTatus | FRSWap | GPSTatus | GCCodes}
:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage <type> (see page 928)	:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage? (see page 928)	<type> ::= {SRCap | REQuest | BIST | SNCap | BSTatus | ALERt | GCINfo | VDEFined}
:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage <type> (see page 929)	:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage? (see page 929)	<type> ::= {SCX | STATus | GBCap | GBSTatus | BCAP | GMINfo | MINFo | SREQuest | SRESponse | FREQuest | FRESponse | PSTatus | CINFo | CCODes}
:SBUS<n>:USBPd:TRIGger:HEADer:VALue <string> (see page 931)	:SBUS<n>:USBPd:TRIGger:HEADer:VALue? (see page 931)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier <type> (see page 932)	:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier? (see page 932)	<type> ::= {NONE | SOP | SPRime | SDPRime}

Table 35 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 935)	<count> ::= an integer count value
:SEARch:EVENt <event_number> (see page 936)	:SEARch:EVENt? (see page 936)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 937)	:SEARch:MODE? (see page 937)	<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2} | PEAK}
:SEARch:STATe <value> (see page 938)	:SEARch:STATe? (see page 938)	<value> ::= {{0 | OFF} | {1 | ON}}

Table 36 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 940)	:SEARch:EDGE:SLOPe? (see page 940)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 941)	:SEARch:EDGE:SOURce? (see page 941)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 37 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 943)	:SEARch:GLITch:GREaterthan? (see page 943)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 944)	:SEARch:GLITch:LESSthan? (see page 944)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 945)	:SEARch:GLITch:POLarity? (see page 945)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 946)	:SEARch:GLITch:QUALifier? (see page 946)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 947)	:SEARch:GLITch:RANGe? (see page 947)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:SOURce <source> (see page 948)	:SEARch:GLITch:SOURce? (see page 948)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 38 :SEARch:PEAK Commands Summary

Command	Query	Options and Query Returns
:SEARch:PEAK:EXCursion <delta_level> (see page 950)	:SEARch:PEAK:EXCursion? (see page 950)	<delta_level> ::= required change in level to be recognized as a peak, in NR3 format.
:SEARch:PEAK:NPEaks <number> (see page 951)	:SEARch:PEAK:NPEaks? (see page 951)	<number> ::= max number of peaks to find, 1-11 in NR1 format.
:SEARch:PEAK:SOURce <source> (see page 952)	:SEARch:PEAK:SOURce? (see page 952)	<source> ::= {FUNCtion<m> | MATH<m>} (must be an FFT waveform) <m> ::= 1 to 4 in NR1 format
:SEARch:PEAK:THReshold <level> (see page 953)	:SEARch:PEAK:THReshold? (see page 953)	<level> ::= necessary level to be considered a peak, in NR3 format.

Table 39 :SEARch:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARch:RUNT:POLarity <polarity> (see page 955)	:SEARch:RUNT:POLarity? (see page 955)	<polarity> ::= {POSitive | NEGative | EITHer}
:SEARch:RUNT:QUALifier <qualifier> (see page 956)	:SEARch:RUNT:QUALifier? (see page 956)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:SEARch:RUNT:SOURce <source> (see page 957)	:SEARch:RUNT:SOURce? (see page 957)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:RUNT:TIME <time>[suffix] (see page 958)	:SEARch:RUNT:TIME? (see page 958)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 40 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 960)	:SEARch:TRANsition:QUALifier? (see page 960)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 961)	:SEARch:TRANsition:SLOPe? (see page 961)	<slope> ::= {NEGative | POSitive}
:SEARch:TRANsition:SOURce <source> (see page 962)	:SEARch:TRANsition:SOURce? (see page 962)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 963)	:SEARch:TRANsition:TIME? (see page 963)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 41 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:LABel <value> (see page 965)	:SEARch:SERial:A429:LABel? (see page 965)	<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SEARch:SERial:A429:MODE <condition> (see page 966)	:SEARch:SERial:A429:MODE? (see page 966)	<condition> ::= {LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors}
:SEARch:SERial:A429:PATTern:DATA <string> (see page 967)	:SEARch:SERial:A429:PATTern:DATA? (see page 967)	<string> ::= "nn...n" where n ::= {0 | 1}, length depends on FORMat
:SEARch:SERial:A429:PATTern:SDI <string> (see page 968)	:SEARch:SERial:A429:PATTern:SDI? (see page 968)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
:SEARch:SERial:A429:PATTern:SSM <string> (see page 969)	:SEARch:SERial:A429:PATTern:SSM? (see page 969)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits

Table 42 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 971)	:SEARch:SERial:CAN:MODE? (see page 971)	<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload | MESSage | MSIGnal}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 973)	:SEARch:SERial:CAN:PATTern:DATA? (see page 973)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 974)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 974)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 975)	:SEARch:SERial:CAN:PATTern:ID? (see page 975)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 976)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 976)	<value> ::= {STANdard | EXTended}
:SEARch:SERial:CAN:SYMBolic:MESSage <name> (see page 977)	:SEARch:SERial:CAN:SYMBolic:MESSage? (see page 977)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:SIGNal <name> (see page 978)	:SEARch:SERial:CAN:SYMBolic:SIGNal? (see page 978)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:VALue <data> (see page 979)	:SEARch:SERial:CAN:SYMBolic:VALue? (see page 979)	<data> ::= value in NR3 format

Table 43 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 981)	:SEARch:SERial:IIC:MODE? (see page 981)	<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | READEprom | READ7 | WRITE7 | R7Data2 | W7Data2}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 983)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 983)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

Table 43 :SEARch:SERial:IIC Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 984)	:SEARch:SERial:IIC:PATTern:DATA? (see page 984)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 985)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 985)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:QUALifier <value> (see page 986)	:SEARch:SERial:IIC:QUALifier? (see page 986)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

Table 44 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 989)	:SEARch:SERial:LIN:ID? (see page 989)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with AUTO license) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SEARch:SERial:LIN:MODE <value> (see page 990)	:SEARch:SERial:LIN:MODE? (see page 990)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 991)	:SEARch:SERial:LIN:PATTern:DATA? (see page 991)	When :SEARch:SERial:LIN:PATTern:FORMat DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMat HEX, <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length> (see page 992)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 992)	<length> ::= integer from 1 to 8 in NR1 format

Table 44 :SEARch:SERial:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 993)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 993)	<base> ::= {HEX | DECimal}
:SEARch:SERial:LIN:SYMBolic:FRAMe <name> (see page 994)	:SEARch:SERial:LIN:SYMBolic:FRAMe? (see page 994)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:SIGNal <name> (see page 995)	:SEARch:SERial:LIN:SYMBolic:SIGNal? (see page 995)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:VALue <data> (see page 996)	:SEARch:SERial:LIN:SYMBolic:VALue? (s?e page 996)	<data> ::= value in NR3 format

Table 45 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553:MODE <value> (see page 998)	:SEARch:SERial:M1553:MODE? (see page 998)	<value> ::= {DSTArt | CSTArt | RTA | RTA11 | PERRor | SERRor | MERRor}
:SEARch:SERial:M1553:PATTern:DATA <string> (see page 999)	:SEARch:SERial:M1553:PATTern:DATA? (see page 999)	<string> ::= "nn...n" where n ::= {0 | 1}
:SEARch:SERial:M1553:RTA <value> (see page 1000)	:SEARch:SERial:M1553:RTA? (see page 1000)	<value> ::= 5-bit integer in decimal, <hexadecimal>, <binary>, or <string> from 0-31 < hexadecimal > ::= #Hnn where n ::= {0,..,9|A,..,F} <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n::= {0,..,9|A,..,F}

Table 46 :SEARch:SERial:SENT Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SENT:FAST:DATA <string> (see page 1002)	:SEARch:SERial:SENT:FAST:DATA? (see page 1002)	<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
:SEARch:SERial:SENT:MODE <mode> (see page 1003)	:SEARch:SERial:SENT:MODE? (see page 1003)	<mode> ::= {FCData | SCMid | SCData | CRCerror PPERror}
:SEARch:SERial:SENT:SLOW:DATA <data> (see page 1004)	:SEARch:SERial:SENT:SLOW:DATA? (see page 1004)	<data> ::= from -1 (don't care) to 65535, in NR1 format.
:SEARch:SERial:SENT:SLOW:ID <id> (see page 1005)	:SEARch:SERial:SENT:SLOW:ID? (see page 1005)	<id> ::= from -1 (don't care) to 255, in NR1 format.

Table 47 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 1007)	:SEARch:SERial:UART:DATA? (see page 1007)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9| A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SEARch:SERial:UART:MODE <value> (see page 1008)	:SEARch:SERial:UART:MODE? (see page 1008)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | PARityerror | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 1009)	:SEARch:SERial:UART:QUALifier? (see page 1009)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

Table 48 :SYSTem Commands Summary

Command	Query	Options and Query Returns
n/a	:SYSTem:DATE? (see page 1014)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,..,12} <day> ::= {1,..31}
:SYSTem:DSP <string> (see page 1015)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 1016)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1293).
:SYSTem:GUI:SHOW {{0 | OFF} | {1 | ON}} (see page 1017)	:SYSTem:GUI:SHOW? (see page 1017)	<setting> ::= {0 | 1}
:SYSTem:LOCK <value> (see page 1018)	:SYSTem:LOCK? (see page 1018)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:PERSona[:MANufacturer] <manufacturer_string> (see page 1019)	:SYSTem:PERSona[:MANufacturer]? (see page 1019)	<manufacturer_string> ::= quoted ASCII string, up to 63 characters
:SYSTem:PERSona[:MANufacturer]:DEFault (see page 1020)	n/a	Sets manufacturer string to "KEYSIGHT TECHNOLOGIES"
:SYSTem:PRECision {{1 | ON} | {0 | OFF}} (see page 1021)	:SYSTem:PRECision? (see page 1021)	{1 | 0}
:SYSTem:PRECision:LENGth <length> (see page 1022)	:SYSTem:PRECision:LENGth? (see page 1022)	<length> ::= between 100k and 1M points in NR1 format
:SYSTem:PRESet (see page 1023)	n/a	See :SYSTem:PRESet (see page 1023)
:SYSTem:PROTection:LOCK <value> (see page 1026)	:SYSTem:PROTection:LOCK? (see page 1026)	<value> ::= {{1 | ON} | {0 | OFF}}

Table 48 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]] (see page 1027)	n/a	<setting> ::= {{0 | OFF} | {1 | ON}} <file_name> ::= quoted ASCII string <write_mode> ::= {CREate | APPend}
:SYSTem:RLOGger:DESTination <dest> (see page 1028)	:SYSTem:RLOGger:DESTination? (see page 1028)	<dest> ::= {FILE | SCReen | BOTH}
:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}} (see page 1029)	:SYSTem:RLOGger:DISPlay? (see page 1029)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:FNAMe <file_name> (see page 1030)	:SYSTem:RLOGger:FNAMe? (see page 1030)	<file_name> ::= quoted ASCII string
:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}} (see page 1031)	:SYSTem:RLOGger:STATe? (see page 1031)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:TRANsparent {{0 | OFF} | {1 | ON}} (see page 1032)	:SYSTem:RLOGger:TRANsparent? (see page 1032)	<setting> ::= 0
:SYSTem:RLOGger:WMODe <write_mode> (see page 1033)	:SYSTem:RLOGger:WMODe? (see page 1033)	<write_mode> ::= {CREate | APPend}
:SYSTem:SETup <setup_data> (see page 1034)	:SYSTem:SETup? (see page 1034)	<setup_data> ::= data in IEEE 488.2 # format.
n/a	:SYSTem:TIME? (see page 1036)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TOUCh {{1 | ON} | {0 | OFF}} (see page 1037)	:SYSTem:TOUCh? (see page 1037)	{1 | 0}

Table 49 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 1041)	:TIMebase:MODE? (see page 1041)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 1042)	:TIMebase:POSition? (see page 1042)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 1043)	:TIMebase:RANGe? (see page 1043)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase:REFClock {{0 | OFF} | {1 | ON}} (see page 1044)	:TIMebase:REFClock? (see page 1044)	{0 | 1}
:TIMebase:REFerence {LEFT | CENTer | RIGHt | CUSTom} (see page 1045)	:TIMebase:REFerence? (see page 1045)	<return_value> ::= {LEFT | CENTer | RIGHt | CUSTom}
:TIMebase:REFerence:LOCation <loc> (see page 1046)	:TIMebase:REFerence:LOCation? (see page 1046)	<loc> ::= 0.0 to 1.0 in NR3 format
:TIMebase:SCALe <scale_value> (see page 1047)	:TIMebase:SCALe? (see page 1047)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 1048)	:TIMebase:VERNier? (see page 1048)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 1049)	:TIMebase:WINDow:POSition? (see page 1049)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 1050)	:TIMebase:WINDow:RANGe? (see page 1050)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 1051)	:TIMebase:WINDow:SCALe? (see page 1051)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Table 50 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 1057)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 1058)	:TRIGger:HFReject? (see page 1058)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 1059)	:TRIGger:HOLDoff? (see page 1059)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:HOLDoff:MAXimum <max_holdoff> (see page 1060)	:TRIGger:HOLDoff:MAXimum? (see page 1060)	<max_holdoff> ::= maximum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:MINimum <min_holdoff> (see page 1061)	:TRIGger:HOLDoff:MINimum? (see page 1061)	<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}} (see page 1062)	:TRIGger:HOLDoff:RANDom? (see page 1062)	<setting> ::= {0 | 1}
:TRIGger:LEVel:ASETup (see page 1063)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 1064)	:TRIGger:LEVel:HIGH? <source> (see page 1064)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 1065)	:TRIGger:LEVel:LOW? <source> (see page 1065)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see page 1066)	:TRIGger:MODE? (see page 1066)	<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2} | NFC}

Table 50 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 1067)	:TRIGger:NREJect? (see page 1067)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 1068)	:TRIGger:SWEep? (see page 1068)	<sweep> ::= {AUTO | NORMal}

Table 51 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SLOPe <slope> (see page 1070)	:TRIGger:DELay:ARM:SLOPe? (see page 1070)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:ARM:SOURce <source> (see page 1071)	:TRIGger:DELay:ARM:SOURce? (see page 1071)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:DELay:TDELay:TIME <time_value> (see page 1072)	:TRIGger:DELay:TDELay:TIME? (see page 1072)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGger:COUNt <count> (see page 1073)	:TRIGger:DELay:TRIGger:COUNt? (see page 1073)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGger:SLOPe <slope> (see page 1074)	:TRIGger:DELay:TRIGger:SLOPe? (see page 1074)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:TRIGger:SOURce <source> (see page 1075)	:TRIGger:DELay:TRIGger:SOURce? (see page 1075)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 52 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNt <count> (see page 1077)	:TRIGger:EBURst:COUNt? (see page 1077)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 1078)	:TRIGger:EBURst:IDLE? (see page 1078)	<time_value> ::= time in seconds in NR3 format

Table 52 :TRIGger:EBURst Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:EBURst:SLOPe <slope> (see page 1079)	:TRIGger:EBURst:SLOPe? (see page 1079)	<slope> ::= {NEGative | POSitive}
:TRIGger:EBURst:SOURce <source> (see page 1080)	:TRIGger:EBURst:SOURce? (see page 1080)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 53 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 1082)	:TRIGger[:EDGE]:COUPling? (see page 1082)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 1083)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 1083)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 1084)	:TRIGger[:EDGE]:REJect? (see page 1084)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 1085)	:TRIGger[:EDGE]:SLOPe? (see page 1085)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 1086)	:TRIGger[:EDGE]:SOURce? (see page 1086)	<source> ::= {CHANnel<n> | EXTernal | WGEN | WGEN1 | WMOD} <n> ::= 1 to (# analog channels) in NR1 format Note: WGEN and WGEN1 are equivalent.

Table 54 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 1088)	:TRIGger:GLITch:GREaterthan? (see page 1088)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 1089)	:TRIGger:GLITch:LESSthan? (see page 1089)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 1090)	:TRIGger:GLITch:LEVel? (see page 1090)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 1091)	:TRIGger:GLITch:POLarity? (see page 1091)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 1092)	:TRIGger:GLITch:QUALifier? (see page 1092)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1093)	:TRIGger:GLITch:RANGe? (see page 1093)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 1094)	:TRIGger:GLITch:SOURce? (see page 1094)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 55 :TRIGger:NFC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:NFC:AEVent <arm_event> (see page 1096)	:TRIGger:NFC:AEVent? (see page 1096)	<arm_event> ::= {NONE | ASReq | AALLreq | AEITher | BSReq | BALLreq | BEITher | FSReq}
n/a	:TRIGger:NFC:ATTime? (see page 1097)	<time> ::= seconds in NR3 format
:TRIGger:NFC:RPOLarity {{0 | OFF} | {1 | ON}} (see page 1098)	:TRIGger:NFC:RPOLarity? (see page 1098)	{0 | 1}
:TRIGger:NFC:SOURce <source> (see page 1099)	:TRIGger:NFC:SOURce? (see page 1099)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:NFC:STANdard <standard> (see page 1100)	:TRIGger:NFC:STANdard? (see page 1100)	<standard> ::= {{A | A106} | {B | B106} | F212 | F424}
:TRIGger:NFC:TEVent <trigger_event> (see page 1101)	:TRIGger:NFC:TEVent? (see page 1101)	<trigger_event> ::= {ATRigger | ASReq | AALLreq | AEITher | ASDDreq | BSReq | BALLreq | BEITher | BATTrib | FSReq | FAReq | FPReamble}
n/a	:TRIGger:NFC:TIMeout? (see page 1103)	{0 | 1}
:TRIGger:NFC:TIMeout:ENABle {{0 | OFF} | {1 | ON}} (see page 1104)	:TRIGger:NFC:TIMeout:ENABle? (see page 1104)	{0 | 1}
:TRIGger:NFC:TIMeout:TIME <time> (see page 1105)	:TRIGger:NFC:TIMeout:TIME? (see page 1105)	<time> ::= seconds in NR3 format

Table 56 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1107)	:TRIGger:OR? (see page 1107)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog channel as shown on the soft front panel display.

Table 57 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 1109)	:TRIGger:PATTern? (see page 1109)	<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when <base> = ASCii <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX <edge_source> ::= {CHANnel<n> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 1111)	:TRIGger:PATTern:FORMat? (see page 1111)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 1112)	:TRIGger:PATTern:GREaterthan? (see page 1112)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 1113)	:TRIGger:PATTern:LESSthan? (see page 1113)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:QUALifier <qualifier> (see page 1114)	:TRIGger:PATTern:QUALifier? (see page 1114)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1115)	:TRIGger:PATTern:RANGe? (see page 1115)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 58 :TRIGger:PXI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PXI:MALine<n>:ENABle {{0 | OFF} | {1 | ON}} (see page 1118)	:TRIGger:PXI:MALine<n>:ENABle? (see page 1118)	<setting> ::= {0 | 1} <n> ::= 0 to (# chassis lines - 1) in NR1 format
:TRIGger:PXI:MODE <mode> (see page 1119)	:TRIGger:PXI:MODE? (see page 1119)	<mode> ::= {OFF | MASTer | SLAVe}
:TRIGger:PXI:MSLot <slot_number> (see page 1120)	:TRIGger:PXI:MSLot? (see page 1120)	<slot_number> ::= 2 to (# chassis slots) in NR1 format
:TRIGger:PXI:SALine <trigger_line> (see page 1121)	:TRIGger:PXI:SALine? (see page 1121)	<arm_line> ::= {PTRig<n>} <n> ::= 0 to (# chassis lines - 1) in NR1 format
:TRIGger:PXI:SYNC {{0 | OFF} | {1 | ON}} (see page 1122)	:TRIGger:PXI:SYNC? (see page 1122)	<setting> ::= {0 | 1}
:TRIGger:PXI:TLINe <trigger_line> (see page 1123)	:TRIGger:PXI:TLINe? (see page 1123)	<trigger_line> ::= {PTRig<n>} <n> ::= 0 to (# chassis lines - 1) in NR1 format

Table 59 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 1125)	:TRIGger:RUNT:POLarity? (see page 1125)	<polarity> ::= {POSitive | NEGative | EITHer}
:TRIGger:RUNT:QUALifier <qualifier> (see page 1126)	:TRIGger:RUNT:QUALifier? (see page 1126)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 1127)	:TRIGger:RUNT:SOURce? (see page 1127)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1128)	:TRIGger:RUNT:TIME? (see page 1128)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 60 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 1130)	:TRIGger:SHOLd:SLOPe? (see page 1130)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce:CLOCk <source> (see page 1131)	:TRIGger:SHOLd:SOURce:CLOCk? (see page 1131)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:SHOLd:SOURce:DATA <source> (see page 1132)	:TRIGger:SHOLd:SOURce:DATA? (see page 1132)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 1133)	:TRIGger:SHOLd:TIME:HOLD? (see page 1133)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 1134)	:TRIGger:SHOLd:TIME:SETup? (see page 1134)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 61 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 1136)	:TRIGger:TRANsition:QUALifier? (see page 1136)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 1137)	:TRIGger:TRANsition:SLOPe? (see page 1137)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 1138)	:TRIGger:TRANsition:SOURce? (see page 1138)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 1139)	:TRIGger:TRANsition:TIME? (see page 1139)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 62 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 1141)	:TRIGger:TV:LINE? (see page 1141)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 1142)	:TRIGger:TV:MODE? (see page 1142)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | LFIeld1 | LFIeld2 | LALTernate}
:TRIGger:TV:POLarity <polarity> (see page 1143)	:TRIGger:TV:POLarity? (see page 1143)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 1144)	:TRIGger:TV:SOURce? (see page 1144)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see page 1145)	:TRIGger:TV:STANdard? (see page 1145)	<standard> ::= {NTSC | PAL | PALM | SECam} <standard> ::= {GENeric | {P480L60HZ | P480} | {P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | P1080L50HZ | P1080L60HZ | {I1080L50HZ | I1080} | I1080L60HZ} with extended video triggering license
:TRIGger:TV:UDTV:ENUMber <count> (see page 1146)	:TRIGger:TV:UDTV:ENUMber? (see page 1146)	<count> ::= edge number in NR1 format
:TRIGger:TV:UDTV:HSYNc {{0 | OFF} | {1 | ON}} (see page 1147)	:TRIGger:TV:UDTV:HSYNc? (see page 1147)	{0 | 1}
:TRIGger:TV:UDTV:HTIMe <time> (see page 1148)	:TRIGger:TV:UDTV:HTIMe? (see page 1148)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTHan <min_time> (see page 1149)	:TRIGger:TV:UDTV:PGTHan? (see page 1149)	<min_time> ::= seconds in NR3 format

Table 63 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE:SOURce <source> (see page 1151)	:TRIGger:ZONE:SOURce? (see page 1151)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:ZONE:STATe {{0 | OFF} | {1 | ON}} (see page 1152)	:TRIGger:ZONE:STATe? (see page 1152)	{0 | 1}
:TRIGger:ZONE<n>:MODE <mode> (see page 1153)	:TRIGger:ZONE<n>:MODE? (see page 1153)	<mode> ::= {INTersect | NOTintersect} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:PLACement <width>, <height>, <x_center>, <y_center> (see page 1154)	:TRIGger:ZONE<n>:PLACement? (see page 1154)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <n> ::= 1-2 in NR1 format
n/a	:TRIGger:ZONE<n>:VALidity? (see page 1155)	<value> ::= {VALid | INValid | OSCReen} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:STATe {{0 | OFF} | {1 | ON}} (see page 1156)	:TRIGger:ZONE<n>:STATe? (see page 1156)	{0 | 1} <n> ::= 1-2 in NR1 format

Table 64 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 1165)	:WAVeform:BYTeorder? (see page 1165)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 1166)	<count> ::= an integer from 1 to 65536 in NR1 format

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:DATA? (see page 1167)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMat <value> (see page 1169)	:WAVeform:FORMat? (see page 1169)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <# points> (see page 1170)	:WAVeform:POINts? (see page 1170)	<# points> ::= {100 | 250 | 500 | 1000 | <points_mode>} if waveform points mode is NORMal <# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMal | MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode> (see page 1172)	:WAVeform:POINts:MODE? (see page 1173)	<points_mode> ::= {NORMal | MAXimum | RAW}

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:PREamble? (see page 1174)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 3 for AVERage type • 4 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
n/a	:WAVeform:SEGMented:COUNt? (see page 1177)	<count> ::= an integer from 2 to 1000 in NR1 format (with SGM license)
n/a	:WAVeform:SEGMented:TTAG? (see page 1178)	<time_tag> ::= in NR3 format (with SGM license)
:WAVeform:SOURce <source> (see page 1179)	:WAVeform:SOURce? (see page 1179)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r> | SBUS{1 | 2}} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 1183)	:WAVeform:SOURce:SUBSource? (see page 1183)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
n/a	:WAVeform:TYPE? (see page 1184)	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 1185)	:WAVeform:UNSigned? (see page 1185)	{0 | 1}

Table 64 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:VIEW <view> (see page 1186)	:WAVeform:VIEW? (see page 1186)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 1187)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 1188)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 1189)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 1190)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 1191)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 1192)	<return_value> ::= y-reference value in the current preamble in NR1 format

Table 65 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BYTeorder <order> (see page 1198)	:WGEN<w>:ARBitrary:BYTeorder? (see page 1198)	<order> ::= {MSBFirst | LSBFirst} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 1199)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 to (# WaveGen outputs) in NR1 format
n/a	:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts? (see page 1202)	<points> ::= number of points in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

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

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:DATA:CLEar (see page 1203)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 1204)	n/a	<binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format <value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 1205)	:WGEN<w>:ARBitrary:INTerpolate? (see page 1205)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:STORe <source> (see page 1206)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | FFT | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FREQuency <frequency> (see page 1207)	:WGEN<w>:FREQuency? (see page 1207)	<frequency> ::= frequency in Hz in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion <signal> (see page 1208)	:WGEN<w>:FUNCtion? (see page 1211)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:PULSe:WIDTh <width> (see page 1212)	:WGEN<w>:FUNCtion:PULSe:WIDTh? (see page 1212)	<width> ::= pulse width in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

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

Command	Query	Options and Query Returns
:WGEN<w>:FUNCtion:RAMP:SYMMetry <percent> (see page 1213)	:WGEN<w>:FUNCtion:RAMP:SYMMetry? (see page 1213)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent> (see page 1214)	:WGEN<w>:FUNCtion:SQUare:DCYCle? (see page 1214)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:MODulation:AM:DEPTh <percent> (see page 1215)	:WGEN<w>:MODulation:AM:DEPTh? (see page 1215)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:AM:FREQuency <frequency> (see page 1216)	:WGEN<w>:MODulation:AM:FREQuency? (see page 1216)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:DEViation <frequency> (see page 1217)	:WGEN<w>:MODulation:FM:DEViation? (see page 1217)	<frequency> ::= frequency deviation in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:FREQuency <frequency> (see page 1218)	:WGEN<w>:MODulation:FM:FREQuency? (see page 1218)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FSKey:FREQuency <percent> (see page 1219)	:WGEN<w>:MODulation:FSKey:FREQuency? (see page 1219)	<frequency> ::= hop frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FSKey:RATE <rate> (see page 1220)	:WGEN<w>:MODulation:FSKey:RATE? (see page 1220)	<rate> ::= FSK modulation rate in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion <shape> (see page 1221)	:WGEN<w>:MODulation:FUNCtion? (see page 1221)	<shape> ::= {SINusoid | SQUare| RAMP} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry <percent> (see page 1222)	:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry? (see page 1222)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format

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

Command	Query	Options and Query Returns
:WGEN<w>:MODulation:NOISe <percent> (see page 1223)	:WGEN<w>:MODulation:NOISe? (see page 1223)	<percent> ::= 0 to 100 <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:STATe {{0 | OFF} | {1 | ON}} (see page 1224)	:WGEN<w>:MODulation:STATe? (see page 1224)	{0 | 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:TYPE <type> (see page 1225)	:WGEN<w>:MODulation:TYPE? (see page 1225)	<type> ::= {AM | FM | FSK} <w> ::= 1 in NR1 format
:WGEN<w>:OUTPut {{0 | OFF} | {1 | ON}} (see page 1227)	:WGEN<w>:OUTPut? (see page 1227)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1228)	:WGEN<w>:OUTPut:LOAD? (see page 1228)	<impedance> ::= {ONEMeg | FIFTy} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:MODE <mode> (see page 1229)	:WGEN<w>:OUTPut:MODE? (see page 1229)	<mode> ::= {NORMal | SINGle}
:WGEN<w>:OUTPut:POLarity <polarity> (see page 1230)	:WGEN<w>:OUTPut:POLarity? (see page 1230)	<polarity> ::= {NORMal | INVerted} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:SINGle (see page 1231)	n/a	n/a
:WGEN<w>:PERiod <period> (see page 1232)	:WGEN<w>:PERiod? (see page 1232)	<period> ::= period in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:RST (see page 1233)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage <amplitude> (see page 1234)	:WGEN<w>:VOLTage? (see page 1234)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1235)	:WGEN<w>:VOLTage:HIGH? (see page 1235)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

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

Command	Query	Options and Query Returns
:WGEN<w>:VOLTage:LOW <low> (see page 1236)	:WGEN<w>:VOLTage:LOW? (see page 1236)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1237)	:WGEN<w>:VOLTage:OFFSet? (see page 1237)	<offset> ::= offset in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 66 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1241)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 1242)	:WMEMory<r>:DISPlay? (see page 1242)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 1243)	:WMEMory<r>:LABel? (see page 1243)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 1244)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format NOTE: Math functions whose x-axis is not frequency can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1245)	:WMEMory<r>:SKEW? (see page 1245)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format

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

Command	Query	Options and Query Returns
:WMEMory<r>:YOFFset <offset>[suffix] (see page 1246)	:WMEMory<r>:YOFFset? (see page 1246)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YRANge <range>[suffix] (see page 1247)	:WMEMory<r>:YRANge? (see page 1247)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YSCale <scale>[suffix] (see page 1248)	:WMEMory<r>:YSCale? (see page 1248)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V | mV}

Syntax Elements

Number Format

NR1 specifies integer data.

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

<NL> (Line Terminator)

<NL> = new line or linefeed (ASCII decimal 10).

The line terminator, or a leading colon, will send the parser to the "root" of the command tree.

[ ] (Optional Syntax Terms)

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

{ } (Braces)

When several items are enclosed by braces, { }, only one of these elements may be selected. Vertical line (|) indicates "or". For example, {ON | OFF} indicates that only ON or OFF may be selected, not both.

::= (Defined As)

::= means "defined as".

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

< > (Angle Brackets)

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

... (Ellipsis)

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

n,..,p (Value Ranges)

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

d (Digits)

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

Quoted ASCII String

A quoted ASCII string is a string delimited by either double quotes (") or single quotes ('). Some command parameters require a quoted ASCII string. For example, when using the Keysight VISA COM library in Visual Basic, the command:

```
myScope.WriteString ":CHANNEL1:LABEL 'One'"
```

has a quoted ASCII string of:

```
'One'
```

In order to read quoted ASCII strings from query return values, some programming languages require special handling or syntax.

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. This syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 1000 bytes of data, the syntax would be

```
#800001000<1000 bytes of data> <NL>
```

8 is the number of digits that follow

00001000 is the number of bytes to be transmitted

<1000 bytes of data> is the actual data

6 Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments.
See "Introduction to Common (*) Commands" on page 176.

Table 67 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 178)	n/a	n/a
*ESE <mask> (see page 179)	*ESE? (see page 179)	<mask> ::= 0 to 255; an integer in NR1 format: Bit Weight Name Enables --- ------ ---- ---------- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete
n/a	*ESR? (see page 181)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 181)	KEYSIGHT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 184)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*OPC (see page 185)	*OPC? (see page 185)	ASCII "1" is placed in the output queue when all pending device operations have completed.
n/a	*OPT? (see page 186)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <reserved>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Frequency Response Analysis>, <Power Measurements>, <RS-232/UART Serial>, <Segmented Memory>, <Mask Test>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Educator's Kit>, <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>, <Advanced Math>, <reserved>, <reserved>, <reserved>, <reserved>, <Digital Voltmeter/Counter>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Remote Command Logging>, <reserved>, <SENT Serial>, <CAN FD Serial>, <CXPI Serial>, <NFC Trigger>, <reserved>, <reserved>, <reserved>, <Manchester/NRZ Serial>, <USB PD Serial>

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 186) (cont'd)	<All field> ::= {0 | All} <reserved> ::= 0 <Memory> ::= {0 | MEMUP} <Low Speed Serial> ::= {0 | EMBD} <Automotive Serial> ::= {0 | AUTO} <Frequency Response Analysis> ::= {0 | FRA} <Power Measurements> ::= {0 | PWR} <RS-232/UART Serial> ::= {0 | COMP} <Segmented Memory> ::= {0 | SGM} <Mask Test> ::= {0 | MASK} <Educator's Kit> ::= {0 | EDK} <Waveform Generator> ::= {0 | WAVEGEN} <MIL-1553/ARINC 429 Serial> ::= {0 | AERO} <Extended Video> ::= {0 | VID} <Advanced Math> ::= {0 | ADVMATH} <Digital Voltmeter/Counter> ::= {0 | DVMCTR} <Remote Command Logging> ::= {0 | RML} <SENT Serial> ::= {0 | SENSOR} <CAN FD Serial> ::= {0 | CANFD} <CXPI Serial> ::= {0 | CXPI} <NFC Trigger> ::= {0 | NFC} <Manchester/NRZ Serial> ::= {0 | NRZ} <USB PD Serial> ::= {0 | USBPD}
*RCL <value> (see page 188)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 189)	n/a	See *RST (Reset) (see page 189)
*SAV <value> (see page 192)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}

Table 67 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*SRE <mask> (see page 193)	*SRE? (see page 194)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables --- ------ ---- ---------- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger
n/a	*STB? (see page 195)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following: Bit Weight Name "1" Indicates --- ------ ---- --------------- 7 128 OPER Operation status condition occurred. 6 64 RQS/ MSS Instrument is requesting service. 5 32 ESB Enabled event status condition occurred. 4 16 MAV Message available. 3 8 ---- (Not used.) 2 4 MSG Message displayed. 1 2 USR User event condition occurred. 0 1 TRG A trigger occurred.
*TRG (see page 197)	n/a	n/a
n/a	*TST? (see page 198)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 199)	n/a	n/a

Introduction to Common (*) Commands

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

Common commands can be received and processed by the instrument whether they are sent over the interface as separate program messages or within other program messages. If an instrument subsystem has been selected and a common

command is received by the instrument, the instrument remains in the selected subsystem. For example, if the program message ":ACQuire:TYPE AVERage; *CLS; COUNt 256" is received by the instrument, the instrument sets the acquire type, then clears the status information and sets the average count.

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

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

*CLS (Clear Status)

 (see page 1354)

Command Syntax `*CLS`

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

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*STB (Read Status Byte)" on page 195
- "*ESE (Standard Event Status Enable)" on page 179
- "*ESR (Standard Event Status Register)" on page 181
- "*SRE (Service Request Enable)" on page 193
- ":SYSTem:ERRor" on page 1016

*ESE (Standard Event Status Enable)

(see page 1354)

Command Syntax `*ESE <mask_argument>`

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

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

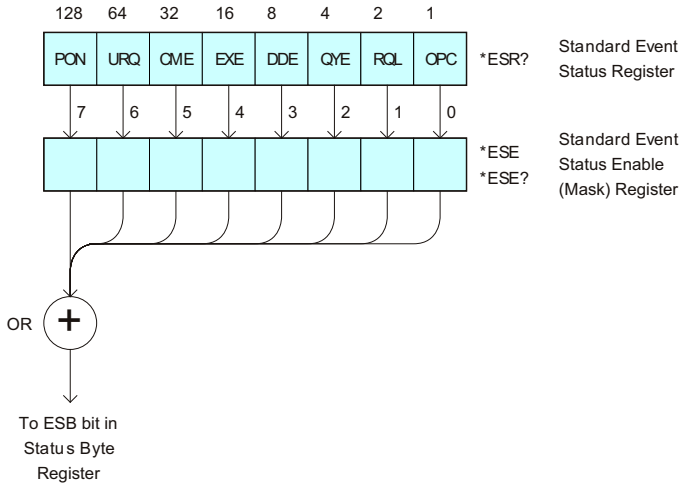


Table 68 Standard Event Status Enable (ESE)

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

Query Syntax `*ESE?`

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

Return Format

```
<mask_argument><NL>

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*ESR (Standard Event Status Register)" on page 181
- "*OPC (Operation Complete)" on page 185
- "*CLS (Clear Status)" on page 178

*ESR (Standard Event Status Register)

(see page 1354)

Query Syntax `*ESR?`

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

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

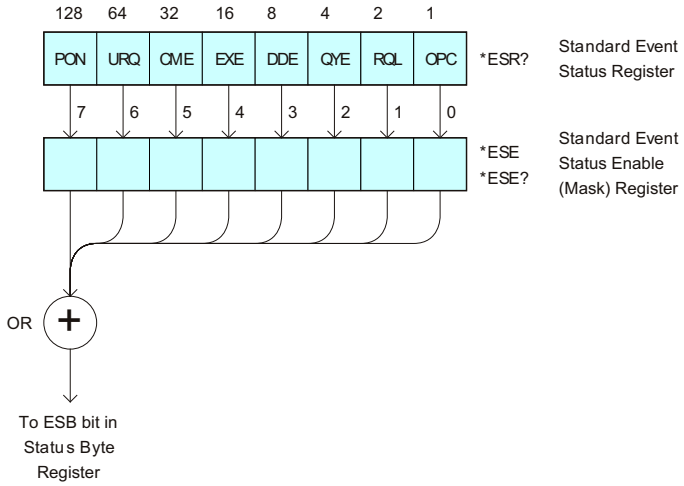


Table 69 Standard Event Status Register (ESR)

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

Return Format `<status><NL>`

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

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

See Also
- "Introduction to Common (*) Commands" on page 176
- "*ESE (Standard Event Status Enable)" on page 179
- "*OPC (Operation Complete)" on page 185
- "*CLS (Clear Status)" on page 178
- ":SYSTem:ERRor" on page 1016

*IDN (Identification Number)

C (see page 1354)

Query Syntax

```
*IDN?
```

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

Return Format

```
<manufacturer_string>,<model>,<serial_number>,X.XX.XX <NL>
<manufacturer_string> ::= KEYSIGHT TECHNOLOGIES
<model> ::= the model number of the instrument
<serial_number> ::= the serial number of the instrument
X.XX.XX ::= the software revision of the instrument
```

See Also

- "Introduction to Common (*) Commands" on page 176
- "*OPT (Option Identification)" on page 186
- ":SYSTem:PERSona[:MANufacturer]" on page 1019
- ":SYSTem:PERSona[:MANufacturer]:DEFault" on page 1020

*LRN (Learn Device Setup)

C (see page 1354)

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 1034) query, except that it contains ":SYST:SET " before the binary block data. The query result is a valid command that can be used to restore instrument settings at a later time.

Return Format

```
<learn_string><NL>

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

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

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

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*RCL (Recall)" on page 188
- "*SAV (Save)" on page 192
- ":SYSTem:SETup" on page 1034

*OPC (Operation Complete)

 (see page 1354)

Command Syntax

```
*OPC
```

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

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

NOTE

The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, in certain situations. Bit 4 in the Operation Status Condition Register shows whether the remote user interface is enabled or disabled. For more information, see ":OPERegister:CONDition (Operation Status Condition Register)" on page 223.

Query Syntax

```
*OPC?
```

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

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

Return Format

```
<complete><NL>

<complete> ::= 1
```

See Also

- "Introduction to Common (*) Commands" on page 176
- "*ESE (Standard Event Status Enable)" on page 179
- "*ESR (Standard Event Status Register)" on page 181
- "*CLS (Clear Status)" on page 178
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- Chapter 4, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 63
- "Synchronization with an Averaging Acquisition" on page 1340
- "Set Up the Oscilloscope" on page 1334
- "Example: Blocking and Polling Synchronization" on page 1342
- "Example: Waiting for IO Operation Complete" on page 1330

*OPT (Option Identification)

(see page 1354)

Query Syntax `*OPT?`

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

Return Format `0,0,<license info>`

```
<license info> ::= <All field>, <reserved>, <reserved>, <reserved>,
   <reserved>, <Memory>, <Low Speed Serial>, <Automotive Serial>,
   <reserved>, <Frequency Response Analysis>, <Power Measurements>,
   <RS-232/UART Serial>, <Segmented Memory>, <Mask Test>, <reserved>,
   <reserved>, <reserved>, <reserved>, <reserved>, <reserved>,
   <reserved>, <reserved>, <reserved>, <reserved>, <Educator's Kit>,
   <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>,
   <Advanced Math>, <reserved>, <reserved>, <reserved>, <reserved>,
   <Digital Voltmeter/Counter>, <reserved>, <reserved>, <reserved>,
   <reserved>, <reserved>, <Remote Command Logging>, <reserved>,
   <SENT Serial>, <CAN FD Serial>, <CXPI Serial>, <NFC Trigger>,
   <reserved>, <reserved>, <reserved>, <Manchester/NRZ Serial>,
   <USB PD Serial>

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

<reserved> ::= 0

<Memory> ::= {0 | memMax}

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

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

<Frequency Response Analysis> ::= {0 | FRA}

<Power Measurements> ::= {0 | PWR}

<RS-232/UART Serial> ::= {0 | COMP}

<Segmented Memory> ::= {0 | SGM}

<Mask Test> ::= {0 | MASK}

<Educator's Kit> ::= {0 | EDK}

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

<MIL-1553/ARINC 429 Serial> ::= {0 | AERO}

<Extended Video> ::= {0 | VID}

<Advanced Math> ::= {0 | ADVMATH}

<Digital Voltmeter/Counter>> ::= {0 | DVMCTR}

<Remote Command Logging> ::= {0 | RML}

<SENT Serial> ::= {0 | SENSOR}
```

```
<CAN FD Serial> ::= {0 | CANFD}
<CXPI Serial> ::= {0 | CXPI}
<NFC Trigger> ::= {0 | NFC}
<Manchester/NRZ Serial> ::= {0 | NRZ}
<USB PD Serial> ::= {0 | USBPD}
```

The *OPT? query returns the following:

Module	Module Id
No modules attached	0,0,0,0,0,0,0,0,0,0,0,0,0,SGM,0,0,0,0,0,0,0,0,0,0,0,0,0,0,ADVMATH,0,0,0,0,0,0,0,0,0,0,RML,0,0,0,0,0,0,0,0,0,0,0,0,0

See Also

- "Introduction to Common (*) Commands" on page 176
- "*IDN (Identification Number)" on page 183

*RCL (Recall)

(see page 1354)

Command Syntax

```
*RCL <value>

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

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*SAV (Save)" on page 192

*RST (Reset)

(see page 1354)

Command Syntax `*RST`

The *RST command places the instrument in a known state. This is the same as pressing **[Save/Recall] > Default/Erase > Factory Default** on the front panel.

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

Reset conditions are:

Acquire Menu	
Mode	Normal
Averaging	Off
# Averages	8

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

Cursor Menu	
Source	Channel 1

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

See Also
- "Introduction to Common (*) Commands" on page 176
- ":SYSTem:PRESet" on page 1023

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

*SAV (Save)

 (see page 1354)

Command Syntax

```
*SAV <value>

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

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*RCL (Recall)" on page 188

*SRE (Service Request Enable)

(see page 1354)

Command Syntax

```
*SRE <mask>

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

The *SRE command sets the bits in the Service Request Enable Register. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A zero disables the bit.

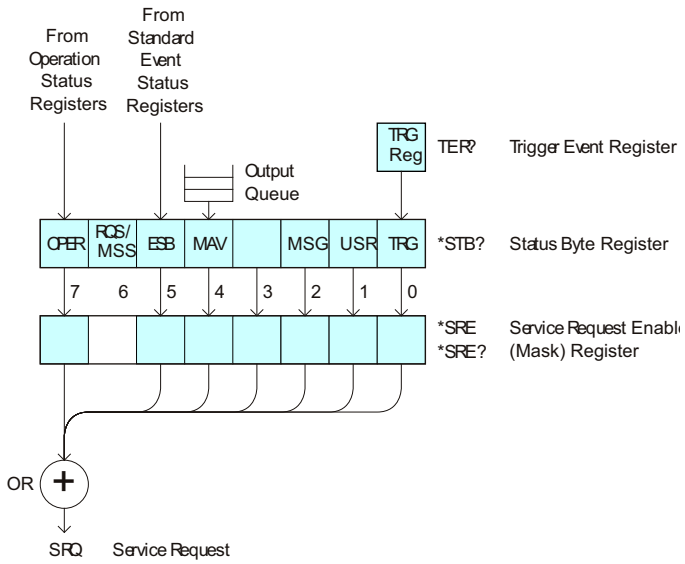


Table 70 Service Request Enable Register (SRE)

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

Query Syntax

```
*SRE?
```

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

Return Format

```
<mask><NL>

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

See Also

- "Introduction to Common (*) Commands" on page 176
- "*STB (Read Status Byte)" on page 195
- "*CLS (Clear Status)" on page 178

*STB (Read Status Byte)

(see page 1354)

Query Syntax `*STB?`

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

Return Format `<value><NL>`

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

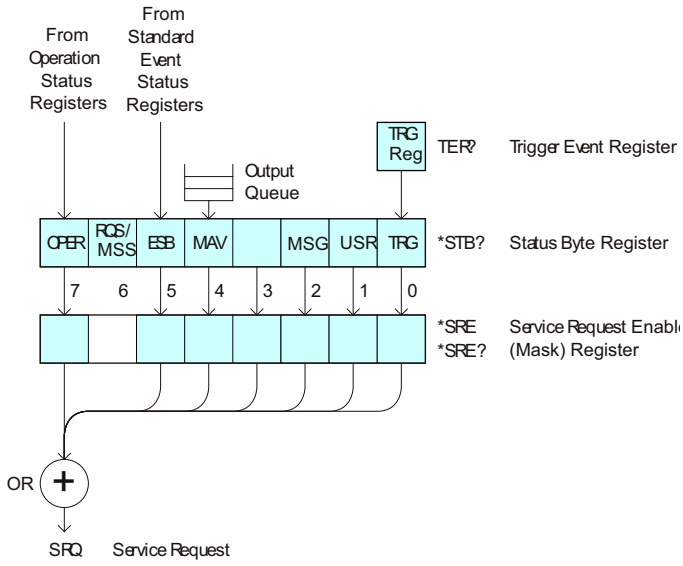


Table 71 Status Byte Register (STB)

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

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

See Also
- "Introduction to Common (*) Commands" on page 176
- "*SRE (Service Request Enable)" on page 193

*TRG (Trigger)

 (see page 1354)

Command Syntax `*TRG`

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

See Also

- "Introduction to Common (*) Commands" on page 176
- ":DIGitize" on page 211
- ":RUN" on page 233
- ":STOP" on page 237

*TST (Self Test)

(see page 1354)

Query Syntax `*TST?`

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

Return Format `<result><NL>`

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

See Also

- "Introduction to Common (*) Commands" on page 176

*WAI (Wait To Continue)

 (see page 1354)

Command Syntax `*WAI`

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

See Also

- "Introduction to Common (*) Commands" on page 176

7 Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree. See "Introduction to Root (:) Commands" on page 203.

Table 72 Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see page 204)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 205)	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format
:AUToscale:AMODE <value> (see page 207)	:AUToscale:AMODE? (see page 207)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 208)	:AUToscale:CHANnels? (see page 208)	<value> ::= {ALL | DISPlayed}}
:AUToscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 209)	:AUToscale:FDEBug? (see page 209)	{0 | 1}
:BLANk [<source>] (see page 210)	n/a	<source> ::= {CHANnel<n>} | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 72 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,..,<source>]] (see page 211)	n/a	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2}} <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:HWEenable <n> (see page 213)	:HWEenable? (see page 213)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDition? (see page 215)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENt]? (see page 216)	<n> ::= 16-bit integer in NR1 format
:MTEenable <n> (see page 217)	:MTEenable? (see page 217)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENt]? (see page 219)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 221)	:OPEE? (see page 222)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister:CONDition? (see page 223)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister[:EVENt]? (see page 226)	<n> ::= 15-bit integer in NR1 format
:OVLenable <mask> (see page 229)	:OVLenable? (see page 230)	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input --- ------ ---------- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see page 231)	<value> ::= integer in NR1 format. See OVLenable for <value>

Table 72 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:RUN (see page 233)	n/a	n/a
n/a	:SERial (see page 234)	<return value> ::= unquoted string containing serial number
:SINGle (see page 235)	n/a	n/a
n/a	:STATus? <display> (see page 236)	{0 | 1} <display> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 237)	n/a	n/a
n/a	:TER? (see page 238)	{0 | 1}
:VIEW <source> (see page 239)	n/a	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | SBUS{1 | 2} | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Introduction to Root (:) Commands

Root level commands control many of the basic operations of the instrument. These commands are always recognized by the parser if they are prefixed with a colon, regardless of current command tree position. After executing a root-level command, the parser is positioned at the root of the command tree.

:AER (Arm Event Register)

(see page 1354)

Query Syntax `:AER?`

The AER query reads the Arm Event Register. After the Arm Event Register is read, it is cleared. A "1" indicates the trigger system is in the armed state, ready to accept a trigger.

The Armed Event Register is summarized in the Wait Trig bit of the Operation Status Event Register. A Service Request can be generated when the Wait Trig bit transitions and the appropriate enable bits have been set in the Operation Status Enable Register (OPEE) and the Service Request Enable Register (SRE).

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:AUToscale

(see page 1354)

Command Syntax

```
:AUToscale

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

<source> ::= CHANnel<n>

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

The <source> parameter may be repeated up to 5 times.
```

The :AUToscale command evaluates all input signals and sets the correct conditions to display the signals. This is the same as pressing the **[Auto Scale]** key on the front panel.

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see ":AUToscale:CHANnels" on page 208) is set to DISPlayed channels. Then, the autoscale is performed.

When the :AUToscale command is sent, the following conditions are affected and actions are taken:

- Thresholds.
- Channels with activity around the trigger point are turned on, others are turned off.
- Channels are reordered on screen; analog channel 1 first, followed by the remaining analog channels.
- Delay is set to 0 seconds.
- Time/Div.

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

- Cursors.
- Measurements.
- Math waveforms.
- Reference waveforms.
- Zoomed (delayed) time base mode.

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AUToscale:CHANnels" on page 208

- ":AUToscale:AMODE" on page 207

Example Code

```
' AUTOSCALE - This command evaluates all the input signals and sets
' the correct conditions to display all of the active signals.
myScope.WriteString ":AUToscale"    ' Same as pressing Auto Scale key.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:AUToscale:AMODE

 (see page 1354)

Command Syntax

```
:AUToscale:AMODE <value>

<value> ::= {NORMal | CURRent}
```

The :AUToscale:AMODE command specifies the acquisition mode that is set by subsequent :AUToscales.

- When NORMal is selected, an :AUToscale command sets the NORMal acquisition type and the RTIMe (real-time) acquisition mode.
- When CURRent is selected, the current acquisition type and mode are kept on subsequent :AUToscales.

Use the :ACQuire:TYPE and :ACQuire:MODE commands to set the acquisition type and mode.

Query Syntax

```
:AUToscale:AMODE?
```

The :AUToscale:AMODE? query returns the autoscale acquire mode setting.

Return Format

```
<value><NL>

<value> ::= {NORM | CURR}
```

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AUToscale" on page 205
- ":AUToscale:CHANnels" on page 208
- ":ACQuire:TYPE" on page 257
- ":ACQuire:MODE" on page 248

:AUToscale:CHANnels

(see page 1354)

Command Syntax

```
:AUToscale:CHANnels <value>

<value> ::= {ALL | DISPlayed}
```

The :AUToscale:CHANnels command specifies which channels will be displayed on subsequent :AUToscales.

- When ALL is selected, all channels that meet the requirements of :AUToscale will be displayed.
- When DISPlayed is selected, only the channels that are turned on are autoscaled.

Use the :VIEW or :BLANk root commands to turn channels on or off.

Query Syntax

```
:AUToscale:CHANnels?
```

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

Return Format

```
<value><NL>

<value> ::= {ALL | DISP}
```

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AUToscale" on page 205
- ":AUToscale:AMODE" on page 207
- ":VIEW" on page 239
- ":BLANk" on page 210

:AUToscale:FDEBug

N (see page 1354)

Command Syntax

```
:AUToscale:FDEBug <on_off>

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

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

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

Channel coupling is maintained for easy viewing of oscillating signals.

Query Syntax

```
:AUToscale:FDEBug?
```

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

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AUToscale" on page 205

:BLANk

(see page 1354)

Command Syntax

```
:BLANk [<source>]

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

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

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

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

The :BLANk command turns off (stops displaying) the specified channel, math function, serial decode bus, or reference waveform location. The :BLANk command with no parameter turns off all sources.

NOTE To turn on (start displaying) a channel, etc., use the :VIEW command. The DISPlay commands, :CHANnel<n>:DISPlay, :FUNCtion:DISPlay, :SBUS<n>:DISPlay, or :WMEMory<r>:DISPlay, are the preferred method to turn on/off a channel, etc.

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":DISPlay:CLEar" on page 320
- ":CHANnel<n>:DISPlay" on page 277
- ":FUNCtion<m>:DISPlay" on page 369
- ":SBUS<n>:DISPlay" on page 724
- ":WMEMory<r>:DISPlay" on page 1242
- ":STATus" on page 236
- ":VIEW" on page 239

Example Code

- "Example Code" on page 239

:DIGitize

(see page 1354)

Command Syntax

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

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

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

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

The <source> parameter may be repeated up to 5 times.
```

The :DIGitize command is a specialized RUN command. It causes the instrument to acquire waveforms according to the settings of the :ACQuire commands subsystem. When the acquisition is complete, the instrument is stopped.

If no argument is given, :DIGitize acquires the channels currently displayed. If no channels are displayed, all channels are acquired.

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

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

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":RUN" on page 233
- ":SINGle" on page 235
- ":STOP" on page 237
- ":TIMebase:MODE" on page 1041
- Chapter 8, ":ACQuire Commands," starting on page 241
- Chapter 32, ":WAVeform Commands," starting on page 1157
- Chapter 4, "Sequential (Blocking) vs. Overlapped Commands," starting on page 63
- "Example: Checking for Armed Status" on page 1325

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:HWEenable (Hardware Event Enable Register)

(see page 1354)

Command Syntax

```
:HWEenable <mask>

<mask> ::= 16-bit integer
```

The :HWEenable command sets a mask in the Hardware Event Enable register. Set any of the following bits to "1" to enable bit 12 in the Operation Status Condition Register and potentially cause an SRQ (Service Request interrupt to be generated.

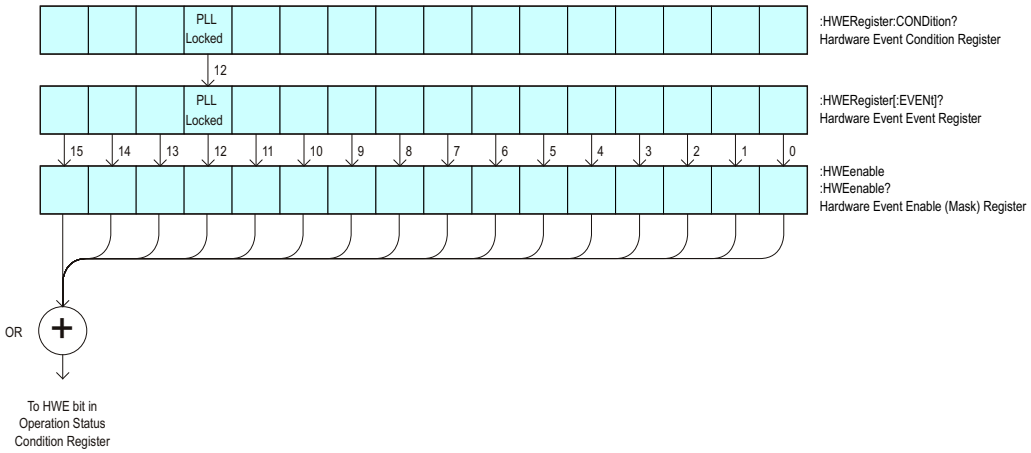


Table 73 Hardware Event Enable Register (HWEenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Query Syntax

```
:HWEenable?
```

The :HWEenable? query returns the current value contained in the Hardware Event Enable register as an integer number.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AER (Arm Event Register)" on page 204
- ":CHANnel<n>:PROTection" on page 290

- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:HWERegister:CONDition (Hardware Event Condition Register)

N (see page 1354)

Query Syntax `:HWERegister:CONDition?`

The :HWERegister:CONDition? query returns the integer value contained in the Hardware Event Condition Register.

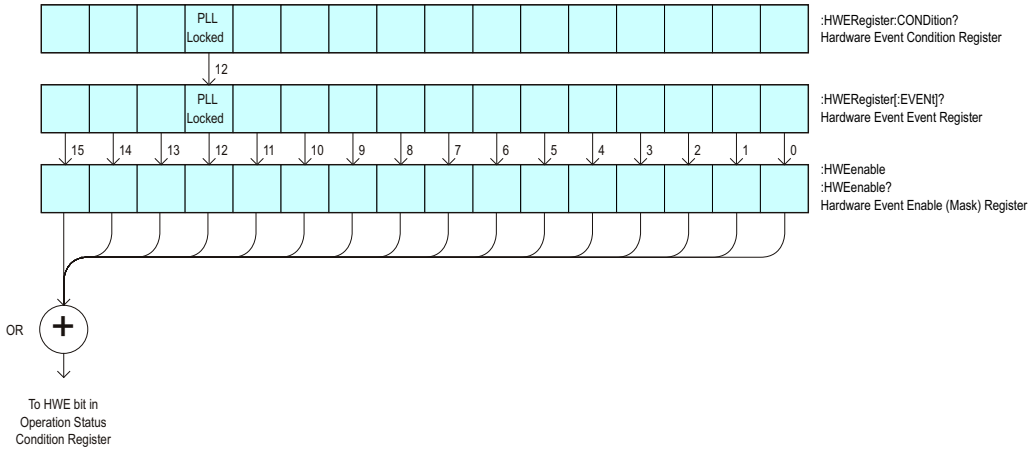


Table 74 Hardware Event Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:HWERegister[:EVENt] (Hardware Event Event Register)

(see page 1354)

Query Syntax `:HWERegister[:EVENt]?`

The :HWERegister[:EVENt]? query returns the integer value contained in the Hardware Event Event Register.

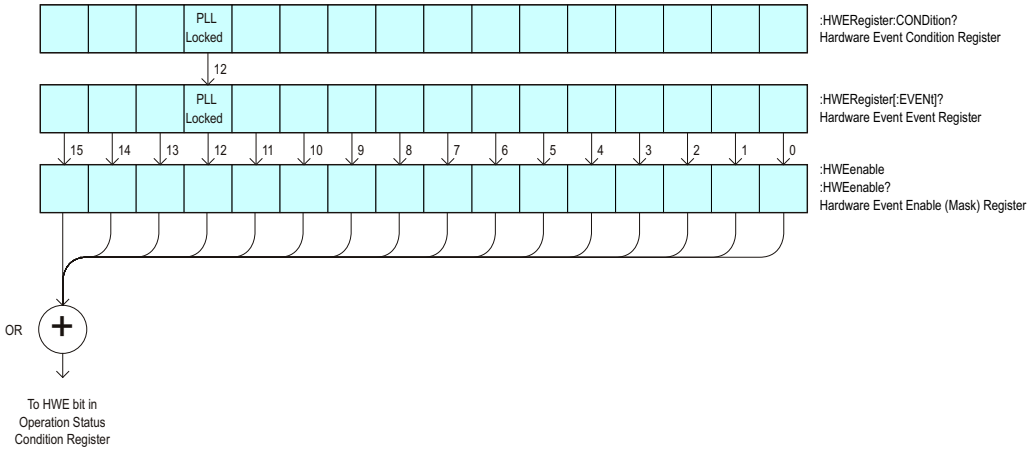


Table 75 Hardware Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:MTEenable (Mask Test Event Enable Register)

N (see page 1354)

Command Syntax

```
:MTEenable <mask>

<mask> ::= 16-bit integer
```

The :MTEenable command sets a mask in the Mask Test Event Enable register. Set any of the following bits to "1" to enable bit 9 in the Operation Status Condition Register and potentially cause an SRQ (Service Request) interrupt to be generated.

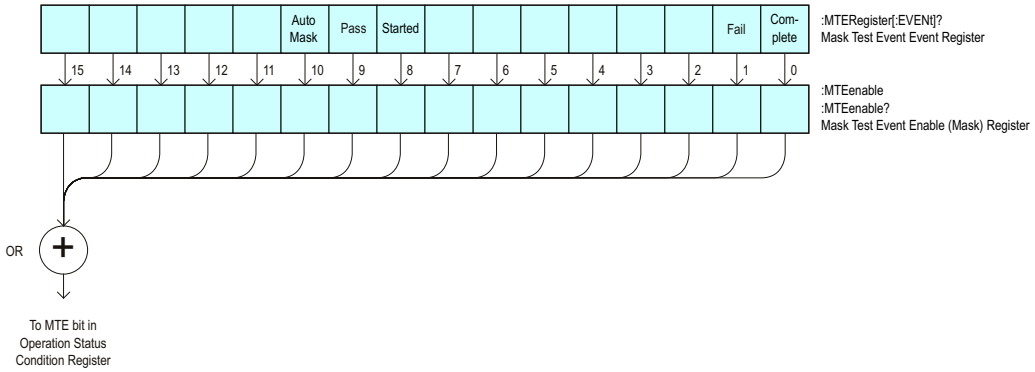


Table 76 Mask Test Event Enable Register (MTEenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	Pass	Mask Test Pass	Mass test passed.
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	Mask test failed.
0	Complete	Mask Test Complete	Mask test is complete.

Query Syntax

```
:MTEenable?
```

The :MTEenable? query returns the current value contained in the Mask Test Event Enable register as an integer number.

Return Format

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

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AER (Arm Event Register)" on page 204
- ":CHANnel<n>:PROTection" on page 290
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:MTERegister[:EVENt] (Mask Test Event Event Register)

N (see page 1354)

Query Syntax `:MTERegister[:EVENt]?`

The :MTERegister[:EVENt]? query returns the integer value contained in the Mask Test Event Event Register and clears the register.

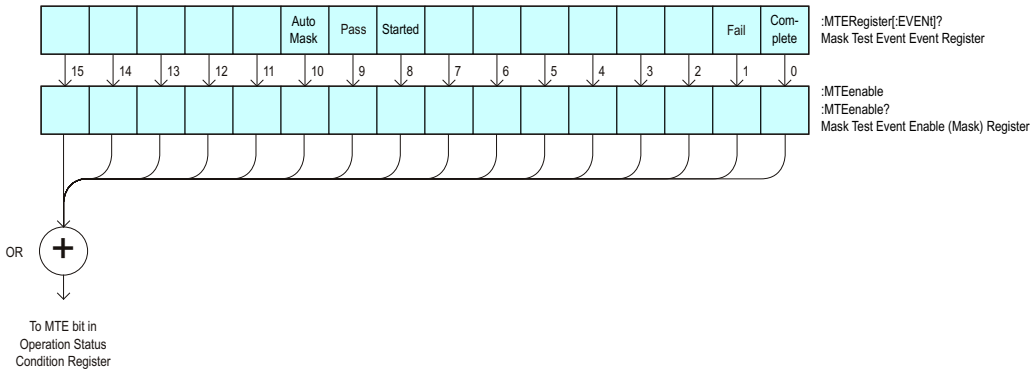


Table 77 Mask Test Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	Pass	Mask Test Pass	The mask test passed.
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	The mask test failed.
0	Complete	Mask Test Complete	The mask test is complete.

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223

- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:OPEE (Operation Status Enable Register)

C (see page 1354)

Command Syntax

```
:OPEE <mask>

<mask> ::= 15-bit integer
```

The :OPEE command sets a mask in the Operation Status Enable register. Set any of the following bits to "1" to enable bit 7 in the Status Byte Register and potentially cause an SRQ (Service Request) interrupt to be generated.

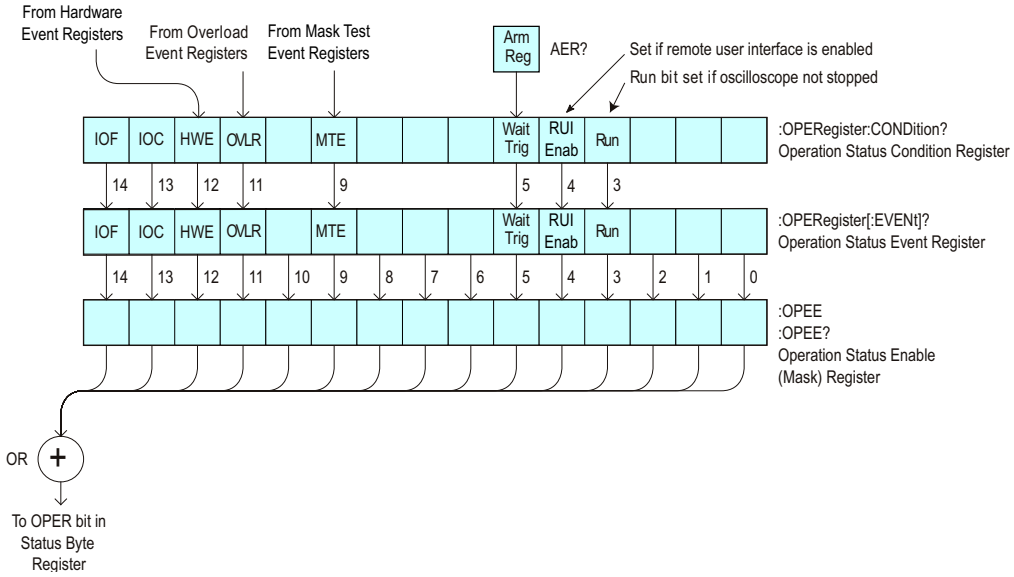


Table 78 Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
14	IOF	IO Operation Failed	Event only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.
13	IOC	IO Operation Complete	Event when any IO operation completes. IO operations are any remote data request using any interface (USB, LAN, or GPIB). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	Event when 50Ω input overload occurs.

Table 78 Operation Status Enable Register (OPEE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
10	---	---	(Not used.)
9	MTE	Mask Test Event	Event when mask test event occurs.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)
3	Run	Running	Event when the oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Query Syntax

```
:OPEE?
```

The :OPEE? query returns the current value contained in the Operation Status Enable register as an integer number.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":AER (Arm Event Register)" on page 204
- ":CHANnel<n>:PROTection" on page 290
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193
- "Operation Status Event Register (:OPERegister[:EVENt])" on page 1315
- "Example: Checking for Armed Status" on page 1325
- "Example: Waiting for IO Operation Complete" on page 1330

:OPERegister:CONDition (Operation Status Condition Register)

(see page 1354)

Query Syntax `:OPERegister:CONDition?`

The :OPERegister:CONDition? query returns the integer value contained in the Operation Status Condition Register.

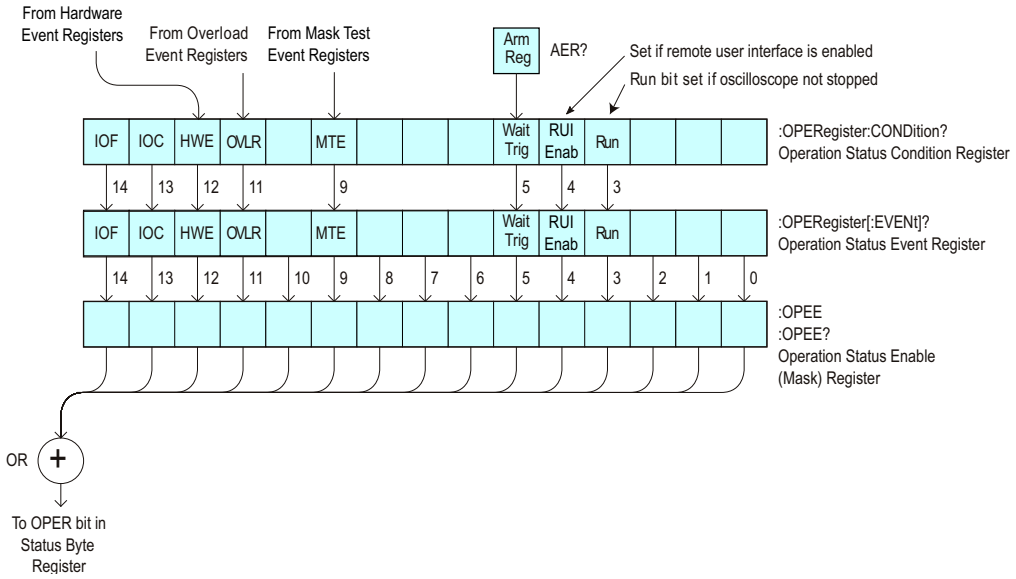


Table 79 Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14	IOF	IO Operation Failed	Event only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.
13	IOC	IO Operation Complete	Event when any IO operation completes. IO operations are any remote data request using any interface (USB, LAN, or GPIB). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)

Table 79 Operation Status Condition Register (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
9	MTE	Mask Test Event	A mask test event has occurred.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	RUI Enab	Remote Enabled	Shows whether the remote user interface is enabled. The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, for example when: ▪ a modal dialog box is open ▪ the user is being prompted ▪ all segments are being analyzed ▪ there is a channel overload ▪ certain compliance applications are running When disabled, commands or queries are accepted, but "settings conflict" errors are returned. The status model is the only part of the remote user interface that is enabled. To determine when the remote interface is re-enabled, you can read this bit or wait for the event that gets generated when its status goes from disabled to enabled.
3	Run	Running	The oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193
- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 219

- ":MTEenable (Mask Test Event Enable Register)" on page 217
- "*OPC (Operation Complete)" on page 185
- "Operation Status Condition Register (:OPERegister:CONDition)" on page 1317
- "Example: Checking for Armed Status" on page 1325
- "Example: Waiting for IO Operation Complete" on page 1330

:OPERegister[:EVENt] (Operation Status Event Register)

(see page 1354)

Query Syntax `:OPERegister[:EVENt]?`

The :OPERegister[:EVENt]? query reads and clears the integer value contained in the Operation Status Event Register.

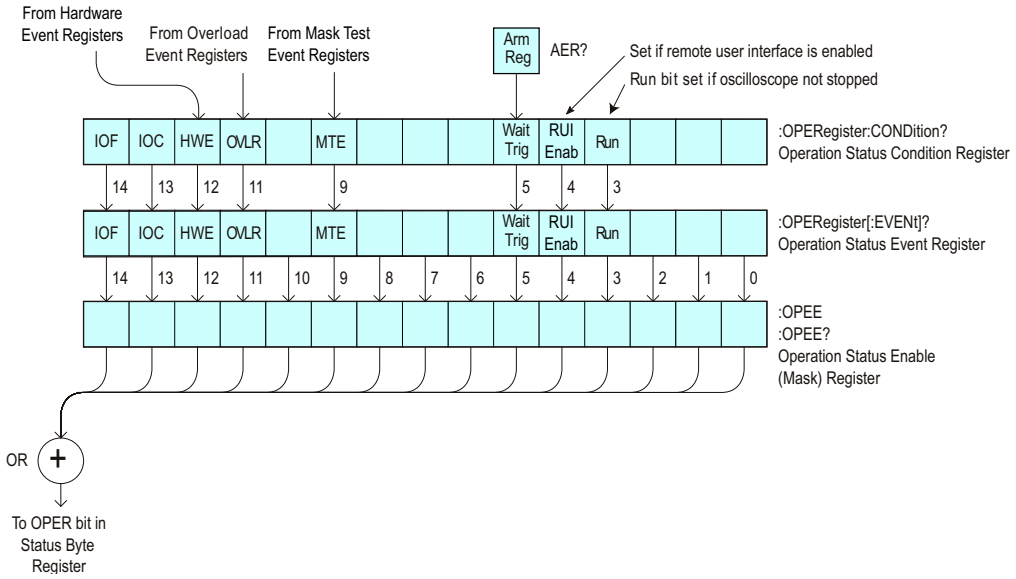


Table 80 Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14	IOF	IO Operation Failed	Event only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.
13	IOC	IO Operation Complete	Event when any IO operation completes. IO operations are any remote data request using any interface (USB, LAN, or GPIB). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)

Table 80 Operation Status Event Register (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
9	MTE	Mask Test Event	A mask test event has occurred.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	RUI Enab	Remote Enabled	The remote user interface has gone from a disabled state to an enabled state. The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, for example when: ▪ a modal dialog box is open ▪ the user is being prompted ▪ all segments are being analyzed ▪ there is a channel overload ▪ certain compliance applications are running When disabled, commands or queries are accepted, but "settings conflict" errors are returned. The status model is the only part of the remote user interface that is enabled. To determine when the remote interface is re-enabled, you can read this bit or wait for the event that gets generated when its status goes from disabled to enabled.
3	Run	Running	The oscilloscope has gone from a stop state to a single or running state.
2-0	---	---	(Not used.)

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223
- ":OVLenable (Overload Event Enable Register)" on page 229
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 219
- ":MTEenable (Mask Test Event Enable Register)" on page 217
- "*OPC (Operation Complete)" on page 185
- "Operation Status Event Register (:OPERegister[:EVENt])" on page 1315
- "Example: Checking for Armed Status" on page 1325
- "Example: Waiting for IO Operation Complete" on page 1330

:OVLenable (Overload Event Enable Register)

(see page 1354)

Command Syntax

```
:OVLenable <enable_mask>

<enable_mask> ::= 16-bit integer
```

The overload enable mask is an integer representing an input as described in the following table.

The :OVLenable command sets the mask in the Overload Event Enable Register and enables the reporting of the Overload Event Register. If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. If enabled, such an event will set bit 11 in the Operation Status Register.

NOTE You can set analog channel input impedance to 50Ω on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models. On these same bandwidth models, if there are only two analog channels, you can also set external trigger input impedance to 50Ω.

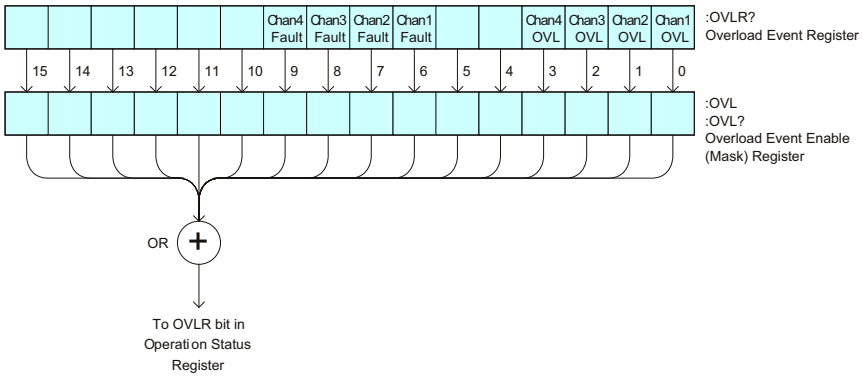


Table 81 Overload Event Enable Register (OVL)

Bit	Description	When Set (1 = High = True), Enables:
15-10	---	(Not used.)
9	Channel 4 Fault	Event when fault occurs on Channel 4 input.
8	Channel 3 Fault	Event when fault occurs on Channel 3 input.
7	Channel 2 Fault	Event when fault occurs on Channel 2 input.
6	Channel 1 Fault	Event when fault occurs on Channel 1 input.
5-4	---	(Not used.)
3	Channel 4 OVL	Event when overload occurs on Channel 4 input.

Table 81 Overload Event Enable Register (OVL) (continued)

Bit	Description	When Set (1 = High = True), Enables:
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.
1	Channel 2 OVL	Event when overload occurs on Channel 2 input.
0	Channel 1 OVL	Event when overload occurs on Channel 1 input.

Query Syntax

```
:OVLenable?
```

The :OVLenable query returns the current enable mask value contained in the Overload Event Enable Register.

Return Format

```
<enable_mask><NL>

<enable_mask> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226
- ":OVLRegister (Overload Event Register)" on page 231
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:OVLRegister (Overload Event Register)

C (see page 1354)

Query Syntax `:OVLRegister?`

The :OVLRegister query returns the overload protection value stored in the Overload Event Register (OVLR). If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. A "1" indicates an overload has occurred.

NOTE You can set analog channel input impedance to 50Ω on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models. On these same bandwidth models, if there are only two analog channels, you can also set external trigger input impedance to 50Ω.

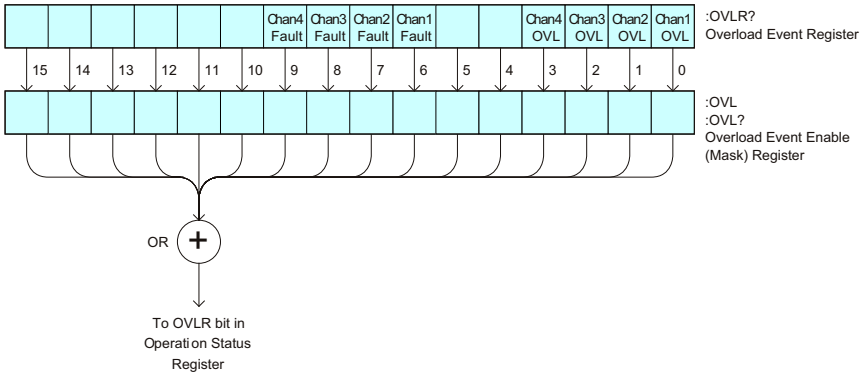


Table 82 Overload Event Register (OVLR)

Bit	Description	When Set (1 = High = True), Indicates:
15-10	---	(Not used.)
9	Channel 4 Fault	Fault has occurred on Channel 4 input.
8	Channel 3 Fault	Fault has occurred on Channel 3 input.
7	Channel 2 Fault	Fault has occurred on Channel 2 input.
6	Channel 1 Fault	Fault has occurred on Channel 1 input.
5-4	---	(Not used.)
3	Channel 4 OVL	Overload has occurred on Channel 4 input.
2	Channel 3 OVL	Overload has occurred on Channel 3 input.
1	Channel 2 OVL	Overload has occurred on Channel 2 input.
0	Channel 1 OVL	Overload has occurred on Channel 1 input.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":CHANnel<n>:PROTection" on page 290
- ":OPEE (Operation Status Enable Register)" on page 221
- ":OVLenable (Overload Event Enable Register)" on page 229
- "*STB (Read Status Byte)" on page 195
- "*SRE (Service Request Enable)" on page 193

:RUN

(see page 1354)

Command Syntax

```
:RUN
```

The :RUN command starts repetitive acquisitions. This is the same as pressing the Run key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 203
- ":SINGle" on page 235
- ":STOP" on page 237

Example Code

```
' RUN_STOP - (not executed in this example)
'   - RUN starts the data acquisition for the active waveform display.
'   - STOP stops the data acquisition and turns off AUTOSTORE.
' myScope.WriteString ":RUN"    ' Start data acquisition.
' myScope.WriteString ":STOP"    ' Stop the data acquisition.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:SERial

 (see page 1354)

Query Syntax `:SERial?`

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

Return Format: `Unquoted string<NL>`

See Also

- "Introduction to Root (:) Commands" on page 203

:SINGle

 (see page 1354)

Command Syntax `:SINGle`

The :SINGle command causes the instrument to acquire a single trigger of data. This is the same as pressing the Single key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 203
- ":RUN" on page 233
- ":STOP" on page 237

:STATus

(see page 1354)

Query Syntax

```
:STATus? <source>

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

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

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

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

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

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":BLANk" on page 210
- ":CHANnel<n>:DISPlay" on page 277
- ":FUNCtion<m>:DISPlay" on page 369
- ":SBUS<n>:DISPlay" on page 724
- ":WMEMory<r>:DISPlay" on page 1242
- ":VIEW" on page 239

:STOP

 (see page 1354)

Command Syntax

```
:STOP
```

The :STOP command stops the acquisition. This is the same as pressing the Stop key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 203
- ":RUN" on page 233
- ":SINGle" on page 235

Example Code

- "Example Code" on page 233

:TER (Trigger Event Register)

 (see page 1354)

Query Syntax `:TER?`

The :TER? query reads the Trigger Event Register. After the Trigger Event Register is read, it is cleared. A one indicates a trigger has occurred. A zero indicates a trigger has not occurred.

The Trigger Event Register is summarized in the TRG bit of the Status Byte Register (STB). A Service Request (SRQ) can be generated when the TRG bit of the Status Byte transitions, and the TRG bit is set in the Service Request Enable register. The Trigger Event Register must be cleared each time you want a new service request to be generated.

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 203
- "*SRE (Service Request Enable)" on page 193
- "*STB (Read Status Byte)" on page 195

:VIEW

(see page 1354)

Command Syntax

```
:VIEW <source>

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

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

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

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

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

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

See Also

- "Introduction to Root (:) Commands" on page 203
- ":BLANk" on page 210
- ":CHANnel<n>:DISPlay" on page 277
- ":FUNCtion<m>:DISPlay" on page 369
- ":SBUS<n>:DISPlay" on page 724
- ":WMEMory<r>:DISPlay" on page 1242
- ":STATus" on page 236

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

8 :ACQuire Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 242.

Table 83 :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see page 244)	{1 | 0}
:ACQuire:COMPlete <complete> (see page 245)	:ACQuire:COMPlete? (see page 245)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 246)	:ACQuire:COUNt? (see page 246)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALias <mode> (see page 247)	:ACQuire:DAALias? (see page 247)	<mode> ::= {DISable | AUTO}
:ACQuire:MODE <mode> (see page 248)	:ACQuire:MODE? (see page 248)	<mode> ::= {RTIMe | ETIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 249)	<# points> ::= an integer in NR1 format
:ACQuire:RSIGnal <ref_signal_mode> (see page 250)	:ACQuire:RSIGnal? (see page 250)	<ref_signal_mode> ::= {OFF | OUT | IN}
:ACQuire:SEGMented:ANALyze (see page 251)	n/a	n/a (with SGM license)
:ACQuire:SEGMented:COUNt <count> (see page 252)	:ACQuire:SEGMented:COUNt? (see page 252)	<count> ::= an integer from 2 to 1000 in NR1 format (with SGM license)
:ACQuire:SEGMented:INDex <index> (see page 253)	:ACQuire:SEGMented:INDex? (see page 253)	<index> ::= an integer from 1 to 1000 in NR1 format (with SGM license)

Table 83 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:ACQuire:SRATe? (see page 256)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 257)	:ACQuire:TYPE? (see page 257)	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Introduction to :ACQuire Commands

The ACQuire subsystem controls the way in which waveforms are acquired. These acquisition types are available: normal, averaging, peak detect, and high resolution.

Normal

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

Averaging

The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 2 to 65536. The COUNt value determines the number of averages that must be acquired.

High-Resolution

The :ACQuire:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range. Instead of decimating samples, they are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.

Peak Detect

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

Reporting the Setup

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

Return Format

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

```
:ACQ:MODE RTIM;TYPE NORM;COMP 100;COUNT 8;SEGM:COUN 2
```

:ACQuire:AALias

(see page 1354)

Query Syntax `:ACQuire:AALias?`

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

Return Format `<value><NL>`

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:DAALias" on page 247

:ACQuire:COMPlete

 (see page 1354)

Command Syntax

```
:ACQuire:COMPlete <complete>

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

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

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

Query Syntax

```
:ACQuire:COMPlete?
```

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

Return Format

```
<completion_criteria><NL>

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:TYPE" on page 257
- ":DIGitize" on page 211
- ":WAVeform:POINts" on page 1170

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:ACQuire:COUNt

(see page 1354)

Command Syntax

```
:ACQuire:COUNt <count>

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

In averaging mode, the :ACQuire:COUNt command specifies the number of values to be averaged for each time bucket before the acquisition is considered to be complete for that time bucket. When :ACQuire:TYPE is set to AVERage, the count can be set to any value from 2 to 65536.

NOTE

The :ACQuire:COUNt 1 command has been deprecated. The AVERage acquisition type with a count of 1 is functionally equivalent to the HRESolution acquisition type; however, you should select the high-resolution acquisition mode with the :ACQuire:TYPE HRESolution command instead.

Query Syntax

```
:ACQuire:COUNT?
```

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

Return Format

```
<count_argument><NL>

<count_argument> ::= an integer from 2 to 65536 in NR1 format
```

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:TYPE" on page 257
- ":DIGitize" on page 211
- ":WAVeform:COUNt" on page 1166

:ACQuire:DAALias

(see page 1354)

Command Syntax

```
:ACQuire:DAALias <mode>

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

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

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

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

Query Syntax

```
:ACQuire:DAALias?
```

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

Return Format

```
<mode><NL>

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:AALias" on page 244

:ACQuire:MODE

(see page 1354)

Command Syntax

```
:ACQuire:MODE <mode>

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

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

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

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

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

Query Syntax

```
:ACQuire:MODE?
```

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

Return Format

```
<mode_argument><NL>

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:TYPE" on page 257

:ACQuire:POINts

(see page 1354)

Query Syntax `:ACQuire:POINts?`

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

Return Format `<points_argument><NL>`

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":DIGitize" on page 211
- ":WAVeform:POINts" on page 1170

:ACQuire:RSIGnal

 (see page 1354)

Command Syntax

```
:ACQuire:RSIGnal <ref_signal_mode>

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

The :ACQuire:RSIGnal command selects the reference signal mode. A common reference signal can be used by multiple instruments to synchronize their timebases.

- The OFF mode disables the oscilloscope's REF I/O MMCX connector. The oscilloscope's internal 10 MHz reference signal is used, and there is no timebase synchronization.
- The OUT mode outputs the oscilloscope's internal 10 MHz reference signal to the REF I/O MMCX connector.
- The IN mode synchronizes the oscilloscope's timebase to an external 10 MHz square or sine wave signal that is input to the MMCX connector labeled REF I/O. The amplitude must be from - 5 dBm to 17 dBm (356 mVpp to 4.48 Vpp).

CAUTION **Do not apply more than 20 dBm Max (6.32 Vpp Max) at the REF I/O MMCX connector on the front panel or damage to the instrument may occur.**

- The PXIE mode synchronizes the oscilloscope's timebase to the PXIe chassis' 100 MHz reference signal. In this mode, the oscilloscope's timebase is synchronized with other instruments in the PXIe chassis that are also using the chassis' 100 MHz reference signal, and the oscilloscope's REF I/O MMCX connector is disabled.

Query Syntax

```
:ACQuire:RSIGnal?
```

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

Return Format

```
<ref_signal_mode><NL>

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

See Also

- ":TIMebase:REFClock" on page 1044
- ":TRIGger:PXI Commands" on page 1116

:ACQuire:SEGMented:ANALyze

 (see page 1354)

Command Syntax `:ACQuire:SEGMented:ANALyze`

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

This command calculates measurement statistics and/or infinite persistence over all segments that have been acquired. It corresponds to the front panel **Analyze Segments** softkey which appears in both the Measurement Statistics and Segmented Memory Menus.

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

See Also
- ":ACQuire:MODE" on page 248
- ":ACQuire:SEGMented:COUNt" on page 252
- "Introduction to :ACQuire Commands" on page 242

:ACQuire:SEGMented:COUNt

(see page 1354)

Command Syntax

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

<count> ::= an integer from 2 to 1000 (w/4M memory) in NR1 format
```

NOTE This command is available when the segmented memory option (SGM license) is installed.

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

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

The maximum number of segments may be limited by the memory depth of your oscilloscope. For example, an oscilloscope with 1M memory allows a maximum of 250 segments.

Query Syntax

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

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

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 1000 (w/4M memory) in NR1 format
```

See Also

- ":ACQuire:MODE" on page 248
- ":DIGitize" on page 211
- ":SINGle" on page 235
- ":RUN" on page 233
- ":WAVeform:SEGMented:COUNt" on page 1177
- ":ACQuire:SEGMented:ANALyze" on page 251
- "Introduction to :ACQuire Commands" on page 242

Example Code

- "Example Code" on page 253

:ACQuire:SEGMented:INDex

N (see page 1354)

Command Syntax

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

<index> ::= an integer from 1 to 1000 (w/4M memory) in NR1 format
```

NOTE This command is available when the segmented memory option (SGM license) is installed.

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

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

The maximum number of segments may be limited by the memory depth of your oscilloscope. For example, an oscilloscope with 1M memory allows a maximum of 250 segments.

Query Syntax

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

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

Return Format

```
<index><NL>

<index> ::= an integer from 1 to 1000 (w/4M memory) in NR1 format
```

See Also

- ":ACQuire:MODE" on page 248
- ":ACQuire:SEGMented:COUNt" on page 252
- ":DIGitize" on page 211
- ":SINGle" on page 235
- ":RUN" on page 233
- ":WAVeform:SEGMented:COUNt" on page 1177
- ":WAVeform:SEGMented:TTAG" on page 1178
- ":ACQuire:SEGMented:ANALyze" on page 251
- "Introduction to :ACQuire Commands" on page 242

Example Code

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

```
Option Explicit

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

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::10.112.94.136::hislip9-0.0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

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

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

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

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

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

  Dim lngSegments As Long
  lngSegments = varQueryResult

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

```
  Dim lngI As Long

  For lngI = lngSegments To 1 Step -1

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

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

  Next lngI

  Exit Sub

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

End Sub
```

:ACQuire:SRATe

(see page 1354)

Query Syntax `:ACQuire:SRATe? [MAXimum]`

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

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

Return Format `<sample_rate><NL>`

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

See Also

- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:POINts" on page 249

:ACQuire:TYPE

 (see page 1354)

Command Syntax

```
:ACQuire:TYPE <type>

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

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

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

  The AVERage type is not available when in segmented memory mode (:ACQuire:MODE SEGMented).
- HRESolution – sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

  For example, if the digitizer samples at 200 MSa/s, but the effective sample rate is 1 MSa/s (because of a slower sweep speed), only 1 out of every 200 samples needs to be stored. Instead of storing one sample (and throwing others away), the 200 samples are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.
- PEAK – sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

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

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

Query Syntax

```
:ACQuire:TYPE?
```

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

Return Format

```
<acq_type><NL>

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

See Also
- "Introduction to :ACQuire Commands" on page 242
- ":ACQuire:COUNt" on page 246
- ":ACQuire:MODE" on page 248
- ":DIGitize" on page 211
- ":WAVeform:FORMat" on page 1169
- ":WAVeform:TYPE" on page 1184
- ":WAVeform:PREamble" on page 1174

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

9 :CALibrate Commands

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

Table 84 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 261)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 262)	:CALibrate:LABel? (see page 262)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 263)	:CALibrate:OUTPut? (see page 264)	<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | WGEN2 | TSOurce} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.
n/a	:CALibrate:PROTected? (see page 265)	{"PROTected" | "UNPRotected"}
:CALibrate:STARt (see page 266)	n/a	n/a
n/a	:CALibrate:STATus? (see page 267)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string

Table 84 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPerature? (see page 268)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 269)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Introduction to :CALibrate Commands

The CALibrate subsystem provides utility commands for:

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

:CALibrate:DATE

 (see page 1354)

Query Syntax `:CALibrate:DATE?`

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

Return Format `<date><NL>`

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:LABel

(see page 1354)

Command Syntax

```
:CALibrate:LABel <string>

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

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

Query Syntax

```
:CALibrate:LABel?
```

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

Return Format

```
<string><NL>

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:OUTPut

N (see page 1354)

Command Syntax

```
:CALibrate:OUTPut <signal>

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

Note: WAVE and WGEN1 are equivalent.
```

The CALibrate:OUTPut command sets the signal that is available on the Aux Out MMCX connector:

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

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

- NFC – This option is available in the Near Field Communication (NFC) trigger mode when the ATRigger (Arm & Trigger) trigger event is selected (see ":TRIGger:NFC:TEVent" on page 1101). The ATRigger trigger event lets you arm the oscilloscope on one event and then trigger on a second event or after a specified timeout period if the second event does not occur.

  When :CALibrate:OUTPut is NFC and the specified event arms, the Aux Out goes high. The oscilloscope waits until a second event is found or until the specified timeout period expires and then triggers.

  - For NFC-A, the second event is SDD_REQ.
  - For NFC-B, the second event is ATTRIB.
  - For NFC-F, the second event is ATR_REQ.

  When the oscilloscope triggers, the Aux Out line goes low.

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

Query Syntax

```
:CALibrate:OUTPut?
```

The :CALibrate:OUTPut query returns the current source of the Aux Out signal.

Return Format

```
<signal><NL>

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

See Also

- "Introduction to :CALibrate Commands" on page 260
- ":WGEN<w>:FUNCtion" on page 1208

:CALibrate:PROTected

(see page 1354)

Query Syntax

```
:CALibrate:PROTected?
```

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

Return Format

```
<switch><NL>

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:STARt

(see page 1354)

Command Syntax `:CALibrate:STARt`

The CALibrate:STARt command starts the user calibration procedure.

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

See Also
- "Introduction to :CALibrate Commands" on page 260
- ":CALibrate:PROTected" on page 265

:CALibrate:STATus

(see page 1354)

Query Syntax

```
:CALibrate:STATus?
```

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

Return Format

```
<return value><NL>

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

<status_code> ::= an integer status code

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

The status codes and strings can be:

Status Code	Status String
+0	Calibrated
-1	Not Calibrated

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:TEMPerature

(see page 1354)

Query Syntax `:CALibrate:TEMPerature?`

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

Return Format `<return value><NL>`

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

See Also

- "Introduction to :CALibrate Commands" on page 260

:CALibrate:TIME

 (see page 1354)

Query Syntax `:CALibrate:TIME?`

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

Return Format `<date><NL>`

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

See Also

- "Introduction to :CALibrate Commands" on page 260

10 :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "Introduction to :CHANnel<n> Commands" on page 273.

Table 85 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 275)	:CHANnel<n>:BWLimit? (see page 275)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 276)	:CHANnel<n>:COUPling? (see page 276)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 277)	:CHANnel<n>:DISPlay? (see page 277)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 278)	:CHANnel<n>:IMPedance? (see page 278)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 279)	:CHANnel<n>:INVert? (see page 279)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 280)	:CHANnel<n>:LABel? (see page 280)	<string> ::= any series of 32 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 281)	:CHANnel<n>:OFFSet? (see page 281)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format

Table 85 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe <attenuation> (see page 282)	:CHANnel<n>:PROBe? (see page 282)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see page 283)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 283)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | DSMA | DSMA6 | NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 284)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:MMODel <value> (see page 285)	:CHANnel<n>:PROBe:MMODel? (see page 285)	<value> ::= {P5205 | P5210 | P6205 | P6241 | P6243 | P6245 | P6246 | P6247 | P6248 | P6249 | P6250 | P6251 | P670X | P671X | TCP202} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:RSENse <value> (see page 286)	:CHANnel<n>:PROBe:RSENse? (see page 286)	<value> ::= Ohms in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 287)	:CHANnel<n>:PROBe:SKEW? (see page 287)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 288)	:CHANnel<n>:PROBe:STYPe? (see page 288)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:ZOOM {{0 | OFF} | {1 | ON}} (see page 289)	:CHANnel<n>:PROBe:ZOOM? (see page 289)	<setting> ::= {0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 290)	:CHANnel<n>:PROTection? (see page 290)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format

Table 85 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:RANGe <range>[suffix] (see page 291)	:CHANnel<n>:RANGe? (see page 291)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 292)	:CHANnel<n>:SCALe? (see page 292)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITs <units> (see page 293)	:CHANnel<n>:UNITs? (see page 293)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 294)	:CHANnel<n>:VERNier? (see page 294)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Introduction to :CHANnel<n> Commands

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

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

A label command provides identifying annotations of up to 10 characters.

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

NOTE The obsolete CHANnel subsystem is supported.

Reporting the Setup

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

Return Format

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

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

:CHANnel<n>:BWLimit

(see page 1354)

Command Syntax

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

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

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

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

Query Syntax

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

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

Return Format

```
<bwlimit><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:COUPling

 (see page 1354)

Command Syntax

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

<coupling> ::= {AC | DC}

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

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

Query Syntax

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

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

Return Format

```
<coupling value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:DISPlay

(see page 1354)

Command Syntax

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

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

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

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

Query Syntax

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

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

Return Format

```
<display value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":VIEW" on page 239
- ":BLANk" on page 210
- ":STATus" on page 236

:CHANnel<n>:IMPedance

 (see page 1354)

Command Syntax

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

<impedance> ::= {ONEMeg | FIFTy}

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

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

Query Syntax

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

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

Return Format

```
<impedance value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:INVert

(see page 1354)

Command Syntax

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

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

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

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

Query Syntax

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

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

Return Format

```
<invert value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:LABel

N (see page 1354)

Command Syntax

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

<string> ::= quoted ASCII string

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

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

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

Query Syntax

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

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

Return Format

```
<string><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":DISPlay:LABel" on page 326
- ":DISPlay:LABList" on page 327

Example Code

```
' LABEL - This command allows you to write a name (32 characters
' maximum) next to the channel number.  It is not necessary, but
' can be useful for organizing the display.
myScope.WriteString ":CHANnel1:LABel ""CAL 1"""  ' Label ch1 "CAL 1".
myScope.WriteString ":CHANnel2:LABel ""CAL2"""   ' Label ch1 "CAL2".
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:CHANnel<n>:OFFSet

(see page 1354)

Command Syntax

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

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

<suffix> ::= {V | mV}

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

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

Query Syntax

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

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

Return Format

```
<offset><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:RANGe" on page 291
- ":CHANnel<n>:SCALe" on page 292
- ":CHANnel<n>:PROBe" on page 282

:CHANnel<n>:PROBe

(see page 1354)

Command Syntax

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

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

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

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

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

The probe attenuation factor may be from 0.001 to 10000.

This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

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

Query Syntax

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

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

Return Format

```
<attenuation><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:RANGe" on page 291
- ":CHANnel<n>:SCALe" on page 292
- ":CHANnel<n>:OFFSet" on page 281

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

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

(see page 1354)

Command Syntax

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

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

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

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

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

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

Query Syntax

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

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

Return Format

```
<head_param><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:PROBe" on page 282
- ":CHANnel<n>:PROBe:ID" on page 284
- ":CHANnel<n>:PROBe:SKEW" on page 287
- ":CHANnel<n>:PROBe:STYPe" on page 288

:CHANnel<n>:PROBe:ID

 (see page 1354)

Query Syntax

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

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

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

Return Format

```
<probe id><NL>

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

Some of the possible returned values are:

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:PROBe:MMODel

(see page 1354)

Command Syntax

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

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

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

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

Query Syntax

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

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

Return Format

```
<value><NL>

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

See Also

- ":CHANnel<n>:PROBe:ID" on page 284

:CHANnel<n>:PROBe:RSENse

N (see page 1354)

Command Syntax

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

<value> ::= Ohms in NR3 format

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

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

Query Syntax

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

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

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

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

Return Format

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

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

See Also

- ":CHANnel<n>:PROBe:ZOOM" on page 289
- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468

:CHANnel<n>:PROBe:SKEW

(see page 1354)

Command Syntax

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

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

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

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

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

Query Syntax

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

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

Return Format

```
<skew value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:PROBe:STYPe

 (see page 1354)

Command Syntax

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

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

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

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

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

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

Query Syntax

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

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

Return Format

```
<signal type><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:OFFSet" on page 281

:CHANnel<n>:PROBe:ZOOM

N (see page 1354)

Command Syntax

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

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

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

Query Syntax

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

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

Return Format

```
<setting><NL>

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

See Also

- ":CHANnel<n>:PROBe:RSENse" on page 286
- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468

:CHANnel<n>:PROTection

(see page 1354)

Command Syntax

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

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

When the analog channel input impedance is set to 50Ω, the input channels are protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the channel is automatically changed to 1 MΩ.

The :CHANnel<n>:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the channel to be used again in 50Ω mode after the signal that caused the overload has been removed from the channel input.

Reset the analog channel input impedance to 50Ω (see ":CHANnel<n>:IMPedance" on page 278) after clearing the overvoltage protection.

Query Syntax

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

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

Return Format

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:COUPling" on page 276
- ":CHANnel<n>:IMPedance" on page 278
- ":CHANnel<n>:PROBe" on page 282

:CHANnel<n>:RANGe

(see page 1354)

Command Syntax

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

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

<suffix> ::= {V | mV}

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

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, legal values for the range are from 8 mV to 40 V.

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

Query Syntax

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

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

Return Format

```
<range_argument><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:SCALe" on page 292
- ":CHANnel<n>:PROBe" on page 282

Example Code

```
' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
' range value is 8 times the volts per division.
myScope.WriteString ":CHANnel1:RANGe 8"   ' Set the vertical range to
8 volts.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:CHANnel<n>:SCALe

 (see page 1354)

Command Syntax

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

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

<suffix> ::= {V | mV}

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

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel.

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

Query Syntax

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

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

Return Format

```
<scale value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:RANGe" on page 291
- ":CHANnel<n>:PROBe" on page 282

:CHANnel<n>:UNITs

N (see page 1354)

Command Syntax

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

<units> ::= {VOLT | AMPere}

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

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

Query Syntax

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

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

Return Format

```
<units><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273
- ":CHANnel<n>:RANGe" on page 291
- ":CHANnel<n>:PROBe" on page 282
- ":EXTernal:UNITs" on page 343

:CHANnel<n>:VERNier

(see page 1354)

Command Syntax

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

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

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

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

Query Syntax

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

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

Return Format

```
<vernier value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

11 :COUNter Commands

These commands control the counter feature. See "Introduction to :COUNter Commands" on page 296.

Table 86 :COUNter Commands Summary

Command	Query	Options and Query Returns
n/a	:COUNter:CURRent? (see page 297)	<value> ::= current counter value in NR3 format
:COUNter:ENABle {{0 | OFF} | {1 | ON}} (see page 298)	:COUNter:ENABle? (see page 298)	{0 | 1}
:COUNter:MODE <mode> (see page 299)	:COUNter:MODE (see page 299)	<mode> ::= {FREQuency | PERiod | TOTalize}
:COUNter:NDIGits <value> (see page 300)	:COUNter:NDIGits (see page 300)	<value> ::= 3 to 8 in NR1 format
:COUNter:SOURce <source> (see page 301)	:COUNter:SOURce? (see page 301)	<source> ::= {CHANnel<n> | TQEVent} <n> ::= 1 to (# analog channels) in NR1 format
:COUNter:TOTalize:CLEar (see page 302)	n/a	n/a
:COUNter:TOTalize:GATE:ENABle {{0 | OFF} | {1 | ON}} (see page 303)	:COUNter:TOTalize:GATE:ENABle? (see page 303)	{0 | 1}
:COUNter:TOTalize:GATE:POLarity <polarity> (see page 304)	:COUNter:TOTalize:GATE:POLarity? (see page 304)	<polarity> ::= {{NEGative | FALLing} | {POSitive | RISing}}

Table 86 :COUNter Commands Summary (continued)

Command	Query	Options and Query Returns
:COUNter:TOTalize:GATE:SOURce <source> (see page 305)	:COUNter:TOTalize:GATE:SOURce? (see page 305)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:COUNter:TOTalize:SLOPe <slope> (see page 306)	:COUNter:TOTalize:SLOPe? (see page 306)	<slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}

Introduction to :COUNter Commands

The :COUNter subsystem provides commands to control the counter feature.

Reporting the Setup

Use :COUNter? to query setup information for the COUNter subsystem.

Return Format

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

```
:COUN:ENAB 0;SOUR CHAN1;MODE FREQ;NDIG 5
```

:COUNter:CURRent

 (see page 1354)

Query Syntax `:COUNter:CURRent?`

The :COUNter:CURRent? query returns the current counter value.

Return Format `<value><NL>`

`<value> ::= current counter value in NR3 format`

See Also

- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301

:COUNter:ENABle

(see page 1354)

Command Syntax `:COUNter:ENABle {{0 | OFF} | {1 | ON}}`

The :COUNter:ENABle command enables or disables the counter feature.

Query Syntax `:COUNter:ENABle?`

The :COUNter:ENABle? query returns whether the counter is enabled or disabled.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301

:COUNter:MODE

(see page 1354)

Command Syntax

```
:COUNter:MODE <mode>

<mode> ::= {FREQuency | PERiod | TOTalize}
```

The :COUNter:MODE command sets the counter mode:

- FREQuency — the cycles per second (Hz) of the signal.
- PERiod — the time periods of the signal's cycles.
- TOTalize — the count of edge events on the signal.

Query Syntax

```
:COUNter:MODE
```

The :COUNter:MODE? query returns the counter mode setting.

Return Format

```
<mode><NL>

<mode> ::= {FREQ | PER | TOT}
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:GATE:SOURce" on page 305
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:NDIGits

(see page 1354)

Command Syntax

```
:COUNter:NDIGits <value>

<value> ::= 3 to 8 in NR1 format
```

The :COUNter:NDIGits command sets the number of digits of resolution used for the frequency or period counter.

Higher resolutions require longer gate times, which cause the measurement times to be longer as well.

Query Syntax

```
:COUNter:NDIGits
```

The :COUNter:NDIGits? query returns the currently set number of digits of resolution.

Return Format

```
<value><NL>

<value> ::= 3 to 8 in NR1 format
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:GATE:SOURce" on page 305
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:SOURce

N (see page 1354)

Command Syntax

```
:COUNter:SOURce <source>

<source> ::= {CHANnel<n> | TQEVent}

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

The :COUNter:SOURce command selects the waveform source that the counter measures. You can select one of the analog input channels or the trigger qualified event signal.

With the TQEVent (trigger qualified event) source (available when the trigger mode is not EDGE), you can see how often trigger events are detected. This can be more often than when triggers actually occur, due to the oscilloscope's acquisition time or update rate capabilities. The 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 40 ns; therefore, the maximum trigger qualified event frequency that can be counted is 25 MHz.

Query Syntax

```
:COUNter:SOURce?
```

The :COUNter:SOURce? query returns the currently set counter source channel.

Return Format

```
<source><NL>
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300

:COUNter:TOTalize:CLEar

(see page 1354)

Command Syntax `:COUNter:TOTalize:CLEar`

The :COUNter:TOTalize:CLEar command zeros the edge event counter.

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:GATE:SOURce" on page 305
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:TOTalize:GATE:ENABle

(see page 1354)

Command Syntax `:COUNter:TOTalize:GATE:ENABle {{0 | OFF} | {1 | ON}}`

The :COUNter:TOTalize:GATE:ENABle command enables or disables totalizer gating.

When totalizer gating is enabled, the totalizer only counts edges when a second gating signal polarity is true. The second gating signal can be one of the remaining analog channel inputs.

Query Syntax `:COUNter:TOTalize:GATE:ENABle?`

The :COUNter:TOTalize:GATE:ENABle? query returns whether totalizer gating is enabled or disabled.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:GATE:SOURce" on page 305
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:TOTalize:GATE:POLarity

 (see page 1354)

Command Syntax

```
:COUNter:TOTalize:GATE:POLarity <polarity>

<polarity> ::= {{NEGative | FALLing} | {POSitive | RISing}}
```

The :COUNter:TOTalize:GATE:POLarity command specifies the gating signal condition under which totalizer edges are counted.

The gating signal is specified with the :COUNter:TOTalize:GATE:SOURce command.

Query Syntax

```
:COUNter:TOTalize:GATE:POLarity?
```

The :COUNter:TOTalize:GATE:POLarity? query returns the currently specified gating signal condition.

Return Format

```
<polarity><NL>

<polarity> ::= {NEG | POS}
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:SOURce" on page 305
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:TOTalize:GATE:SOURce

(see page 1354)

Command Syntax

```
:COUNter:TOTalize:GATE:SOURce <source>

<source> ::= CHANnel<n>

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

The :COUNter:TOTalize:GATE:SOURce command selects the analog channel that has the totalizer gating signal.

Query Syntax

```
:COUNter:TOTalize:GATE:SOURce?
```

The :COUNter:TOTalize:GATE:SOURce? query returns the current totalizer gating signal source.

Return Format

```
<source><NL>
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:SLOPe" on page 306

:COUNter:TOTalize:SLOPe

(see page 1354)

Command Syntax

```
:COUNter:TOTalize:SLOPe <slope>

<slope> ::= {{NEGative | FALLing} | {POSitive | RISing}}
```

The :COUNter:TOTalize:SLOPe command specifies whether positive or negative edges are counted.

Query Syntax

```
:COUNter:TOTalize:SLOPe?
```

The :COUNter:TOTalize:SLOPe? query returns the currently set slope specification.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- ":COUNter:CURRent" on page 297
- ":COUNter:ENABle" on page 298
- ":COUNter:MODE" on page 299
- ":COUNter:NDIGits" on page 300
- ":COUNter:SOURce" on page 301
- ":COUNter:TOTalize:CLEar" on page 302
- ":COUNter:TOTalize:GATE:ENABle" on page 303
- ":COUNter:TOTalize:GATE:POLarity" on page 304
- ":COUNter:TOTalize:GATE:SOURce" on page 305

12 :DEMO Commands

You can output demonstration signals on the oscilloscope's Demo 1 (Probe Comp) terminal. See "Introduction to :DEMO Commands" on page 307.

Table 87 :DEMO Commands Summary

Command	Query	Options and Query Returns
`:DEMO:FUNCtion <signal>` (see page 308)	`:DEMO:FUNCtion?` (see page 309)	`<signal> ::= {SINusoid | NOISy | RINGing | SINGle | CLK | GLITch | BURSt | RUNT | TRANsition | RFBurst | LFSine | FMBurst | NFC | CXPI | ARINc | MANChester | MIL | NMONotonic | HARMonics | COUPling | KEYSight}`
`:DEMO:OUTPut {{0 | OFF} | {1 | ON}}` (see page 310)	`:DEMO:OUTPut?` (see page 310)	`{0 | 1}`

Introduction to :DEMO Commands

The :DEMO subsystem provides commands to output demonstration signals on the oscilloscope's Demo 1 (Probe Comp) terminal.

Reporting the Setup

Use :DEMO? to query setup information for the DEMO subsystem.

Return Format

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

```
:DEMO:FUNC SIN;OUTP 0
```

:DEMO:FUNCtion

N (see page 1354)

Command Syntax

```
:DEMO:FUNCtion <signal>

<signal> ::= {SINusoid | NOISy | RINGing | SINGle | CLK | GLITch
   | BURSt | RUNT | TRANsition | RFBurst | LFSine | FMBurst | NFC
   | CXPI | ARINc | MANChester | MIL | NMONotonic | HARMonics
   | COUPling | KEYSight}
```

The :DEMO:FUNCtion command selects the type of demo signal:

Demo Signal Function	Demo 1 Terminal
SINusoid	5 MHz sine wave @ ~ 6 Vpp, 0 V offset
NOISy	1 kHz sine wave @ ~ 2.4 Vpp, 0.0 V offset, with ~ 0.5 Vpp of random noise added
RINGing	500 kHz digital pulse @ ~ 3 Vpp, 1.5 V offset, and ~500 ns pulse width with ringing
SINGle	~500 ns wide digital pulse with ringing @ ~ 3 Vpp, 1.5 V offset Press the front panel **Set Off Single-Shot** softkey to cause the selected single-shot signal to be output.
CLK	3.6 MHz clock @ ~2 Vpp, 1 V offset, with infrequent glitch (1 glitch per 1,000,000 clocks)
GLITch	Burst of 6 digital pulses (plus infrequent glitch) that occurs once every 80 µs @ ~3.6 Vpp, ~1.8 V offset
BURSt	Burst of digital pulses that occur every 50 µs @ ~ 3.6 Vpp, ~1.5 V offset
RUNT	Digital pulse train with positive and negative runt pulses @ ~ 2.9 Vpp, 1.5 V offset
TRANsition	Digital pulse train with two different edge speeds @ ~ 3.5 Vpp, 1.75 V offset
RFBurst	5-cycle burst of a 10 MHz amplitude modulated sine wave @ ~ 2.6 Vpp, 0 V offset occurring once every 4 ms
LFSine	30 Hz sine wave @ ~2.7 Vpp, 0 V offset, with very narrow glitch near each positive peak
FMBurst	FM burst, modulated from ~100 kHz to ~1 MHz, ~5.0 Vpp, ~600 mV offset.
NFC	~2.4 Vpp unmodulated carrier amplitude, ~1.8 Vpp maximum modulated amplitude, ~0.0 V offset
CXPI	CXPI, ~2.5 Vpp, ~1.25 V offset, frame IDs: 20, 21, and 29 (with a frame error).
ARINc	ARINC 429, 100 kbps, ~5 Vpp, ~0 V offset.

Demo Signal Function	Demo 1 Terminal
MANChester	Manchester/NRZ @ 125 kbps, ~2.5 Vpp, 1.25 V offset, a 125 kb/s 10-bit PSI5-like signal
MIL	MIL-STD-1553 RT to RT transfer, received ~1.3 Vpp, transmitted ~4.8 Vpp, 0 V offset
NMONotonic	Digital pulse train with infrequent non-monotonic rising edges @ ~ 2.85 Vpp, 1.42 V offset
HARMonics	1 kHz sine wave @ ~3.5 Vpp, 0.0 V offset, with a ~2 kHz sine wave coupled in
COUPling	1 kHz square wave @ ~1 Vpp, 0.0 V offset, with a ~90 kHz sine wave with ~180 mVpp riding on top
KEYSight	Series of positive and negative pulses, 125 µs wide, amplitude modulated, ~2.6 Vpp, ~0 V offset

Query Syntax

```
:DEMO:FUNCtion?
```

The :DEMO:FUNCtion? query returns the currently selected demo signal type.

Return Format

```
<signal><NL>

<signal> ::= {SIN | NOIS | RING | SING | CLK | GLIT | BURS | RUNT
   | TRAN | RFB | LFS | FMB | NFC | CXPI | ARIN | MANC | MIL | NMON
   | HARM | COUP | KEYS}
```

See Also

- "Introduction to :DEMO Commands" on page 307

:DEMO:OUTPut

 (see page 1354)

Command Syntax

```
:DEMO:OUTPut <on_off>

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

The :DEMO:OUTPut command specifies whether the demo signal output is ON (1) or OFF (0).

Query Syntax

```
:DEMO:OUTPut?
```

The :DEMO:OUTPut? query returns the current state of the demo signal output setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :DEMO Commands" on page 307
- ":DEMO:FUNCtion" on page 308

13 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 313.

Table 88 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation<n> {{0 | OFF} | {1 | ON}} (see page 314)	:DISPlay:ANNotation<n>? (see page 314)	{0 | 1} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:BACKground <mode> (see page 315)	:DISPlay:ANNotation<n>:BACKground? (see page 315)	<mode> ::= {OPAQue | INVerted} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:COLor <color> (see page 316)	:DISPlay:ANNotation<n>:COLor? (see page 316)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED} <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:TEXT <string> (see page 317)	:DISPlay:ANNotation<n>:TEXT? (see page 317)	<string> ::= quoted ASCII string (up to 254 characters) <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:X1Position <value> (see page 318)	:DISPlay:ANNotation<n>:X1Position? (see page 318)	<value> ::= an integer from 0 to (800 - width of annotation) in NR1 format. <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:ANNotation<n>:Y1Position <value> (see page 319)	:DISPlay:ANNotation<n>:Y1Position? (see page 319)	<value> ::= an integer from 0 to (480 - height of annotation) in NR1 format. <n> ::= an integer from 1 to 4 in NR1 format.
:DISPlay:CLEar (see page 320)	n/a	n/a

Table 88 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:DISPlay:DATA? [<format>][,][<palette>] (see page 321)	<format> ::= {BMP | BMP8bit | PNG} <palette> ::= {COLor | GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}} (see page 322)	:DISPlay:GRATicule:ALABels? (see page 322)	<setting> ::= {0 | 1}
:DISPlay:GRATicule:INTensity <value> (see page 323)	:DISPlay:GRATicule:INTensity? (see page 323)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:GRATicule:TYPE <type> (see page 324)	:DISPlay:GRATicule:TYPE? (see page 324)	<type> ::= {FULL | MVOLt | IRE}
:DISPlay:INTensity:WAVeform <value> (see page 325)	:DISPlay:INTensity:WAVeform? (see page 325)	<value> ::= an integer from 0 to 100 in NR1 format.
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 326)	:DISPlay:LABel? (see page 326)	{0 | 1}
:DISPlay:LABList <binary block> (see page 327)	:DISPlay:LABList? (see page 327)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:MENU <menu> (see page 328)	n/a	<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}
:DISPlay:MESSage:CLEar (see page 329)	n/a	n/a
:DISPlay:PERSistence <value> (see page 330)	:DISPlay:PERSistence? (see page 330)	<value> ::= {MINimum | INFinite | <time> | ADAPtive} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:SIDebar <sidebar> (see page 331)	n/a	<sidebar> ::= {SUMMary | CURSors | MEASurements | DVM | NAVigate | CONTrols | EVENts | COUNter}
:DISPlay:VECTors {1 | ON} (see page 332)	:DISPlay:VECTors? (see page 332)	1

Introduction to :DISPlay Commands

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

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

Reporting the Setup

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

Return Format

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

```
:DISP:LAB 0;VECT 1;PERS MIN
```

:DISPlay:ANNotation<n>

N (see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n> <setting>

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

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n> command turns the annotation on and off. When on, the annotation appears in the upper left corner of the oscilloscope's display.

The annotation is useful for documentation purposes, to add notes before capturing screens.

Query Syntax

```
:DISPlay:ANNotation<n>?
```

The :DISPlay:ANNotation<n>? query returns the annotation setting.

Return Format

```
<value><NL>

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

See Also

- ":DISPlay:ANNotation<n>:TEXT" on page 317
- ":DISPlay:ANNotation<n>:COLor" on page 316
- ":DISPlay:ANNotation<n>:BACKground" on page 315
- ":DISPlay:ANNotation<n>:X1Position" on page 318
- ":DISPlay:ANNotation<n>:Y1Position" on page 319
- "Introduction to :DISPlay Commands" on page 313

:DISPlay:ANNotation<n>:BACKground

(see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n>:BACKground <mode>

<mode> ::= {OPAQue | INVerted}

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n>:BACKground command specifies the background of the annotation:

- OPAQue – the annotation has a solid background.
- INVerted – the annotation's foreground and background colors are switched.

Query Syntax

```
:DISPlay:ANNotation<n>:BACKground?
```

The :DISPlay:ANNotation<n>:BACKground? query returns the specified annotation background mode.

Return Format

```
<mode><NL>

<mode> ::= {OPAQ | INV}
```

See Also

- ":DISPlay:ANNotation<n>" on page 314
- ":DISPlay:ANNotation<n>:TEXT" on page 317
- ":DISPlay:ANNotation<n>:COLor" on page 316
- ":DISPlay:ANNotation<n>:X1Position" on page 318
- ":DISPlay:ANNotation<n>:Y1Position" on page 319
- "Introduction to :DISPlay Commands" on page 313

:DISPlay:ANNotation<n>:COLor

(see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n>:COLor <color>

<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe
             | RED}

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n>:COLor command specifies the annotation color. You can choose white, red, or colors that match analog channels, digital channels, math waveforms, reference waveforms, or markers.

Query Syntax

```
:DISPlay:ANNotation<n>:COLor?
```

The :DISPlay:ANNotation<n>:COLor? query returns the specified annotation color.

Return Format

```
<color><NL>

<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARK | WHIT
             | RED}
```

See Also

- ":DISPlay:ANNotation<n>" on page 314
- ":DISPlay:ANNotation<n>:TEXT" on page 317
- ":DISPlay:ANNotation<n>:BACKground" on page 315
- ":DISPlay:ANNotation<n>:X1Position" on page 318
- ":DISPlay:ANNotation<n>:Y1Position" on page 319
- "Introduction to :DISPlay Commands" on page 313

:DISPlay:ANNotation<n>:TEXT

N (see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n>:TEXT <string>

<string> ::= quoted ASCII string (up to 254 characters)

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n>:TEXT command specifies the annotation string. The annotation string can contain as many characters as will fit in the Edit Annotation box on the oscilloscope's screen, up to 254 characters.

You can include a carriage return in the annotation string using the characters "\n". Note that this is not a new line character but the actual "\" (backslash) and "n" characters in the string. Carriage returns lessen the number of characters available for the annotation string.

Use :DISPlay:ANNotation<n>:TEXT "" to remotely clear the annotation text. (Two sets of quote marks without a space between them creates a NULL string.)

Query Syntax

```
:DISPlay:ANNotation<n>:TEXT?
```

The :DISPlay:ANNotation<n>:TEXT? query returns the specified annotation text.

When carriage returns are present in the annotation text, they are returned as the actual carriage return character (ASCII 0x0D).

Return Format

```
<string><NL>

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

See Also

- ":DISPlay:ANNotation<n>" on page 314
- ":DISPlay:ANNotation<n>:COLor" on page 316
- ":DISPlay:ANNotation<n>:BACKground" on page 315
- ":DISPlay:ANNotation<n>:X1Position" on page 318
- ":DISPlay:ANNotation<n>:Y1Position" on page 319
- "Introduction to :DISPlay Commands" on page 313

:DISPlay:ANNotation<n>:X1Position

(see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n>:X1Position <value>

<value> ::= an integer from 0 to (800 - width of annotation) in NR1 form
at.

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n>:X1Position command sets the annotation's horizontal X1 position.

Query Syntax

```
:DISPlay:ANNotation<n>:X1Position?
```

The :DISPlay:ANNotation<n>:X1Position? query returns the annotation's horizontal X1 position.

Return Format

```
<value><NL>

<value> ::= an integer from 0 to (800 - width of annotation) in NR1 form
at.
```

See Also

- ":DISPlay:ANNotation<n>:Y1Position" on page 319
- ":DISPlay:ANNotation<n>" on page 314
- ":DISPlay:ANNotation<n>:COLor" on page 316
- ":DISPlay:ANNotation<n>:BACKground" on page 315
- ":DISPlay:ANNotation<n>:TEXT" on page 317

:DISPlay:ANNotation<n>:Y1Position

(see page 1354)

Command Syntax

```
:DISPlay:ANNotation<n>:Y1Position <value>

<value> ::= an integer from 0 to (480 - height of annotation) in NR1 for
mat.

<n> ::= an integer from 1 to 10 in NR1 format.
```

The :DISPlay:ANNotation<n>:Y1Position command sets the annotation's vertical Y1 position.

Query Syntax

```
:DISPlay:ANNotation<n>:Y1Position?
```

The :DISPlay:ANNotation<n>:Y1Position? query returns the annotation's vertical Y1 position.

Return Format

```
<value><NL>

<value> ::= an integer from 0 to (480 - height of annotation) in NR1 for
mat.
```

See Also

- ":DISPlay:ANNotation<n>:X1Position" on page 318
- ":DISPlay:ANNotation<n>" on page 314
- ":DISPlay:ANNotation<n>:COLor" on page 316
- ":DISPlay:ANNotation<n>:BACKground" on page 315
- ":DISPlay:ANNotation<n>:TEXT" on page 317

:DISPlay:CLEar

 (see page 1354)

Command Syntax `:DISPlay:CLEar`

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

See Also

- "Introduction to :DISPlay Commands" on page 313

:DISPlay:DATA

N (see page 1354)

Query Syntax

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

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

<palette> ::= COLor
```

The :DISPlay:DATA? query reads screen image data. You can choose 24-bit BMP, 8-bit BMP8bit, or 24-bit PNG formats in color.

If no format or palette option is specified, the screen image is returned in whatever image format is selected by the front panel's **Main Menu > File > Save Menu > Format** softkey. If the **Format** softkey does not select an image format (in other words, it selects a setup or data format), the BMP, COLor format is used.

Screen image data is returned in the IEEE-488.2 # binary block data format.

Return Format

```
<display data><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 313
- ":HCOPY:SDUMp:DATA" on page 404
- ":HCOPY:SDUMp:FORMat" on page 405
- "*RCL (Recall)" on page 188
- "*SAV (Save)" on page 192
- ":VIEW" on page 239

Example Code

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

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:DISPlay:GRATicule:ALABels

 (see page 1354)

Command Syntax `:DISPlay:GRATicule:ALABels {{0 | OFF} | {1 | ON}}`

The :DISPlay:GRATicule:ALABels command turns graticule (grid) axis labels on or off.

Query Syntax `:DISPlay:GRATicule:ALABels?`

The :DISPlay:GRATicule:ALABels? query returns the graticule (grid) axis labels setting

Return Format `<setting><NL>`

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

See Also

- ":DISPlay:GRATicule:INTensity" on page 323
- ":DISPlay:GRATicule:TYPE" on page 324

:DISPlay:GRATicule:INTensity

(see page 1354)

Command Syntax

```
:DISPlay:GRATicule:INTensity <value>

<value> ::= an integer from 0 to 100 in NR1 format.
```

The :DISPlay:GRATicule:INTensity command sets the graticule (grid) intensity.

Query Syntax

```
:DISPlay:GRATicule:INTensity?
```

The :DISPlay:GRATicule:INTensity? query returns the graticule (grid) intensity setting.

Return Format

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

See Also

- ":DISPlay:GRATicule:ALABels" on page 322
- ":DISPlay:GRATicule:TYPE" on page 324

:DISPlay:GRATicule:TYPE

N (see page 1354)

Command Syntax

```
:DISPlay:GRATicule:TYPE <type>

<type> ::= {FULL | MVOLt | IRE}
```

The :DISPlay:GRATicule:TYPE command sets the graticule (grid) type.

When the TV trigger type is selected (see ":TRIGger:MODE" on page 1066), and the vertical scaling of at least one displayed channel is 140 mV/div, the :DISPlay:GRATicule:TYPE command lets you select from these grid types:

- FULL – the normal oscilloscope grid.
- MVOLt – shows vertical grids, labeled on the left, from -0.3 V to 0.8 V.
- IRE – (Institute of Radio Engineers) shows vertical grids in IRE units, labeled on the left, from -40 to 100 IRE. The 0.35 V and 0.7 V levels from the MVOLt grid are also shown and labeled at the right. When the IRE grid is selected, cursor values on the display are also shown in IRE units. However, cursor values via the remote interface are not in IRE units.

The MVOLt and IRE grid values are accurate when the vertical scaling is 140 mV/division and the vertical offset is 245 mV.

Query Syntax

```
:DISPlay:GRATicule:TYPE?
```

The :DISPlay:GRATicule:TYPE? query returns the graticule (grid) type setting.

Return Format

```
<type><NL>

<type> ::= {FULL | MVOL | IRE}
```

See Also

- ":TRIGger:MODE" on page 1066
- ":CHANnel<n>:SCALe" on page 292
- ":CHANnel<n>:OFFSet" on page 281
- ":DISPlay:GRATicule:ALABels" on page 322
- ":DISPlay:GRATicule:INTensity" on page 323

:DISPlay:INTensity:WAVeform

N (see page 1354)

Command Syntax

```
:DISPlay:INTensity:WAVeform <value>
<value> ::= an integer from 0 to 100 in NR1 format.
```

The :DISPlay:INTensity:WAVeform command sets the waveform intensity.

This is the same as adjusting the front panel **[Intensity]** knob.

Query Syntax

```
:DISPlay:INTensity:WAVeform?
```

The :DISPlay:INTensity:WAVeform? query returns the waveform intensity setting.

Return Format

```
<value><NL>
<value> ::= an integer from 0 to 100 in NR1 format.
```

See Also

- "Introduction to :DISPlay Commands" on page 313

:DISPlay:LABel

(see page 1354)

Command Syntax

```
:DISPlay:LABel <value>

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

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

Query Syntax

```
:DISPlay:LABel?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 313
- ":CHANnel<n>:LABel" on page 280

Example Code

```
' DISP_LABEL
'  - Turns label names ON or OFF on the analyzer display.
myScope.WriteString ":DISPlay:LABel ON"    ' Turn on labels.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:DISPlay:LABList

N (see page 1354)

Command Syntax

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

<binary block> ::= an ordered list of up to 75 labels, a maximum of 32
                   characters each, separated by newline characters.
```

The :DISPlay:LABList command adds labels to the label list. Labels are added in alphabetical order.

NOTE

Labels that begin with the same alphabetic base string followed by decimal digits are considered duplicate labels. Duplicate labels are not added to the label list. For example, if label "A0" is in the list and you try to add a new label called "A12345678901234567890123456789012345678901", the new label is not added.

Query Syntax

```
:DISPlay:LABList?
```

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

Return Format

```
<binary block><NL>

<binary block> ::= an ordered list of up to 75 labels, a maximum of 32
                   characters each, separated by newline characters.
```

See Also

- "Introduction to :DISPlay Commands" on page 313
- ":DISPlay:LABel" on page 326
- ":CHANnel<n>:LABel" on page 280

:DISPlay:MENU

 (see page 1354)

Command Syntax

```
:DISPlay:MENU <menu>

<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}
```

The :DISPlay:MENU command changes the front panel softkey menu.

:DISPlay:MESSage:CLEar

 (see page 1354)

Command Syntax `:DISPlay:MESSage:CLEar`

The :DISPlay:MESSage:CLEar command removes all user messages that are currently on screen.

See Also

- ":SYSTem:DSP" on page 1015

:DISPlay:PERSistence

 (see page 1354)

Command Syntax

```
:DISPlay:PERSistence <value>

<value> ::= {MINimum | INFinite | <time> | ADAPtive}

<time> ::= seconds in in NR3 format from 100E-3 to 60E0
```

The :DISPlay:PERSistence command specifies the persistence setting:

- MINimum – indicates zero persistence.
- INFinite – indicates infinite persistence.
- <time> – for variable persistence, that is, you can specify how long acquisitions remain on the screen.
- ADAPtive – all previous acquisitions that have taken place in hardware since the last controller PC screen capture are displayed.

Use the :DISPlay:CLEar command to erase points stored by persistence.

Query Syntax

```
:DISPlay:PERSistence?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 313
- ":DISPlay:CLEar" on page 320

:DISPlay:SIDebar

 (see page 1354)

Command Syntax

```
:DISPlay:SIDebar <sidebar>

<sidebar> ::= {SUMMary | CURSors | MEASurements | DVM | NAVigate
              | CONTrols | EVENts | COUNter}
```

The :DISPlay:SIDebar command specifies the sidebar dialog to display on the screen.

:DISPlay:VECTors

N (see page 1354)

Command Syntax

```
:DISPlay:VECTors <vectors>

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

Vector display is always ON in the M9241/42/43A PXIe oscilloscopes.

When vectors are turned on, the oscilloscope displays lines connecting sampled data points.

Query Syntax

```
:DISPlay:VECTors?
```

The :DISPlay:VECTors? query returns the vectors setting.

Return Format

```
<vectors><NL>

<vectors> ::= 1
```

See Also

- "Introduction to :DISPlay Commands" on page 313

14 :DVM Commands

These commands control the digital voltmeter (DVM) feature.

Table 89 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARANge {{0 | OFF} | {1 | ON}} (see page 334)	:DVM:ARANge? (see page 334)	{0 | 1}
n/a	:DVM:CURRent? (see page 335)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABle {{0 | OFF} | {1 | ON}} (see page 336)	:DVM:ENABle? (see page 336)	{0 | 1}
:DVM:MODE <mode> (see page 337)	:DVM:MODE? (see page 337)	<dvm_mode> ::= {ACRMs | DC | DCRMs}
:DVM:SOURce <source> (see page 338)	:DVM:SOURce? (see page 338)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

:DVM:ARANge

 (see page 1354)

Command Syntax

```
:DVM:ARANge <setting>

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

If the selected digital voltmeter (DVM) source channel is not used in oscilloscope triggering, the :DVM:ARANge command turns the digital voltmeter's Auto Range capability on or off.

- When on, the DVM channel's vertical scale, vertical (ground level) position, and trigger (threshold voltage) level (used for the counter frequency measurement) are automatically adjusted.

  The Auto Range capability overrides attempted adjustments of the channel's vertical scale and position.
- When off, you can adjust the channel's vertical scale and position normally.

Query Syntax

```
:DVM:ARANge?
```

The :DVM:ARANge? query returns a flag indicating whether the digital voltmeter's Auto Range capability is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 338
- ":DVM:ENABle" on page 336
- ":DVM:MODE" on page 337

:DVM:CURRent

 (see page 1354)

Query Syntax `:DVM:CURRent?`

The :DVM:CURRent? query returns the displayed 3-digit DVM value based on the current mode.

Return Format `<dvm_value><NL>`

`<dvm_value> ::= floating-point number in NR3 format`

See Also

- ":DVM:SOURce" on page 338
- ":DVM:ENABle" on page 336
- ":DVM:MODE" on page 337

:DVM:ENABle

(see page 1354)

Command Syntax

```
:DVM:ENABle <setting>

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

The :DVM:ENABle command turns the digital voltmeter (DVM) analysis feature on or off.

Query Syntax

```
:DVM:ENABle?
```

The :DVM:ENABle? query returns a flag indicating whether the digital voltmeter (DVM) analysis feature is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":DVM:SOURce" on page 338
- ":DVM:MODE" on page 337
- ":DVM:ARANge" on page 334

:DVM:MODE

(see page 1354)

Command Syntax

```
:DVM:MODE <dvm_mode>

<dvm_mode> ::= {ACRMs | DC | DCRMs}
```

The :DVM:MODE command sets the digital voltmeter (DVM) mode:

- ACRMs — displays the root-mean-square value of the acquired data, with the DC component removed.
- DC — displays the DC value of the acquired data.
- DCRMs — displays the root-mean-square value of the acquired data.

Query Syntax

```
:DVM:MODE?
```

The :DVM:MODE? query returns the selected DVM mode.

Return Format

```
<dvm_mode><NL>

<dvm_mode> ::= {ACRM | DC | DCRM}
```

See Also

- ":DVM:ENABle" on page 336
- ":DVM:SOURce" on page 338
- ":DVM:ARANge" on page 334
- ":DVM:CURRent" on page 335

:DVM:SOURce

(see page 1354)

Command Syntax

```
:DVM:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1-2 or 1-4 in NR1 format
```

The :DVM:SOURce command sets the select the analog channel on which digital voltmeter (DVM) measurements are made.

The selected channel does not have to be on (displaying a waveform) in order for DVM measurements to be made.

Query Syntax

```
:DVM:SOURce?
```

The :DVM:SOURce? query returns the selected DVM input source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}

<n> ::= 1-2 or 1-4 in NR1 format
```

See Also

- ":DVM:ENABle" on page 336
- ":DVM:MODE" on page 337
- ":DVM:ARANge" on page 334
- ":DVM:CURRent" on page 335

15 :EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXTernal Trigger Commands" on page 339.

Table 90 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 340)	:EXTernal:BWLimit? (see page 340)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 341)	:EXTernal:PROBe? (see page 341)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 342)	:EXTernal:RANGe? (see page 342)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 343)	:EXTernal:UNITs? (see page 343)	<units> ::= {VOLT | AMPere}

Introduction to :EXTernal Trigger Commands

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

Reporting the Setup

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

Return Format

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

```
:EXT:BWL 0;RANG +8.0E+00;UNIT VOLT;PROB +1.000E+00
```

:EXTernal:BWLimit

 (see page 1354)

Command Syntax

```
:EXTernal:BWLimit <bwlimit>

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

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

Query Syntax

```
:EXTernal:BWLimit?
```

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

Return Format

```
<bwlimit><NL>

<bwlimit> ::= 0
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 339
- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:HFReject" on page 1058

:EXTernal:PROBe

(see page 1354)

Command Syntax

```
:EXTernal:PROBe <attenuation>

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

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

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

Query Syntax

```
:EXTernal:PROBe?
```

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

Return Format

```
<attenuation><NL>

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

See Also

- "Introduction to :EXTernal Trigger Commands" on page 339
- ":EXTernal:RANGe" on page 342
- "Introduction to :TRIGger Commands" on page 1053
- ":CHANnel<n>:PROBe" on page 282

:EXTernal:RANGe

(see page 1354)

Command Syntax

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

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

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

The :EXTernal:RANGe command is provided for product compatibility. When using 1:1 probe attenuation, the range can only be set to 8.0 V.

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

Query Syntax

```
:EXTernal:RANGe?
```

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

Return Format

```
<range_argument><NL>

<range_argument> ::= external trigger range value in NR3 format
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 339
- ":EXTernal:PROBe" on page 341
- "Introduction to :TRIGger Commands" on page 1053

:EXTernal:UNITs

(see page 1354)

Command Syntax

```
:EXTernal:UNITs <units>

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

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

Query Syntax

```
:EXTernal:UNITs?
```

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

Return Format

```
<units><NL>

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

See Also

- "Introduction to :EXTernal Trigger Commands" on page 339
- "Introduction to :TRIGger Commands" on page 1053
- ":EXTernal:RANGe" on page 342
- ":EXTernal:PROBe" on page 341
- ":CHANnel<n>:UNITs" on page 293

16 :FRANalysis Commands

Control oscilloscope functions associated with the Frequency Response Analysis (FRA) feature, which is available in oscilloscope models that have a license-enabled built-in waveform generator. See "Introduction to :FRANalysis Commands" on page 346.

Table 91 :FRANalysis Commands Summary

Command	Query	Options and Query Returns
n/a	:FRANalysis:DATA? [SWEep | SINGle] (see page 347)	<binary_block> ::= comma-separated data with newlines at the end of each row
:FRANalysis:ENABle {{0 | OFF} | {1 | ON}} (see page 348)	:FRANalysis:ENABle? (see page 348)	{0 | 1}
:FRANalysis:FREQuency :MODE <setting> (see page 349)	:FRANalysis:FREQuency :MODE? (see page 349)	<setting> ::= {SWEep | SINGle}
:FRANalysis:FREQuency :SINGle <value>[suffix] (see page 350)	:FRANalysis:FREQuency :SINGle? (see page 350)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STARt <value>[suffix] (see page 351)	:FRANalysis:FREQuency :STARt? (see page 351)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:FREQuency :STOP <value>[suffix] (see page 352)	:FRANalysis:FREQuency :STOP? (see page 352)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:FRANalysis:PPDecade <value> (see page 353)	:FRANalysis:PPDecade? (see page 353)	<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:FRANalysis:RUN (see page 354)	n/a	n/a

Table 91 :FRANalysis Commands Summary (continued)

Command	Query	Options and Query Returns
:FRANalysis:SOURce:INPut <source> (see page 355)	:FRANalysis:SOURce:INPut? (see page 355)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:SOURce:OUTPut <source> (see page 356)	:FRANalysis:SOURce:OUTPut? (see page 356)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:FRANalysis:TRACe <selection> (see page 357)	:FRANalysis:TRACe? (see page 357)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
:FRANalysis:WGEN:LOAD <impedance> (see page 358)	:FRANalysis:WGEN:LOAD? (see page 358)	<impedance> ::= {ONEMeg | FIFTy}
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>] (see page 359)	:FRANalysis:WGEN:VOLTage? [<range>] (see page 359)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 360)	:FRANalysis:WGEN:VOLTage:PROFile? (see page 360)	{0 | 1}

Introduction to :FRANalysis Commands

The FRANalysis subsystem controls the Frequency Response Analysis feature in the oscilloscope.

The Frequency Response Analysis (FRA) feature controls the built-in waveform generator to sweep a sine wave across a range of frequencies while measuring the input to and output from a device under test (DUT). At each frequency, gain (A) and phase are measured and plotted on a frequency response chart.

Reporting the Setup

Use :FRANalysis? to query setup information for the FRANalysis subsystem.

Return Format

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

```
:FRAN:SOUR:INP CHAN1;OUTP CHAN2;:FRAN:FREQ:STAR +100E+00;
STOP +20.000000E+06;:FRAN:WGEN:VOLT +200.0E-03;LOAD FIFT
```

:FRANalysis:DATA

(see page 1354)

Query Syntax

```
:FRANalysis:DATA? [SWEep | SINGle]
```

The :FRANalysis:DATA? query returns the frequency response analysis data.

The data is returned in four comma-separated columns of data for each step in the sweep: Frequency (Hz), Amplitude (Vpp), Gain (dB), and Phase (°).

You can use the :FRANalysis:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

The SWEep or SINGle option specifies whether to get the data from a sweep or single-frequency analysis (see :FRANalysis:FREQuency:MODE). If this option is not specified, the data from the sweep analysis is returned by default.

Return Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:MODE" on page 349
- ":FRANalysis:FREQuency:SINGle" on page 350
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:TRACe" on page 357
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359

:FRANalysis:ENABle

(see page 1354)

Command Syntax

```
:FRANalysis:ENABle <setting>

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

The :FRANalysis:ENABle command turns the Frequency Response Analysis (FRA) feature on or off.

Query Syntax

```
:FRANalysis:ENABle?
```

The :FRANalysis:ENABle? query returns a flag indicating whether the Frequency Response Analysis (FRA) feature is on or off.

Return Format

```
<setting><NL>

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

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359

:FRANalysis:FREQuency:MODE

 (see page 1354)

Command Syntax

```
:FRANalysis:FREQuency:MODE <setting>

<setting> ::= {SWEep | SINGle}
```

The :FRANalysis:FREQuency:MODE command lets you select between the normal swept frequency response analysis or analysis at a single frequency, which can be useful when debugging.

Query Syntax

```
:FRANalysis:FREQuency:MODE?
```

The :FRANalysis:FREQuency:MODE? query returns the frequency mode setting.

Return Format

```
<setting><NL>

<setting> ::= {SWEep | SINGle}
```

See Also

- ":FRANalysis:RUN" on page 354
- ":FRANalysis:FREQuency:SINGle" on page 350
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360

:FRANalysis:FREQuency:SINGle

 (see page 1354)

Command Syntax

```
:FRANalysis:FREQuency:SINGle <value>[suffix]

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:SINGle command command sets the single frequency value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the minimum frequency of 20 Hz.

Query Syntax

```
:FRANalysis:FREQuency:SINGle?
```

The :FRANalysis:FREQuency:SINGle? query returns the single frequency setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}
```

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:FREQuency:MODE" on page 349
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360

:FRANalysis:FREQuency:STARt

N (see page 1354)

Command Syntax

```
:FRANalysis:FREQuency:STARt <value>[suffix]

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:STARt command command sets the frequency sweep start value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the minimum frequency of 20 Hz.

Query Syntax

```
:FRANalysis:FREQuency:STARt?
```

The :FRANalysis:FREQuency:STARt? query returns the frequency sweep start setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:FREQuency:MODE" on page 349
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360

:FRANalysis:FREQuency:STOP

(see page 1354)

Command Syntax

```
:FRANalysis:FREQuency:STOP <value>[suffix]

<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}

[suffix] ::= {Hz | kHz| MHz}
```

The :FRANalysis:FREQuency:STOP command sets the frequency sweep stop value. The frequency response analysis is displayed on a log scale Bode plot, so you can select from decade values in addition to the maximum frequency of 20 MHz.

Query Syntax

```
:FRANalysis:FREQuency:STOP?
```

The :FRANalysis:FREQuency:STOP? query returns the frequency sweep stop setting.

Return Format

```
<value><NL>

<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}
```

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:FREQuency:MODE" on page 349
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360

:FRANalysis:PPDecade

(see page 1354)

Command Syntax

```
:FRANalysis:PPDecade <value>

<value> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

The :FRANalysis:PPDecade command specifies the number of points per decade in the frequency response analysis.

Query Syntax

```
:FRANalysis:PPDecade?
```

The :FRANalysis:PPDecade? query returns the points per decade setting.

Return Format

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

See Also

- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360

:FRANalysis:RUN

(see page 1354)

Command Syntax `:FRANalysis:RUN`

The :FRANalysis:RUN command performs the Frequency Response Analysis. This analysis controls the built-in waveform generator to sweep a sine wave across a range of frequencies while measuring the input to and output from a device under test (DUT). At each frequency, gain (A) and phase are measured and plotted on a Bode frequency response chart.

The :FRANalysis:APPLy command is a valid compatible alias for the :FRANalysis:RUN command.

When the frequency response analysis completes, you can use the :FRANalysis:DATA? query to get four comma-separated columns of data for each step in the sweep: Frequency (Hz), Amplitude (Vpp), Gain (dB), and Phase (°).

You can use the :FRANalysis:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

It takes some time for the frequency sweep analysis to complete. You can query bit 0 of the Standard Event Status Register (*ESR?) to find out when the analysis is complete.

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:MODE" on page 349
- ":FRANalysis:FREQuency:SINGle" on page 350
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:TRACe" on page 357
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359
- "*ESR (Standard Event Status Register)" on page 181

:FRANalysis:SOURce:INPut

(see page 1354)

Command Syntax

```
:FRANalysis:SOURce:INPut <source>

<source> ::= CHANnel<n>

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

The :FRANalysis:SOURce:INPut command specifies the analog input channel that is probing the input voltage to the device under test (DUT) in the frequency response analysis.

Query Syntax

```
:FRANalysis:SOURce:INPut?
```

The :FRANalysis:SOURce:INPut? query returns the currently selected channel probing the input voltage.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359

:FRANalysis:SOURce:OUTPut

 (see page 1354)

Command Syntax

```
:FRANalysis:SOURce:OUTPut <source>

<source> ::= CHANnel<n>

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

The :FRANalysis:SOURce:OUTPut command specifies the analog input channel that is probing the output voltage from the device under test (DUT) in the frequency response analysis.

Query Syntax

```
:FRANalysis:SOURce:OUTPut?
```

The :FRANalysis:SOURce:OUTPut? query returns the currently selected channel probing the output voltage.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:WGEN:LOAD" on page 358
- ":FRANalysis:WGEN:VOLTage" on page 359

:FRANalysis:TRACe

(see page 1354)

Command Syntax

```
:FRANalysis:TRACe <selection>

<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
```

The :FRANalysis:TRACe command specifies whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

NOTE This command affects the oscilloscope's front panel graphical user interface (plot and table) as well as when saving analysis data.

Query Syntax

```
:FRANalysis:TRACe?
```

The :FRANalysis:TRACe? query returns a comma-separated list of the types of data that are currently included in the frequency response analysis results, or "NONE" if neither gain nor phase data is included.

Return Format

```
<selection_list><NL>

<selection_list> ::= {"NONE" | "GAIN" | "PHASe" | "GAIN,PHASe"}
```

See Also

- ":FRANalysis:RUN" on page 354
- ":FRANalysis:DATA" on page 347

:FRANalysis:WGEN:LOAD

 (see page 1354)

Command Syntax

```
:FRANalysis:WGEN:LOAD <impedance>

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

The :FRANalysis:WGEN:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax

```
:FRANalysis:WGEN:LOAD?
```

The :FRANalysis:WGEN:LOAD? query returns the current expected output load impedance.

Return Format

```
<impedance><NL>

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

See Also

- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:VOLTage" on page 359

:FRANalysis:WGEN:VOLTage

(see page 1354)

Command Syntax

```
:FRANalysis:WGEN:VOLTage <amplitude>,[<range>]

<amplitude> ::= amplitude in volts in NR3 format

<range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ
   | F10MHZ | F20MHZ}
```

The :FRANalysis:WGEN:VOLTage command specifies the waveform generator's output sine wave amplitude.

Use the :WGEN:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

When the amplitude profile setting is on (:FRANalysis:WGEN:VOLTage:PROFile ON), the <range> option specifies the initial ramp amplitude at the frequency setting. Amplitudes ramp between the settings specified for individual frequencies.

Query Syntax

```
:FRANalysis:WGEN:VOLTage? [<range>]
```

The :FRANalysis:WGEN:VOLTage? query returns the currently specified waveform generator amplitude.

When the amplitude profile setting is on (:FRANalysis:WGEN:VOLTage:PROFile ON), the <range> option specifies the frequency whose initial ramp amplitude is returned.

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":FRANalysis:WGEN:VOLTage:PROFile" on page 360
- ":FRANalysis:DATA" on page 347
- ":FRANalysis:ENABle" on page 348
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:RUN" on page 354
- ":FRANalysis:SOURce:INPut" on page 355
- ":FRANalysis:SOURce:OUTPut" on page 356
- ":FRANalysis:WGEN:LOAD" on page 358

:FRANalysis:WGEN:VOLTage:PROFile

(see page 1354)

Command Syntax

```
:FRANalysis:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}}
```

The :FRANalysis:WGEN:VOLTage:PROFile command enables or disables the ability to specify amplitude ramping within different decades.

Query Syntax

```
:FRANalysis:WGEN:VOLTage:PROFile?
```

The :FRANalysis:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Return Format

```
<setting><NL>

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

See Also

- ":FRANalysis:WGEN:VOLTage" on page 359
- ":FRANalysis:PPDecade" on page 353
- ":FRANalysis:FREQuency:STARt" on page 351
- ":FRANalysis:FREQuency:STOP" on page 352

17 :FUNCtion<m> Commands

Control math functions in the oscilloscope. See "Introduction to :FUNCtion<m> Commands" on page 365.

Table 92 :FUNCtion<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCtion<m>:AVERage:COUNt <count> (see page 367)	:FUNCtion<m>:AVERage:COUNt? (see page 367)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:CLEar (see page 368)	n/a	n/a
:FUNCtion<m>:DISPlay {{0 | OFF} | {1 | ON}} (see page 369)	:FUNCtion<m>:DISPlay? (see page 369)	{0 | 1} <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:BSIZe? (see page 370)	<bin_size> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:CENTer <frequency> (see page 371)	:FUNCtion<m>[:FFT]:CENTer? (see page 371)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from -25 GHz to 25 GHz. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets> (see page 372)	:FUNCtion<m>[:FFT]:DETection:POINts? (see page 372)	<number_of_buckets> ::= an integer in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:DETection:TYPE <type> (see page 373)	:FUNCtion<m>[:FFT]:DETection:TYPE? (see page 373)	<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage} <m> ::= 1 to (# math functions) in NR1 format

Table 92 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency> (see page 374)	:FUNCtion<m>[:FFT]:FREQuency:STARt? (see page 374)	<frequency> ::= the start frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency> (see page 375)	:FUNCtion<m>[:FFT]:FREQuency:STOP? (see page 375)	<frequency> ::= the stop frequency in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:GATE <gating> (see page 376)	:FUNCtion<m>[:FFT]:GATE? (see page 376)	<gating> ::= {NONE | ZOOM} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point> (see page 377)	:FUNCtion<m>[:FFT]:PHASe:REFerence? (see page 377)	<ref_point> ::= {TRIGger | DISPlay} <m> ::= 1-4 in NR1 format
n/a	:FUNCtion<m>[:FFT]:RBWidth? (see page 378)	<resolution_bw> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:READout<n> <readout_type> (see page 379)	:FUNCtion<m>[:FFT]:READout<n>? (see page 379)	<readout_type> ::= {SRATe | BSIZe | RBWidth} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1-2 in NR1 format, 2 is for dedicated FFT function
:FUNCtion<m>[:FFT]:SPAN (see page 380)	:FUNCtion<m>[:FFT]:SPAN? (see page 380)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1 to (# math functions) in NR1 format
n/a	:FUNCtion<m>[:FFT]:SRATe? (see page 381)	<sample_rate> ::= Hz in NR3 format <m> ::= 1 to (# math functions) in NR1 format

Table 92 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>[:FFT]:VTYPe <units> (see page 382)	:FUNCtion<m>[:FFT]:VTYPe? (see page 382)	<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation <units> ::= {DEGRees | RADians} for the FFTPhase operation <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>[:FFT]:WINDow <window> (see page 383)	:FUNCtion<m>[:FFT]:WINDow? (see page 383)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris | BARTlett} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq> (see page 384)	:FUNCtion<m>:FREQuency:HIGHpass? (see page 384)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq> (see page 385)	:FUNCtion<m>:FREQuency:LOWPass? (see page 385)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:INTegrate:IOFFset <input_offset> (see page 386)	:FUNCtion<m>:INTegrate:IOFFset? (see page 386)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:GAIN <value> (see page 387)	:FUNCtion<m>:LINear:GAIN? (see page 387)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:LINear:OFFSet <value> (see page 388)	:FUNCtion<m>:LINear:OFFSet? (see page 388)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:OFFSet <offset> (see page 389)	:FUNCtion<m>:OFFSet? (see page 389)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format

Table 92 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:OPERation <operation> (see page 390)	:FUNCtion<m>:OPERation? (see page 392)	<operation> ::= {ADD | SUBTract | MULTiply | DIVide | INTegrate | DIFF | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | AVERage | LINear | MAXimum | MINimum | PEAK | MAXHold | MINHold | TRENd} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:RANGe <range> (see page 394)	:FUNCtion<m>:RANGe? (see page 394)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:REFerence <level> (see page 395)	:FUNCtion<m>:REFerence? (see page 395)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SCALe <scale value>[<suffix>] (see page 396)	:FUNCtion<m>:SCALe? (see page 396)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SMOoth:POINts <points> (see page 397)	:FUNCtion<m>:SMOoth:POINts? (see page 397)	<points> ::= odd integer in NR1 format

Table 92 :FUNCtion<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion<m>:SOURce1 <source> (see page 398)	:FUNCtion<m>:SOURce1? (see page 398)	<source> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>} <n> ::= 1 to (# analog channels) in NR1 format <c> ::= {1}, must be lower than <m> <r> ::= 1 to (# ref waveforms) in NR1 format <b> ::= {1 | 2} <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:SOURce2 <source> (see page 400)	:FUNCtion<m>:SOURce2? (see page 400)	<source> ::= {CHANnel<n> | WMEMory<r> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format
:FUNCtion<m>:TRENd:NMEasurement MEAS<n> (see page 401)	:FUNCtion<m>:TRENd:NMEasurement? (see page 401)	<n> ::= # of installed measurement, from 1 to 8 <m> ::= 1 to (# math functions) in NR1 format

Introduction to :FUNCtion<m> Commands

The FUNCtion subsystem controls the math functions in the oscilloscope. Two math functions are available — the <m> in :FUNCtion<m> can be from 1 to 2.

The math function operator, transform, filter, or visualization is selected using the :FUNCtion<m>:OPERation command. Depending on the selected operation, there may be other commands for specifying options for that operation. See ":FUNCtion<m>:OPERation" on page 390.

The SOURce1, DISPlay, RANGe, and OFFSet (or REFerence) commands apply to any function.

Reporting the Setup

Use :FUNCtion<m>? to query setup information for the FUNCtion subsystem.

Return Format

The following is a sample response from the :FUNCtion1? query. In this case, the query was issued following a *RST command.

```
:FUNC1:OPER ADD;DISP 0;SOUR1 CHAN1;SOUR2 CHAN2;RANG +8.00E+00;OFFS
+0.0E+00
```

:FUNCtion<m>:AVERage:COUNt

 (see page 1354)

Command Syntax

```
:FUNCtion<m>:AVERage:COUNt <count>

<count> ::= an integer from 2 to 65536 in NR1 format

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

The :FUNCtion<m>:AVERage:COUNt command sets the number of waveforms to be averaged together.

The number of averages can be set from 2 to 65536 in increments of powers of 2.

Increasing the number of averages will increase resolution and reduce noise.

Query Syntax

```
:FUNCtion<m>:AVERage:COUNt?
```

The :FUNCtion<m>:AVERage:COUNt? query returns the number of waveforms to be averaged together.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 65536 in NR1 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:CLEar

 (see page 1354)

Command Syntax `:FUNCtion<m>:CLEar`

When the :FUNCtion<m>:OPERation is AVERage, MAXHold, or MINHold, the :FUNCtion<m>:CLEar command clears the number of evaluated waveforms.

See Also

- ":FUNCtion<m>:AVERage:COUNt" on page 367

:FUNCtion<m>:DISPlay

(see page 1354)

Command Syntax

```
:FUNCtion<m>:DISPlay <display>

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

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

The :FUNCtion<m>:DISPlay command turns the display of the function on or off. When ON is selected, the function operates as specified by the other :FUNCtion<m> commands.

NOTE

Automatic vertical scaling of the math function waveform occurs when the function display is turned from off to on. If you wait for the operation to complete (with an *OPC? query for example), then query the math functions's scale, you can see the vertical scaling that was automatically determined.

One math function can be displayed at a time. If one math function is on and then another math function is turned on, the first math function will be turned off.

When math functions are off, they are still calculated so that they can be cascaded, that is, used as a source for a higher math function.

Query Syntax

```
:FUNCtion<m>:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- "*OPC (Operation Complete)" on page 185
- ":FUNCtion<m>:SCALe" on page 396
- ":VIEW" on page 239
- ":BLANk" on page 210
- ":STATus" on page 236

:FUNCtion<m>[:FFT]:BSIZe

N (see page 1354)

Query Syntax

```
:FUNCtion<m>[:FFT]:BSIZe?
<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:BSIZe? query returns the Bin Size setting for the FFT.

Return Format

```
<bin_size><NL>
<bin_size> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 379

:FUNCtion<m>[:FFT]:CENTer

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:CENTer <frequency>

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

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

The :FUNCtion<m>[:FFT]:CENTer command sets the center frequency when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion<m>[:FFT]:CENTer?
```

The :FUNCtion<m>[:FFT]:CENTer? query returns the current center frequency in Hertz.

Return Format

```
<frequency><NL>

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

NOTE

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>[:FFT]:SPAN" on page 380
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:SCALe" on page 1047

:FUNCtion<m>[:FFT]:DETection:POINts

N (see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:DETection:POINts <number_of_buckets>

<number_of_buckets> ::= an integer in NR1 format

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:DETection:POINts command specifies the maximum number of points that the FFT detector should decimate to. This is also the number of buckets that sampled FFT points are grouped into before the selected detection type reduction (decimation) is applied.

The minimum number of points is 640.

When precision analysis is off, the maximum number of points is the measurement record limit of 64K.

When precision analysis is on (see ":SYSTem:PRECision" on page 1021), the maximum number of points is 1/2 the power-of-two value needed to hold the precision analysis record length.

Query Syntax

```
:FUNCtion<m>[:FFT]:DETection:POINts?
```

The :FUNCtion<m>[:FFT]:DETection:POINts? query returns the FFT detector points setting.

Return Format

```
<number_of_buckets><NL>

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

See Also

- ":SYSTem:PRECision" on page 1021
- ":FUNCtion<m>[:FFT]:DETection:TYPE" on page 373

:FUNCtion<m>[:FFT]:DETection:TYPE

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:DETection:TYPE <type>

<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage}

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:DETection:TYPE command sets the FFT detector decimation type.

Detectors give you a way of manipulating the acquired data to emphasize different features of the data. Detectors reduce (decimate) the number of FFT points to at most the number of specified detector points. In this reduction, sampled FFT points are bucketized, that is, split into a number of groups that equals the specified number of detector points. Then, the points in each bucket are reduced to a single point according to the selected detection type. The detector types are:

- OFF – No detector is used.
- SAMPle – Takes the point nearest to the center of every bucket.
- PPOSitive (+ Peak) – Takes the most positive point in every bucket.
- PNEGative (- Peak) – Takes the most negative point in every bucket.
- AVERage – Takes the average of all points in every bucket.
- NORMal – Implements a rosenfell algorithm. This method to picks either the minimum or maximum sample in every bucket depending on whether the data is monotonically increasing, decreasing, or varying. For details, see the Spectrum Analysis Basics application note at www.keysight.com.

When detectors are used, the FFT's output is decimated, and any analysis is performed on the reduced or detected data set.

Query Syntax

```
:FUNCtion<m>[:FFT]:DETection:TYPE?
```

The :FUNCtion<m>[:FFT]:DETection:TYPE? query returns the FFT detector type setting

Return Format

```
<type><NL>

<type> ::= {OFF | SAMPle | PPOSitive | PNEGative | NORMal | AVERage}
```

See Also

- ":FUNCtion<m>[:FFT]:DETection:POINts" on page 372

:FUNCtion<m>[:FFT]:FREQuency:STARt

N (see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STARt <frequency>

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

<frequency> ::= the start frequency in NR3 format.
```

The :FUNCtion<m>[:FFT]:FREQuency:STARt command sets the start frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FUNCtion<m>[:FFT]:CENTer and :FUNCtion<m>[:FFT]:SPAN commands.

Query Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STARt?
```

The :FUNCtion<m>[:FFT]:FREQuency:STARt? query returns the current start frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the start frequency in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:FREQuency:STOP" on page 375
- ":FUNCtion<m>[:FFT]:CENTer" on page 371
- ":FUNCtion<m>[:FFT]:SPAN" on page 380

:FUNCtion<m>[:FFT]:FREQuency:STOP

N (see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STOP <frequency>

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

<frequency> ::= the stop frequency in NR3 format.
```

The :FUNCtion<m>[:FFT]:FREQuency:STOP command sets the stop frequency in the FFT (Fast Fourier Transform) math function's displayed range.

The FFT (Fast Fourier Transform) math function's displayed range can also be set with the :FUNCtion<m>[:FFT]:CENTer and :FUNCtion<m>[:FFT]:SPAN commands.

Query Syntax

```
:FUNCtion<m>[:FFT]:FREQuency:STOP?
```

The :FUNCtion<m>[:FFT]:FREQuency:STOP? query returns returns the current stop frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the stop frequency in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:FREQuency:STARt" on page 374
- ":FUNCtion<m>[:FFT]:CENTer" on page 371
- ":FUNCtion<m>[:FFT]:SPAN" on page 380

:FUNCtion<m>[:FFT]:GATE

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:GATE <gating>

<m> ::= 1-4 in NR1 format

<gating> ::= {NONE | ZOOM}
```

The :FUNCtion<m>[:FFT]:GATE command specifies whether the FFT is performed on the Main time base window (NONE) or the ZOOM window when the zoomed time base is displayed.

Query Syntax

```
:FUNCtion<m>[:FFT]:GATE?
```

The :FUNCtion<m>[:FFT]:GATE? query returns the gate setting.

Return Format

```
<gating><NL>

<gating> ::= {NONE | ZOOM}
```

See Also

- ":FUNCtion<m>[:FFT]:VTYPe" on page 382
- ":FUNCtion<m>[:FFT]:WINDow" on page 383
- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>[:FFT]:PHASe:REFerence

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:PHASe:REFerence <ref_point>

<ref_point> ::= {TRIGger | DISPlay}

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:PHASe:REFerence command sets the reference point for calculating the FFT Phase function to either the trigger point or beginning of the displayed waveform.

Query Syntax

```
:FUNCtion<m>[:FFT]:PHASe:REFerence?
```

The :FUNCtion<m>[:FFT]:PHASe:REFerence? query returns the selected reference point.

Return Format

```
<ref_point><NL>

<ref_point> ::= {TRIGger | DISPlay}
```

See Also

- ":FUNCtion<m>:OPERation" on page 390
- "Introduction to :FUNCtion<m> Commands" on page 365

:FUNCtion<m>[:FFT]:RBWidth

N (see page 1354)

Query Syntax

```
:FUNCtion<m>[:FFT]:RBWidth?

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:RBWidth? query returns the Resolution Bandwidth setting for the FFT.

Return Format

```
<resolution_bw><NL>

<resolution_bw> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 379

:FUNCtion<m>[:FFT]:READout<n>

N (see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:READout<n> <readout_type>

<readout_type> ::= {SRATe | BSIZe | RBWidth}

<m> ::= 1-4 in NR1 format

<n> ::= 1-2 in NR1 format, 2 is valid only on oscilloscopes that have
   the dedicated FFT function
```

The :FUNCtion<m>[:FFT]:READout<n> command selects from these types of readouts for the FFT:

- SRATe – Sample Rate
- BSIZe – Bin Size
- RBWidth – Resolution Bandwidth

Note that READout1 is used for both the dedicated FFT and the general-purpose math FFT function and READout2 is used only for oscilloscopes that have a dedicated FFT function (like the 3000T X-Series).

Query Syntax

```
:FUNCtion<m>[:FFT]:READout<n>?
```

The :FUNCtion<m>[:FFT]:READout<n>? query returns the readout selection.

Return Format

```
<readout_type><NL>

<readout_type> ::= {SRAT | BSIZ | RBW}
```

See Also

- ":FUNCtion<m>[:FFT]:BSIZe" on page 370
- ":FUNCtion<m>[:FFT]:RBWidth" on page 378
- ":FUNCtion<m>[:FFT]:SRATe" on page 381

:FUNCtion<m>[:FFT]:SPAN

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:SPAN <span>

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

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

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

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

Query Syntax

```
:FUNCtion<m>[:FFT]:SPAN?
```

The :FUNCtion<m>[:FFT]:SPAN? query returns the current frequency span in Hertz.

NOTE

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

Return Format

```
<span><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>[:FFT]:CENTer" on page 371
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:SCALe" on page 1047

:FUNCtion<m>[:FFT]:SRATe

N (see page 1354)

Query Syntax

```
:FUNCtion<m>[:FFT]:SRATe?

<m> ::= 1-4 in NR1 format
```

The :FUNCtion<m>[:FFT]:SRATe? query returns the Sample Rate setting for the FFT.

Return Format

```
<sample_rate><NL>

<sample_rate> ::= Hz in NR3 format
```

See Also

- ":FUNCtion<m>[:FFT]:READout<n>" on page 379

:FUNCtion<m>[:FFT]:VTYPe

(see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:VTYPe <units>

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

<units> ::= {DECibel | VRMS} for the FFT (magnitude) operation

<units> ::= {DEGRees | RADians} for the FFTPhase operation
```

The :FUNCtion<m>[:FFT]:VTYPe command specifies FFT vertical units.

For the FFT (Magnitude) operation units, DECibel equates to the user interface's Logarithmic selection, and VRMS equates to the user interface's Linear selection.

Query Syntax

```
:FUNCtion<m>[:FFT]:VTYPe?
```

The :FUNCtion<m>[:FFT]:VTYPe? query returns the current FFT vertical units.

Return Format

```
<units><NL>

<units> ::= {DEC | VRMS} for the FFT (magnitude) operation

<units> ::= {DEGR | RAD} for the FFTPhase operation
```

See Also

- ":FUNCtion<m>[:FFT]:GATE" on page 376
- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>[:FFT]:WINDow

N (see page 1354)

Command Syntax

```
:FUNCtion<m>[:FFT]:WINDow <window>

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

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

The :FUNCtion<m>[:FFT]:WINDow command allows the selection of different windowing transforms or operations for the FFT (Fast Fourier Transform) function.

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

- RECTangular – useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing – useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements. This is the default window.
- FLATtop – best for making accurate amplitude measurements of frequency peaks.
- BHARris (Blackman-Harris) – reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).
- BARTlett – (triangular, with end points at zero) window is similar to the Hanning window in that it is good for making accurate frequency measurements, but its higher and wider secondary lobes make it not quite as good for resolving frequencies that are close together.

Query Syntax

```
:FUNCtion<m>[:FFT]:WINDow?
```

The :FUNCtion<m>[:FFT]:WINDow? query returns the value of the window selected for the FFT function.

Return Format

```
<window><NL>

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

See Also

- ":FUNCtion<m>[:FFT]:GATE" on page 376
- "Introduction to :FUNCtion<m> Commands" on page 365

:FUNCtion<m>:FREQuency:HIGHpass

(see page 1354)

Command Syntax

```
:FUNCtion<m>:FREQuency:HIGHpass <3dB_freq>

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

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion<m>:FREQuency:HIGHpass command sets the high-pass filter's -3 dB cutoff frequency.

The high-pass filter is a single-pole high pass filter.

Query Syntax

```
:FUNCtion<m>:FREQuency:HIGHpass?
```

The :FUNCtion<m>:FREQuency:HIGHpass query returns the high-pass filter's cutoff frequency.

Return Format

```
<3dB_freq><NL>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:FREQuency:LOWPass

 (see page 1354)

Command Syntax

```
:FUNCtion<m>:FREQuency:LOWPass <3dB_freq>

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

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion<m>:FREQuency:LOWPass command sets the low-pass filter's -3 dB cutoff frequency.

The low-pass filter is a 4th order Bessel-Thompson filter.

Query Syntax

```
:FUNCtion<m>:FREQuency:LOWPass?
```

The :FUNCtion<m>:FREQuency:LOWPass query returns the low-pass filter's cutoff frequency.

Return Format

```
<3dB_freq><NL>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:INTegrate:IOFFset

 (see page 1354)

Command Syntax

```
:FUNCtion<m>:INTegrate:IOFFset <input_offset>

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

<input_offset> ::= DC offset correction in NR3 format.
```

The :FUNCtion<m>:INTegrate:IOFFset command lets you enter a DC offset correction factor for the integrate math waveform input signal. This DC offset correction lets you level a "ramp"ed waveform.

Query Syntax

```
:FUNCtion<m>:INTegrate:IOFFset?
```

The :FUNCtion<m>:INTegrate:IOFFset? query returns the current input offset value.

Return Format

```
<input_offset><NL>

<input_offset> ::= DC offset correction in NR3 format.
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:LINear:GAIN

N (see page 1354)

Command Syntax

```
:FUNCtion<m>:LINear:GAIN <value>

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

<value> ::= 'A' in Ax + B, value in NR3 format
```

The :FUNCtion<m>:LINear:GAIN command specifies the 'A' value in the Ax + B operation.

Query Syntax

```
:FUNCtion<m>:LINear:GAIN?
```

The :FUNCtion<m>:LINear:GAIN query returns the gain value.

Return Format

```
<value><NL>

<value> ::= 'A' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:LINear:OFFSet

(see page 1354)

Command Syntax

```
:FUNCtion<m>:LINear:OFFSet <value>

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

<value> ::= 'B' in Ax + B, value in NR3 format
```

The :FUNCtion<m>:LINear:OFFSet command specifies the 'B' value in the Ax + B operation.

Query Syntax

```
:FUNCtion<m>:LINear:OFFSet?
```

The :FUNCtion<m>:LINear:OFFSet query returns the offset value.

Return Format

```
<value><NL>

<value> ::= 'B' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:OFFSet

N (see page 1354)

Command Syntax

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

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

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

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

NOTE **The :FUNCtion<m>:OFFSet command is equivalent to the :FUNCtion<m>:REFerence command.**

Query Syntax

```
:FUNCtion<m>:OFFSet?
```

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

Return Format

```
<offset><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:RANGe" on page 394
- ":FUNCtion<m>:REFerence" on page 395
- ":FUNCtion<m>:SCALe" on page 396

:FUNCtion<m>:OPERation

N (see page 1354)

Command Syntax

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

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

<operation> ::= {ADD | SUBTract | MULTiply | DIVide | DIFF | INTegrate
   | FFT | FFTPhase | SQRT | MAGNify | ABSolute | SQUare | LN | LOG
   | EXP | TEN | LOWPass | HIGHpass | AVERage | SMOoth | ENVelope
   | LINear | MAXimum | MINimum | PEAK | MAXHold | MINHold | TRENd}
```

The :FUNCtion<m>:OPERation command sets the desired waveform math operator, transform, filter or visualization:

- Operators:
  - ADD – Source1 + source2.
  - SUBTract – Source1 - source2.
  - MULTiply – Source1 * source2.
  - DIVide – Source1 / source2.

  Operators perform their function on two analog channel sources.
- Transforms:
  - DIFF – Differentiate
  - INTegrate – The INTegrate:IOFFset command lets you specify a DC offset correction factor.
  - FFT (magnitude) – Using the Fast Fourier Transform (FFT), this operation displays the magnitudes of the frequency content that makes up the source waveform. The FFT takes the digitized time record of the specified source and transforms it to the frequency domain.

    The SPAN, CENTer, VTYPe, and WINDow commands are used for FFT functions. When FFT is selected, the horizontal cursors change from time to frequency (Hz), and the vertical cursors change from volts to decibels or V RMS.
  - FFTPhase – Using the Fast Fourier Transform (FFT), this operation shows the phase relationships of the frequency content that makes up the source waveform. The FFT takes the digitized time record of the specified source and transforms it to the frequency domain.

    The SPAN, CENTer, VTYPe, and WINDow commands are used for FFT functions. When FFTPhase is selected, the horizontal cursors change from time to frequency (Hz), and the vertical cursors change from volts to degrees or radians.
  - LINear – Ax + B – The LINear commands set the gain (A) and offset (B) values for this function.

- SQUare
- SQRT – Square root
- ABSolute – Absolute Value
- LOG – Common Logarithm
- LN – Natural Logarithm
- EXP – Exponential (e^X)
- TEN – Base 10 exponential (10^X)

Transforms operate on a single analog channel source or on lower math functions.

- Filters:
  - LOWPass – Low pass filter – The FREQuency:LOWPass command sets the -3 dB cutoff frequency.
  - HIGHpass – High pass filter – The FREQuency:HIGHpass command sets the -3 dB cutoff frequency.
  - AVERage – Averaged value – The AVERage:COUNt command specifies the number of averages.

    Unlike acquisition averaging, the math averaging operator can be used to average the data on a single analog input channel or math function.

    If acquisition averaging is also used, the analog input channel data is averaged and the math function averages it again. You can use both types of averaging to get a certain number of averages on all waveforms and an increased number of averages on a particular waveform.

    Averages are calculated using a "decaying average" approximation, where:

    next_average = current_average + (new_data - current_average)/N

    Where N starts at 1 for the first acquisition and increments for each following acquisition until it reaches the selected number of averages, where it holds.
  - SMOoth – Smoothing – The resulting math waveform is the selected source with a normalized rectangular (boxcar) FIR filter applied.

    The boxcar filter is a moving average of adjacent waveform points, where the number of adjacent points is specified by the SMOoth:POINts command. You can choose an odd number of points, from 3 to 999.

    The smoothing operator limits the bandwidth of the source waveform. The smoothing operator can be used, for example, to smooth measurement trend waveforms.
  - ENVelope – Envelope – The resulting math waveform is the amplitude envelope for an amplitude modulated (AM) input signal.

This function uses a Hilbert transform to get the real (in-phase, I) and imaginary (quadrature, Q) parts of the input signal and then performs a square root of the sum of the real and imaginary parts to get the demodulated amplitude envelope waveform.

Filters operate on a single analog channel source or on a lower math function.

- Visualizations:
  - MAGNify — Operates on a single analog channel source or on a lower math function.
  - MAXimum — This operator is like the MAXHold operator without the hold. The maximum vertical values found at each horizontal bucket are used to build a waveform.
  - MINimum — This operator is like the MINHold operator without the hold. The minimum vertical values found at each horizontal bucket are used to build a waveform.
  - PEAK — The PEAK operator is like the MAXimum operator minus the MINimum operator. At each horizontal bucket, the minimum vertical values found are subtracted from the maximum vertical values found to build a waveform.
  - MAXHold — Operates on a single analog channel source or on a lower math function. The Max Hold (or Max Envelope) operator records the maximum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform.
  - MINHold — Operates on a single analog channel source or on a lower math function. The Min Hold (or Min Envelope) operator records the minimum vertical values found at each horizontal bucket across multiple analysis cycles and uses those values to build a waveform.
  - TRENd — Measurement trend — Operates on a single analog channel source. The TRENd:NMEasurement command selects the measurement whose trend you want to measure.

NOTE **If a math function is on (see :FUNCtion<m>:DISPlay), changing the operator will cause an automatic vertical scaling of the new math function waveform. If you wait for the operation to complete (with an *OPC? query for example), then query the math functions's scale, you can see the vertical scaling that was automatically determined.**

Query Syntax

```
:FUNCtion<m>:OPERation?
```

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

Return Format

```
<operation><NL>

<operation> ::= {ADD | SUBT | MULT | DIV | INT | DIFF | FFT | FFTP
   | SQRT | MAGN | ABS | SQU | LN | LOG | EXP | TEN | LOWP | HIGH
   | AVER | SMO | ENV | LIN | MAX | MIN | PEAK | MAXH | MINH | TREN}
```

See Also
- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:DISPlay" on page 369
- "*OPC (Operation Complete)" on page 185
- ":FUNCtion<m>:SOURce1" on page 398
- ":FUNCtion<m>:SOURce2" on page 400
- ":FUNCtion<m>:INTegrate:IOFFset" on page 386
- ":FUNCtion<m>[:FFT]:SPAN" on page 380
- ":FUNCtion<m>[:FFT]:CENTer" on page 371
- ":FUNCtion<m>[:FFT]:PHASe:REFerence" on page 377
- ":FUNCtion<m>[:FFT]:VTYPe" on page 382
- ":FUNCtion<m>[:FFT]:WINDow" on page 383
- ":FUNCtion<m>:LINear:GAIN" on page 387
- ":FUNCtion<m>:LINear:OFFSet" on page 388
- ":FUNCtion<m>:FREQuency:LOWPass" on page 385
- ":FUNCtion<m>:FREQuency:HIGHpass" on page 384
- ":FUNCtion<m>:AVERage:COUNt" on page 367
- ":FUNCtion<m>:SMOoth:POINts" on page 397
- ":FUNCtion<m>:TRENd:NMEasurement" on page 401

:FUNCtion<m>:RANGe

 (see page 1354)

Command Syntax

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

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

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

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

Query Syntax

```
:FUNCtion<m>:RANGe?
```

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

Return Format

```
<range><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:SCALe" on page 396

:FUNCtion<m>:REFerence

 (see page 1354)

Command Syntax

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

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

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

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

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

Query Syntax

```
:FUNCtion<m>:REFerence?
```

The :FUNCtion<m>:REFerence? query outputs the current reference level value for the selected function.

Return Format

```
<level><NL>

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

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OFFSet" on page 389
- ":FUNCtion<m>:RANGe" on page 394
- ":FUNCtion<m>:SCALe" on page 396

:FUNCtion<m>:SCALe

 (see page 1354)

Command Syntax

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

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

<scale value> ::= vertical units/div value in NR3 format

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

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

NOTE Automatic vertical scaling of the math function waveform occurs when the function display is turned from off to on (see :FUNCtion<m>:DISPlay) or, if the function is already on, when the operation is changed (see :FUNCtion<m>:OPERation). If you want to change the math function's vertical scaling, you should do it after the math function display is turned on or after the operation is changed.

Query Syntax

```
:FUNCtion<m>:SCALe?
```

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

Return Format

```
<scale value><NL>

<scale value> ::= vertical units/div value in NR3 format
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:DISPlay" on page 369
- ":FUNCtion<m>:OPERation" on page 390
- ":FUNCtion<m>:RANGe" on page 394

:FUNCtion<m>:SMOoth:POINts

 (see page 1354)

Command Syntax

```
:FUNCtion<m>:SMOoth:POINts <points>

<points> ::= odd integer in NR1 format
```

When the :FUNCtion<m>:OPERation is SMOoth, the :FUNCtion<m>:SMOoth:POINts command sets the number of smoothing points to use.

You can choose an odd number of points, from 3 up to half of the measurement record or precision analysis record.

Query Syntax

```
:FUNCtion<m>:SMOoth:POINts?
```

The :FUNCtion<m>:SMOoth:POINts? query returns the number of smoothing points specified.

Return Format

```
<points><NL>
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:SOURce1

N (see page 1354)

Command Syntax

```
:FUNCtion<m>:SOURce1 <value>

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

<value> ::= {CHANnel<n> | FUNCtion<c> | MATH<c> | WMEMory<r> | BUS<b>}

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

<c> ::= {1}, must be lower than <m>

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

<b> ::= {1 | 2}
```

The :FUNCtion<m>:SOURce1 command is used for any :FUNCtion<m>:OPERation selection. This command selects the first source for the operator math functions or the single source for the transform functions, filter functions, or visualization functions.

The FUNCtion<c> or MATH<c> parameters are available for the transform functions, filter functions, and the magnify visualization function (see "Introduction to :FUNCtion<m> Commands" on page 365) when <c> is lower than <m>.

In other words, higher math functions can operate on lower math functions when using operators other than the simple arithmetic operations (+, - , *, /). For example, if :FUNCtion1:OPERation is a SUBTract of CHANnel1 and CHANnel2, the :FUNCtion2:OPERation could be set up as a FFT operation on the FUNCtion1 source. These are called cascaded math functions.

To cascade math functions, select the lower math function using the :FUNCtion<m>:SOURce1 command.

When cascading math functions, to get the most accurate results, be sure to vertically scale lower math functions so that their waveforms take up the full screen without being clipped.

The BUS<m> parameter is available for the bus charting visualization functions.

NOTE **Another shorthand notation for SOURce1 in this command/query (besides SOUR1) is SOUR.**

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce1 command reports error -221,"Settings conflict" because the TRENd function operates on a measurement and not a source waveform.

Query Syntax

```
:FUNCtion<m>:SOURce1?
```

The :FUNCtion<m>:SOURce1? query returns the current source1 for function operations.

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce1? query returns the source of the measurement.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | FUNC<c> | WMEM<r> | BUS<b>}
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion<m>:SOURce2

N (see page 1354)

Command Syntax

```
:FUNCtion<m>:SOURce2 <value>

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

<value> ::= {CHANnel<n> | WMEMory<r> | NONE}

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

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

The :FUNCtion<m>:SOURce2 command specifies the second source for math operator functions that have two sources. (The :FUNCtion<m>:SOURce1 command specifies the first source.)

The :FUNCtion<m>:SOURce2 setting is not used for the transform functions, filter functions, or visualization functions (except when the measurement trend visualization's measurement requires two sources).

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce2 command reports error -221,"Settings conflict" because the TRENd function operates on a measurement and not a source waveform.

Query Syntax

```
:FUNCtion<m>:SOURce2?
```

The :FUNCtion<m>:SOURce2? query returns the currently specified second source for math operations.

When :FUNCtion<m>:OPERation is TRENd, the :FUNCtion<m>:SOURce2? query returns the source of the measurement.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | WMEM<r> | NONE}
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390
- ":FUNCtion<m>:SOURce1" on page 398

:FUNCtion<m>:TRENd:NMEasurement

N (see page 1354)

Command Syntax

```
:FUNCtion<m>:TRENd:NMEasurement MEAS<n>

<n> ::= # of installed measurement, from 1 to 8

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

The :FUNCtion<m>:TRENd:NMEasurement command selects the measurement whose trend is shown in the math waveform.

There are 8 locations (or slots) that installed measurements can occupy. The MEAS<n> parameter specifies the location of the measurement whose trend you want to analyze.

You can view the trend math function waveform for these installed measurements:

- :MEASure:VAVerage, <interval> ::= CYCLe
- :MEASure:VRMS, <type> ::= AC (AC RMS)
- :MEASure:VRATio, <interval> ::= CYCLe
- :MEASure:PERiod
- :MEASure:FREQuency
- :MEASure:PWIDth
- :MEASure:NWIDth
- :MEASure:DUTYcycle
- :MEASure:NDUTy
- :MEASure:RISetime
- :MEASure:FALLtime

Query Syntax

```
:FUNCtion<m>:TRENd:NMEasurement?
```

The :FUNCtion<m>:TRENd:NMEasurement? query returns the selected measurement.

If no measurements are installed, the :FUNCtion<m>:TRENd:NMEasurement? query will return NONE.

Return Format

```
MEAS<n><NL>

<n> ::= # of installed measurement, from 1 to 8
```

See Also

- ":FUNCtion<m>:OPERation" on page 390

18 :HCOPY Commands

Set and query the selection of hardcopy device and formatting options.

Table 93 :HCOPY Commands Summary

Command	Query	Options and Query Returns
n/a	:HCOPY:SDUMp:DATA? (see page 404)	<display_data> ::= binary block data in IEEE-488.2 # format.
:HCOPY:SDUMp:FORMat <format> (see page 405)	:HCOPY:SDUMp:FORMat? (see page 405)	<format> ::= {BMP | BMP8bit | PNG}

:HCOPY:SDUMp:DATA

N (see page 1354)

Query Syntax

```
:HCOPY:SDUMp:DATA? [<format>]

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

The :HCOPY:SDUMp:DATA? query reads and returns screen image data. You can choose 24-bit BMP, 8-bit BMP8bit, or 24-bit PNG formats.

In addition to the <format> option of this query, the screen image data format can also be set by the :HCOPY:SDUMp:FORMat command or the front panel's **Main Menu > File > Save Menu > Format** softkey, when an image format (instead of a setup or data format) is selected.

If no <format> option is specified with this query, the screen image data is returned in the currently selected format. After a *RST (factory default) command, the PNG format is selected by default.

If the <format> option is specified with this query, the format setting will affect the front panel's **Main Menu > File > Save Menu > Format** softkey setting if the **Format** softkey currently selects an image format (instead of a setup or data format).

Screen image data is returned in the IEEE-488.2 # binary block data format.

Return Format

```
<display_data><NL>

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

See Also

- ":HCOPY:SDUMp:FORMat" on page 405
- ":DISPlay:DATA" on page 321
- "*RST (Reset)" on page 189

:HCOPY:SDUMp:FORMat

N (see page 1354)

Command Syntax

```
:HCOPY:SDUMp:FORMat <format>

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

The :HCOPY:SDUMp:FORMat command specifies the format for screen image data: 24-bit PNG, 24-bit BMP, or 8-bit BMP8bit.

The :HCOPY:SDUMp:FORMat setting will persist when cycling power but will be reset to PNG after a *RST (factory default) command.

The :HCOPY:SDUMp:FORMat setting will affect the front panel's **Main Menu > File > Save Menu > Format** softkey setting if the **Format** softkey currently selects an image format (instead of a setup or data format).

Query Syntax

```
:HCOPY:SDUMp:FORMat?
```

The :HCOPY:SDUMp:FORMat? query returns the specified screen image data format.

The screen image data format can be set by the :HCOPY:SDUMp:FORMat command or the front panel's **Main Menu > File > Save Menu > Format** softkey, when it selects an image format (instead of a setup or data format).

Return Format

```
<format><NL>

<format> ::= {PNG | BMP | BMP8}
```

See Also

- ":HCOPY:SDUMp:DATA" on page 404
- ":DISPlay:DATA" on page 321
- "*RST (Reset)" on page 189

19 :LISTer Commands

Table 94 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 408)	<binary_block> ::= comma-separated data with newlines at the end of each row
:LISTer:DISPlay {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2} | ALL} (see page 409)	:LISTer:DISPlay? (see page 409)	{OFF | SBUS1 | SBUS2 | ALL}
:LISTer:REFerence <time_ref> (see page 410)	:LISTer:REFerence? (see page 410)	<time_ref> ::= {TRIGger | PREVious}

Introduction to :LISTer Commands

The LISTer subsystem is used to turn on/off the serial decode Lister display and return data from the Lister display.

:LISTer:DATA

(see page 1354)

Query Syntax `:LISTer:DATA?`

The :LISTer:DATA? query returns the lister data.

Return Format `<binary block><NL>`

```
<binary_block> ::= comma-separated data with newlines at the
                   end of each row
```

See Also

- "Introduction to :LISTer Commands" on page 407
- ":LISTer:DISPlay" on page 409
- "Definite-Length Block Response Data" on page 171

:LISTer:DISPlay

N (see page 1354)

Command Syntax

```
:LISTer:DISPlay <value>

<value> ::= {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2} | ALL}
```

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".

When set to "ALL", the decode information for different buses is interleaved in time.

Serial bus decode must be on before it can be displayed in the Lister.

Query Syntax

```
:LISTer:DISPlay?
```

The :LISTer:DISPlay? query returns the Lister display setting.

Return Format

```
<value><NL>

<value> ::= {OFF | SBUS1 | SBUS2 | ALL}
```

See Also

- "Introduction to :LISTer Commands" on page 407
- ":SBUS<n>:DISPlay" on page 724
- ":LISTer:DATA" on page 408

:LISTer:REFerence

(see page 1354)

Command Syntax

```
:LISTer:REFerence <time_ref>

<time_ref> ::= {TRIGger | PREVious}
```

The :LISTer:REFerence command selects whether the time value for a Lister row is relative to the trigger or the previous Lister row.

Query Syntax

```
:LISTer:REFerence?
```

The :LISTer:REFerence? query returns the Lister time reference setting.

Return Format

```
<time_ref><NL>

<time_ref> ::= {TRIGger | PREVious}
```

See Also

- "Introduction to :LISTer Commands" on page 407
- ":SBUS<n>:DISPlay" on page 724
- ":LISTer:DATA" on page 408
- ":LISTer:DISPlay" on page 409

20 :MARKer Commands

Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). See "Introduction to :MARKer Commands" on page 413.

Table 95 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 414)	<return_value> ::= •Y/•X value in NR3 format
:MARKer:MODE <mode> (see page 415)	:MARKer:MODE? (see page 415)	<mode> ::= {OFF | MEASurement | MANual | WAVeform | BINary | HEX}
:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}} (see page 416)	:MARKer:X1:DISPlay? (see page 416)	<setting> ::= {0 | 1}
:MARKer:X1Position <position>[suffix] (see page 417)	:MARKer:X1Position? (see page 417)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 418)	:MARKer:X1Y1source? (see page 418)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}} (see page 419)	:MARKer:X2:DISPlay? (see page 419)	<setting> ::= {0 | 1}

Table 95 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X2Position <position>[suffix] (see page 420)	:MARKer:X2Position? (see page 420)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 421)	:MARKer:X2Y2source? (see page 421)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 422)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 423)	:MARKer:XUNits? (see page 423)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}
:MARKer:XUNits:USE (see page 424)	n/a	n/a
:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}} (see page 425)	:MARKer:Y1:DISPlay? (see page 425)	<setting> ::= {0 | 1}
:MARKer:Y1Position <position>[suffix] (see page 426)	:MARKer:Y1Position? (see page 426)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}} (see page 427)	:MARKer:Y2:DISPlay? (see page 427)	<setting> ::= {0 | 1}
:MARKer:Y2Position <position>[suffix] (see page 428)	:MARKer:Y2Position? (see page 428)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format

Table 95 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MARKer:YDELta? (see page 429)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 430)	:MARKer:YUNits? (see page 430)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 431)	n/a	n/a

Introduction to :MARKer Commands

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

Reporting the Setup

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

Return Format

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

```
:MARK:X1Y1 CHAN1;X2Y2 CHAN1;MODE MAN
```

:MARKer:DYDX

(see page 1354)

Query Syntax `:MARKer:DYDX?`

The MARKer:DYDX? query returns the cursor ΔY/ΔX value.

X cursor units are set by the :MARKer:XUNits command.

NOTE If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format `<value><NL>`

`<value> ::= •Y/•X value in NR3 format.`

See Also
- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:XUNits" on page 423

:MARKer:MODE

(see page 1354)

Command Syntax

```
:MARKer:MODE <mode>

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

The :MARKer:MODE command sets the cursors mode:

- OFF – removes the cursor information from the display.
- MANual – enables manual placement of the X and Y cursors.

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

  Setting the mode to MEASurement sets the marker sources (:MARKer:X1Y1source and :MARKer:X2Y2source) to the measurement source (:MEASure:SOURce). Setting the measurement source remotely always sets the marker sources.
- WAVeform – the Y1 cursor tracks the voltage value at the X1 cursor of the waveform specified by the X1Y1source, and the Y2 cursor does the same for the X2 cursor and its X2Y2source.
- BINary – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in binary.
- HEX – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in hexadecimal.

Query Syntax

```
:MARKer:MODE?
```

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

Return Format

```
<mode><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428

:MARKer:X1:DISPlay

(see page 1354)

Command Syntax `:MARKer:X1:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:X1:DISPlay command specifies whether the X1 cursor is displayed.

Query Syntax `:MARKer:X1:DISPlay?`

The :MARKer:X1:DISPlay? query returns the X1 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:X1:DISPlay" on page 416

:MARKer:X1Position

(see page 1354)

Command Syntax

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

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

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

The :MARKer:X1Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVeform (see ":MARKer:MODE" on page 415).
- Sets the X1 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the current X1 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

NOTE If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X2Position" on page 420
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:XUNits" on page 423
- ":MEASure:TSTArt" on page 1273

:MARKer:X1Y1source

N (see page 1354)

Command Syntax

```
:MARKer:X1Y1source <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

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

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

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

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 415):

- Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X1Y1) sets the source for the other (for example, X2Y2).

If the marker mode is currently WAVeform, the X1Y1 source can be set separate from the X2Y2 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

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

Query Syntax

```
:MARKer:X1Y1source?
```

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

Return Format

```
<source><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494

:MARKer:X2:DISPlay

N (see page 1354)

Command Syntax `:MARKer:X2:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:X2:DISPlay command specifies whether the X2 cursor is displayed.

Query Syntax `:MARKer:X2:DISPlay?`

The :MARKer:X2:DISPlay? query returns the X2 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:X2:DISPlay" on page 419

:MARKer:X2Position

(see page 1354)

Command Syntax

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

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

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

The :MARKer:X2Position command:

- Sets :MARKer:MODE to MANual if it is not currently set to WAVeform (see ":MARKer:MODE" on page 415).
- Sets the X2 cursor position to the specified value.

X cursor units are set by the :MARKer:XUNits command.

Query Syntax

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns current X2 cursor position. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

NOTE If the front-panel cursors are off, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Y2source" on page 421
- ":MARKer:XUNits" on page 423
- ":MEASure:TSTOp" on page 1274

:MARKer:X2Y2source

N (see page 1354)

Command Syntax

```
:MARKer:X2Y2source <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

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

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

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

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVeform (see ":MARKer:MODE" on page 415):

- Sending a :MARKer:X2Y2source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X2Y2) sets the source for the other (for example, X1Y1).

If the marker mode is currently WAVeform, the X2Y2 source can be set separate from the X1Y1 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

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

Query Syntax

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Y1source" on page 418
- ":MEASure:SOURce" on page 494

:MARKer:XDELta

(see page 1354)

Query Syntax `:MARKer:XDELta?`

The MARKer:XDELta? query returns the value difference between the current X1 and X2 cursor positions.

Xdelta = (Value at X2 cursor) - (Value at X1 cursor)

X cursor units are set by the :MARKer:XUNits command.

NOTE If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format `<value><NL>`

`<value> ::= difference value in NR3 format.`

See Also
- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:XUNits" on page 423

:MARKer:XUNits

(see page 1354)

Command Syntax

```
:MARKer:XUNits <units>

<units> ::= {SEConds | HERTz | DEGRees | PERCent}
```

The :MARKer:XUNits command sets the X cursors units:

- SEConds – for making time measurements.
- HERTz – for making frequency measurements.
- DEGRees – for making phase measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 degrees and the current X2 location as 360 degrees.
- PERCent – for making ratio measurements. Use the :MARKer:XUNits:USE command to set the current X1 location as 0 percent and the current X2 location as 100 percent.

Changing X units affects the input and output values of the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries.

Query Syntax

```
:MARKer:XUNits?
```

The :MARKer:XUNits? query returns the current X cursors units.

Return Format

```
<units><NL>

<units> ::= {SEC | HERT | DEGR | PERC}
```

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:XUNits:USE" on page 424
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420

:MARKer:XUNits:USE

 (see page 1354)

Command Syntax `:MARKer:XUNits:USE`

When DEGRees is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 degrees and the current X2 location as 360 degrees.

When PERCent is selected for :MARKer:XUNits, the :MARKer:XUNits:USE command sets the current X1 location as 0 percent and the current X2 location as 100 percent.

Once the 0 and 360 degree or 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:X1Position, :MARKer:X2Position, and :MARKer:XDELta commands/queries are relative to the set locations.

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:XUNits" on page 423
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:XDELta" on page 422

:MARKer:Y1:DISPlay

 (see page 1354)

Command Syntax `:MARKer:Y1:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:Y1:DISPlay command specifies whether the Y1 cursor is displayed.

Query Syntax `:MARKer:Y1:DISPlay?`

The :MARKer:Y1:DISPlay? query returns the Y1 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:Y1:DISPlay" on page 425

:MARKer:Y1Position

(see page 1354)

Command Syntax

```
:MARKer:Y1Position <position> [suffix]

<position> ::= Y1 cursor position in NR3 format

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

If the :MARKer:MODE is not currently set to WAVeform (see ":MARKer:MODE" on page 415), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y1 cursor position to the specified value.

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

Query Syntax

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns current Y1 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query.

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

<position> ::= Y1 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:Y2Position" on page 428
- ":MARKer:YUNits" on page 430
- ":MEASure:VSTArt" on page 1278

:MARKer:Y2:DISPlay

N (see page 1354)

Command Syntax `:MARKer:Y2:DISPlay {{0 | OFF} | {1 | ON}}`

The :MARKer:Y2:DISPlay command specifies whether the Y2 cursor is displayed.

Query Syntax `:MARKer:Y2:DISPlay?`

The :MARKer:Y2:DISPlay? query returns the Y2 cursor display setting.

Return Format `<setting><NL>`

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

See Also

- ":MARKer:Y2:DISPlay" on page 427

:MARKer:Y2Position

(see page 1354)

Command Syntax

```
:MARKer:Y2Position <position> [suffix]

<position> ::= Y2 cursor position in NR3 format

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

If the :MARKer:MODE is not currently set to WAVeform (see ":MARKer:MODE" on page 415), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y2 cursor position to the specified value.

Y cursor units are set by the :MARKer:YUNits command.

When the :MARKer:MODE is set to WAVeform, Y positions cannot be set.

Query Syntax

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns current Y2 cursor position. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined and an error is generated. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

Return Format

```
<position><NL>

<position> ::= Y2 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:Y1Position" on page 426
- ":MARKer:YUNits" on page 430
- ":MEASure:VSTOp" on page 1279

:MARKer:YDELta

(see page 1354)

Query Syntax `:MARKer:YDELta?`

The :MARKer:YDELta? query returns the value difference between the current Y1 and Y2 cursor positions.

Ydelta = (Value at Y2 cursor) - (Value at Y1 cursor)

NOTE **If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.**

Y cursor units are set by the :MARKer:YUNits command.

Return Format `<value><NL>`

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

See Also
- "Introduction to :MARKer Commands" on page 413
- ":MARKer:MODE" on page 415
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YUNits" on page 430

:MARKer:YUNits

 (see page 1354)

Command Syntax

```
:MARKer:YUNits <units>

<units> ::= {BASE | PERCent}
```

The :MARKer:YUNits command sets the Y cursors units:

- BASE – for making measurements in the units associated with the cursors source.
- PERCent – for making ratio measurements. Use the :MARKer:YUNits:USE command to set the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Changing Y units affects the input and output values of the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries.

Query Syntax

```
:MARKer:YUNits?
```

The :MARKer:YUNits? query returns the current Y cursors units.

Return Format

```
<units><NL>

<units> ::= {BASE | PERC}
```

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:YUNits:USE" on page 431
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YDELta" on page 429

:MARKer:YUNits:USE

 (see page 1354)

Command Syntax `:MARKer:YUNits:USE`

When PERCent is selected for :MARKer:YUNits, the :MARKer:YUNits:USE command sets the current Y1 location as 0 percent and the current Y2 location as 100 percent.

Once the 0 and 100 percent locations are set, inputs to and outputs from the :MARKer:Y1Position, :MARKer:Y2Position, and :MARKer:YDELta commands/queries are relative to the set locations.

See Also

- "Introduction to :MARKer Commands" on page 413
- ":MARKer:YUNits" on page 430
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YDELta" on page 429

21 :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 448.

Table 96 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 451)	n/a	n/a
:MEASure:AREa [<interval>][,<source>] (see page 452)	:MEASure:AREa? [<interval>][,<source>] (see page 452)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BRATe [<source>] (see page 453)	:MEASure:BRATe? [<source>] (see page 453)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# of analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= bit rate in Hz, NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:BWIDth [<source>] (see page 454)	:MEASure:BWIDth? [<source>] (see page 454)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 455)	n/a	n/a
:MEASure:COUNter [<source>] (see page 456)	:MEASure:COUNter? [<source>] (see page 456)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 457)	:MEASure:DEFine? DELay (see page 459)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 457)	:MEASure:DEFine? THResholds (see page 459)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent | ABSolute}

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DELay [<source1>] [,<source2>] (see page 460)	:MEASure:DELay? [<source1>] [,<source2>] (see page 460)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>, <source1_edge_threshold>, <source2_edge_slope>, <source2_edge_number>, <source2_edge_threshold> (see page 462)	:MEASure:DELay:DEFine? (see page 462)	<source1_edge_slope>, <source2_edge_slope> ::= {RISing | FALLing} <source1_edge_number>, <source2_edge_number> ::= 0 to 1000 in NR1 format <source1_edge_threshold>, <source2_edge_threshold> ::= MIDDle
:MEASure:DUAL:CHARge [<interval>] [,<source1>] [,<source2>] (see page 463)	:MEASure:DUAL:CHARge? [<interval>] [,<source1>] [,<source2>] (see page 463)	<interval> ::= {CYCLe | DISPlay} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= area in Amp-hours, NR3 format
:MEASure:DUAL:VAMPlitude [<source1>] [,<source2>] (see page 464)	:MEASure:DUAL:VAMPlitude? [<source1>] [,<source2>] (see page 464)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUAL:VAVerage [<interval>] [,<source1>] [,<source2>] (see page 465)	:MEASure:DUAL:VAVerage? [<interval>] [,<source1>] [,<source2>] (see page 465)	<interval> ::= {CYCLe | DISPlay} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:DUAL:VBASe [<source1>] [,<source2>] (see page 466)	:MEASure:DUAL:VBASe? [<source1>] [,<source2>] (see page 466)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:DUAL:VPP [<source1>] [,<source2>] (see page 467)	:MEASure:DUAL:VPP? [<source1>] [,<source2>] (see page 467)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:DUAL:VRMS [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 468)	:MEASure:DUAL:VRMS? [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 468)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated RMS voltage in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 469)	:MEASure:DUTYcycle? [<source>] (see page 469)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 470)	:MEASure:FALLtime? [<source>] (see page 470)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FFT:ACPR <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 471)	:MEASure:FFT:ACPR? <chan_width>, <chan_spacing>, <chan>[,<source>] (see page 471)	<chan_width> ::= width of main range and sideband channels, Hz in NR3 format <chan_spacing> ::= spacing between main range and sideband channels, Hz in NR3 format <chan> ::= {CENTer | HIGH<sb> | LOW<sb>} <sb> ::= sideband 1 to 5 <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= adjacent channel power ratio, dBV in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FFT:CPOWer [<source>] (see page 472)	:MEASure:FFT:CPOWer? [<source>] (see page 472)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= spectral channel power, dBV in NR3 format
:MEASure:FFT:OBW <percentage>[,<source>] (see page 473)	:MEASure:FFT:OBW? <percentage>[,<source>] (see page 473)	<percentage> ::= percent of spectral power occupied bandwidth is measured for in NR3 format <source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= occupied bandwidth, Hz in NR3 format
:MEASure:FFT:THD [<source>] (see page 474)	:MEASure:FFT:THD? [<source>] (see page 474)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= total harmonic distortion ratio percent in NR3 format
:MEASure:FREQuency [<source>] (see page 475)	:MEASure:FREQuency? [<source>] (see page 475)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= frequency in Hertz in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NDUTy [<source>] (see page 476)	:MEASure:NDUTy? [<source>] (see page 476)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NEDGes [<source>] (see page 477)	:MEASure:NEDGes? [<source>] (see page 477)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling edge count in NR3 format
:MEASure:NPULses [<source>] (see page 478)	:MEASure:NPULses? [<source>] (see page 478)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the falling pulse count in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 479)	:MEASure:NWIDth? [<source>] (see page 479)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 480)	:MEASure:OVERshoot? [<source>] (see page 480)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PEDGes [<source>] (see page 482)	:MEASure:PEDGes? [<source>] (see page 482)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising edge count in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PERiod [<source>] (see page 483)	:MEASure:PERiod? [<source>] (see page 483)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 484)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 484)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PPULses [<source>] (see page 485)	:MEASure:PPULses? [<source>] (see page 485)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising pulse count in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PREShoot [<source>] (see page 486)	:MEASure:PREShoot? [<source>] (see page 486)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 487)	:MEASure:PWIDth? [<source>] (see page 487)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 488)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 491)	:MEASure:RISetime? [<source>] (see page 491)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SDEViation [<source>] (see page 492)	:MEASure:SDEViation? [<source>] (see page 492)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 | OFF} | {1 | ON}} (see page 493)	:MEASure:SHOW? (see page 493)	{0 | 1}
:MEASure:SOURce <source1> [,<source2>] (see page 494)	:MEASure:SOURce? (see page 494)	<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 496)	:MEASure:STATistics? (see page 496)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 497)	:MEASure:STATistics:DISPlay? (see page 497)	{0 | 1}
:MEASure:STATistics:INCRement (see page 498)	n/a	n/a
:MEASure:STATistics:MCOunt <setting> (see page 499)	:MEASure:STATistics:MCOunt? (see page 499)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:STATistics:RESet (see page 500)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 501)	:MEASure:STATistics:RSDeviation? (see page 501)	{0 | 1}
n/a	:MEASure:TEDGe? <slope><occurrence>[,<source>] (see page 502)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>,[<slope>]<occurrence>[,<source>] (see page 504)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VAMPlitude [<source>] (see page 506)	:MEASure:VAMPlitude? [<source>] (see page 506)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<interval>][,<source>] (see page 507)	:MEASure:VAVerage? [<interval>][,<source>] (see page 507)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 508)	:MEASure:VBASe? [<source>] (see page 508)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 509)	:MEASure:VMAX? [<source>] (see page 509)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 510)	:MEASure:VMIN? [<source>] (see page 510)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 511)	:MEASure:VPP? [<source>] (see page 511)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRATio [<interval>] [,<source 1>] [,<source2>] (see page 512)	:MEASure:VRATio? [<interval>] [,<source 1>] [,<source2>] (see page 512)	<interval> ::= {CYCLe | DISPlay} <source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>] [,<type>] [,<source>] (see page 513)	:MEASure:VRMS? [<interval>] [,<type>] [,<source>] (see page 513)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 514)	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 515)	:MEASure:VTOP? [<source>] (see page 515)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 516)	:MEASure:WINDow? (see page 516)	<type> ::= {MAIN | ZOOM | AUTO | GATE}
:MEASure:XMAX [<source>] (see page 517)	:MEASure:XMAX? [<source>] (see page 517)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 518)	:MEASure:XMIN? [<source>] (see page 518)	<source> ::= {CHANnel<n> | FUNCtion<m> | FFT | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format

Introduction to :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the zoomed (delayed) time base mode (:TIMebase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the zoomed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on FFT (Fast Fourier Transform).

Reporting the Setup

Use the :MEASure? query to obtain setup information for the MEASure subsystem. (Currently, this is only :MEASure:SOURce.)

Return Format

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

```
:MEAS:SOUR CHAN1,CHAN2;STAT ON
```

:MEASure:ALL

 (see page 1354)

Command Syntax `:MEASure:ALL`

This command installs a Snapshot All measurement on the screen.

See Also

- "Introduction to :MEASure Commands" on page 448

:MEASure:AREa

(see page 1354)

Command Syntax

```
:MEASure:AREa [<interval>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:AREa command installs an area measurement on screen. Area measurements show the area between the waveform and the ground level.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:AREa? [<interval>][,<source>]
```

The :MEASure:AREa? query measures and returns the area value.

Return Format

```
<value><NL>

<value> ::= the area value in volt-seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:BRATe

N (see page 1354)

Command Syntax

```
:MEASure:BRATe [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:BRATe command installs a screen measurement and starts the bit rate measurement. If the optional source parameter is specified, the currently specified source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:BRATe? [<source>]
```

The :MEASure:BRATe? query measures all positive and negative pulse widths on the waveform, takes the minimum value found of either width type and inverts that minimum width to give a value in Hertz.

Return Format

```
<value><NL>

<value> ::= the bit rate value in Hertz
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:FREQuency" on page 475
- ":MEASure:PERiod" on page 483

:MEASure:BWIDth

(see page 1354)

Command Syntax

```
:MEASure:BWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:BWIDth command installs a burst width measurement on screen. If the optional source parameter is not specified, the current measurement source is used.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:BWIDth? [<source>]
```

The :MEASure:BWIDth? query measures and returns the width of the burst on the screen.

The burst width is calculated as follows:

burst width = (last edge on screen - first edge on screen)

Return Format

```
<value><NL>

<value> ::= burst width in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:CLEar

 (see page 1354)

Command Syntax `:MEASure:CLEar`

This command clears all selected measurements and markers from the screen.

See Also

- "Introduction to :MEASure Commands" on page 448

:MEASure:COUNter

N (see page 1354)

Command Syntax

```
:MEASure:COUNter [<source>]

<source> ::= {CHANnel<n> | EXTernal}

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

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math or Reference Waveforms may be selected for the source.

The counter measurement counts trigger level crossings within a certain amount of time (gate time) and displays the results in Hz.

The gate time is the horizontal range of the oscilloscope but is limited to >= 0.1 s and <= 10 s. Unlike other measurements, the Zoom horizontal timebase window does not gate the Counter measurement.

The Counter measurement can measure frequencies up to the bandwidth of the oscilloscope. The minimum frequency supported is 2.0 / gateTime.

Only one counter measurement may be displayed at a time.

NOTE This command is not available if the source is MATH.

Query Syntax

```
:MEASure:COUNter? [<source>]
```

The :MEASure:COUNter? query measures and outputs the counter frequency of the specified source.

NOTE The :MEASure:COUNter? query times out if the counter measurement is installed on the front panel. Use :MEASure:CLEar to remove the front-panel measurement before executing the :MEASure:COUNter? query.

Return Format

```
<source><NL>

<source> ::= count in Hertz in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:FREQuency" on page 475
- ":MEASure:CLEar" on page 455

:MEASure:DEFine

(see page 1354)

Command Syntax

```
:MEASure:DEFine <meas_spec>[,<source>]

<meas_spec> ::= {DELay | THResholds}, for remaining syntax, see:
```

- ":MEASure:DEFine DELay Command Syntax" on page 457
- ":MEASure:DEFine THResholds Command Syntax" on page 458

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DELay	THResholds
DUTYcycle		x
DELay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PREShoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

:MEASure:DEFine DELay Command Syntax

NOTE

The DELay portion of the MEASure:DEFine command has been deprecated, and you should instead use the :MEASure:DELay:DEFine command to specify delay measurement parameters. The delay measurement now allows an edge occurrence of zero (0) to specify automatic edge selection. The limitation of this command is that you cannot specify a negative slope with an occurrence count of zero (0).

```
:MEASure:DEFine DELay,<delay spec>[,<source>]

<delay spec> ::= <edge_spec1>,<edge_spec2>

<edge_spec1> ::= [<slope>]<occurrence>

<edge_spec2> ::= [<slope>]<occurrence>

<slope> ::= {+ | -}

<occurrence> ::= integer

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

This command defines the behavior of the :MEASure:DELay command/query by specifying the start and stop edge to be used. <edge_spec1> specifies the slope and edge number on source1. <edge_spec2> specifies the slope and edge number on source2. The measurement is taken as:

delay = t(<edge_spec2>) - t(<edge_spec1>)

:MEASure:DEFine THResholds Command Syntax

```
:MEASure:DEFine THResholds,<threshold spec>[,<source>]

<threshold spec> ::= {STANdard}
                     | {<threshold mode>,<upper>,<middle>,<lower>}

<threshold mode> ::= {PERCent | ABSolute}
```

for <threshold mode> = PERCent:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <threshold mode> = ABSolute:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold absolute values in
                               NR3 format.

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.

- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGe or :CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

Query Syntax

```
:MEASure:DEFine? <meas_spec>[,<source>]

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

Return Format

for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <meas_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold voltages in NR3
                               format.
```

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:DELay" on page 460
- ":MEASure:DELay:DEFine" on page 462
- ":MEASure:SOURce" on page 494
- ":CHANnel<n>:RANGe" on page 291
- ":CHANnel<n>:SCALe" on page 292
- ":CHANnel<n>:PROBe" on page 282
- ":CHANnel<n>:UNITs" on page 293

:MEASure:DELay

(see page 1354)

Command Syntax

```
:MEASure:DELay [<edge_select_mode>][,][<source1>][,<source2>]

<edge_select_mode> ::= {MANual | AUTO}

<source1>, <source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r
>}

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

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

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

The :MEASure:DELay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

delay = t(<edge_spec_2>) - t(<edge_spec_1>)

where the <edge_spec> definitions are determined by the <edge_select_mode> and the :MEASure:DELay:DEFine command.

When the <edge_select_mode> is:

- AUTO – edges are automatically selected: the source1 edge closest to the timebase reference point is used, and the source2 edge closest to the source1 edge is used.
- MANual – edge numbers are determined by the :MEASure:DELay:DEFine settings.

  Edge numbers greater than zero (0) are counted from the left side of the display for both sources.

Edge slopes are always determined by the :MEASure:DELay:DEFine settings.

If the <edge_select_mode> is not included with the command, AUTO is the default.

Query Syntax

```
:MEASure:DELay? [<edge_select_mode>][,][<source1>][,<source2>]
```

The :MEASure:DELay? query measures and returns the delay between source1 and source2. Delay measurement slope and edge count parameters are determined by the <edge_select_mode> and the :MEASure:DELay:DEFine command.

The <edge_select_mode> definitions are the same as with the command syntax.

However, if the <edge_select_mode> is not included with the query, MANual is the default.

In the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%,

50%, and 10% values between Vbase and Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THResholds command.

Return Format

```
<value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:DELay:DEFine" on page 462
- ":MEASure:DEFine" on page 457
- ":MEASure:PHASe" on page 484

:MEASure:DELay:DEFine

N (see page 1354)

Command Syntax

```
:MEASure:DELay:DEFine <source1_edge_slope>, <source1_edge_number>,
   <source1_edge_threshold>, <source2_edge_slope>,
   <source2_edge_number>, <source2_edge_threshold>

<source1_edge_slope>,
<source2_edge_slope> ::= {RISing | FALLing}

<source1_edge_number>,
<source2_edge_number> ::= 0 to 1000 in NR1 format

<source1_edge_threshold>,
<source2_edge_threshold> ::= MIDDle
```

The :MEASure:DELay:DEFine command defines slope directions and edge numbers for the delay measurement started or returned by the :MEASure:DELay command. The :MEASure:DELay:DEFine command also defines edge position parameters, but currently, MIDDle is the only valid selection.

A <source1_edge_number> setting of zero (0) specifies that the edge closest to the timebase reference point automatically be selected. In this case, the <source2_edge_number> setting must also be zero (0), and the source2 edge closest to the selected source1 edge is used.

When edge numbers greater than zero (0) are specified, edges are counted from the left side of the display for both sources.

The selection of the source1 and source2 waveforms is made in the :MEASure:DELay or :MEASure:SOURce commands.

Query Syntax

```
:MEASure:DELay:DEFine?
```

The :MEASure:DELay:DEFine? query returns the specified delay measurement parameter definitions.

Return Format

```
<source1_edge_slope>, <source1_edge_number>,
   <source1_edge_threshold>, <source2_edge_slope>,
   <source2_edge_number>,<source2_edge_threshold><NL>

<source1_edge_slope>,
<source2_edge_slope> ::= {RIS | FALL}

<source1_edge_number>,
<source2_edge_number> ::= 0 to 1000 in NR1 format

<source1_edge_threshold>,
<source2_edge_threshold> ::= MIDD
```

See Also

- ":MEASure:DELay" on page 460
- ":MEASure:SOURce" on page 494
- ":MEASure:DEFine" on page 457

:MEASure:DUAL:CHARge

(see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:CHARge [<interval>][,<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:CHARge command installs a charge measurement on screen. Charge measurements show the area between the waveform and the ground level.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:CHARge? [<interval>][,<source1>][,<source2>]
```

The :MEASure:DUAL:CHARge? query measures and returns the charge measurement value.

Return Format

```
<value><NL>

<value> ::= the charge value in Amp-hours in NR3 format
```

See Also

- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:DUAL:VAMPlitude

N (see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VAMPlitude [<source1>][,<source2>]

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VAMPlitude? [<source1>][,<source2>]
```

The :MEASure:DUAL:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

vertical amplitude = Vtop - Vbase

Return Format

```
<value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format
```

See Also

- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VTOP" on page 515

:MEASure:DUAL:VAVerage

N (see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VAVerage [<interval>][,<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:VAVerage command installs a screen measurement and starts an average value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VAVerage? [<interval>][,<source1>][,<source2>]
```

The :MEASure:DUAL:VAVerage? query returns the average value measurement.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:DUAL:VBASe

(see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VBASe [<source1>][,<source2>]

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:VBASe command installs a screen measurement and starts a waveform base value measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VBASe? [<source1>][,<source2>]
```

The :MEASure:DUAL:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

```
<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in
                   NR3 format
```

See Also

- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VPP" on page 467
- ":MEASure:DUAL:VRMS" on page 468
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VTOP" on page 515
- ":MEASure:VMIN" on page 510

:MEASure:DUAL:VPP

N (see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VPP [<source1>][,<source2>]

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VPP? [<source1>][,<source2>]
```

The :MEASure:DUAL:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

Vpp = Vmax - Vmin

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format

```
<value><NL>

<value> ::= vertical peak to peak value in NR3 format
```

See Also

- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VRMS" on page 468
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VMAX" on page 509
- ":MEASure:VMIN" on page 510

:MEASure:DUAL:VRMS

N (see page 1354)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VRMS [<interval>][,<type>][,<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<type> ::= {AC | DC}

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected

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

The :MEASure:DUAL:VRMS command installs a screen measurement and starts an RMS value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VRMS? [<interval>][,<type>][,<source1>][,<source2>]
```

The :MEASure:DUAL:VRMS? query measures and outputs the RMS value measurement.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- ":MEASure:DUAL:CHARge" on page 463
- ":MEASure:DUAL:VAMPlitude" on page 464
- ":MEASure:DUAL:VAVerage" on page 465
- ":MEASure:DUAL:VBASe" on page 466
- ":MEASure:DUAL:VPP" on page 467
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:DUTYcycle

(see page 1354)

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:DUTYcycle command installs a screen measurement and starts a positive duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the positive duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

+duty cycle = (+pulse width/period)*100

Return Format

```
<value><NL>

<value> ::= ratio of positive pulse width to period in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:PERiod" on page 483
- ":MEASure:PWIDth" on page 487
- ":MEASure:SOURce" on page 494
- ":MEASure:NDUTy" on page 476

Example Code

- "Example Code" on page 495

:MEASure:FALLtime

(see page 1354)

Command Syntax

```
:MEASure:FALLtime [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FALLtime? [<source>]
```

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

fall time = time at lower threshold - time at upper threshold

Return Format

```
<value><NL>

<value> ::= time in seconds between the lower threshold and upper
            threshold in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:RISetime" on page 491
- ":MEASure:SOURce" on page 494

:MEASure:FFT:ACPR

N (see page 1354)

Command Syntax

```
:MEASure:FFT:ACPR <chan_width>,<chan_spacing>,<chan>[,<source>]

<chan_width> ::= of main range and sideband channels, Hz in NR3 format

<chan_spacing> ::= spacing between main range and sideband channels,
   Hz in NR3 format

<chan> ::= {CENTer | HIGH<sb> | LOW<sb>}

<sb> ::= sideband 1 to 5

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

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

The :MEASure:FFT:ACPR command installs an FFT analysis Adjacent Channel Power Ratio (ACPR) measurement on screen.

Adjacent Channel Power Ratio (or channel leakage ratio) measures the ratio of the power in the main frequency range to the power contained in one or more sidebands.

The main range is specified by a channel width and a center frequency. The center frequency used in the measurement is the one defined for the FFT function.

Sidebands (with the same width as the main range) exist above and below the main range separated by the channel spacing width.

The sideband used for the measurement is selected with the <chan> parameter. You can select the first through fifth sidebands above or below the main range (HIGH1 through HIGH5 above and LOW1 through LOW5 below). The full sideband must be in the graticule (on screen) to be measured. Otherwise, the measurement results will be "Incomplete".

When this measurement is tracked with cursors, the cursors show the sideband being measured.

Query Syntax

```
:MEASure:FFT:ACPR? <chan_width>,<chan_spacing>,<chan>[,<source>]
```

The :MEASure:FFT:ACPR? query returns the measured Adjacent Channel Power Ratio (ACPR) value.

Return Format

```
<return_value><NL>

<return_value> ::= adjacent channel power ratio, dBV in NR3 format
```

See Also

- ":MEASure:FFT:CPOWer" on page 472
- ":MEASure:FFT:OBW" on page 473
- ":MEASure:FFT:THD" on page 474

:MEASure:FFT:CPOWer

 (see page 1354)

Command Syntax

```
:MEASure:FFT:CPOWer [<source>]

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

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

The :MEASure:FFT:CPOWer command installs an FFT analysis Channel Power measurement on screen.

Channel Power measures the spectral power across a frequency range.

The center frequency used in the measurement is the one defined for the FFT function, and the FFT span specifies the frequency range.

When this measurement is tracked with cursors, the cursors are at the left and right edges of the frequency span.

Query Syntax

```
:MEASure:FFT:CPOWer? [<source>]
```

The :MEASure:FFT:CPOWer? query returns the measured Channel Power value.

Return Format

```
<return_value><NL>

<return_value> ::= spectral channel power, dBV in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 471
- ":MEASure:FFT:OBW" on page 473
- ":MEASure:FFT:THD" on page 474

:MEASure:FFT:OBW

N (see page 1354)

Command Syntax

```
:MEASure:FFT:OBW <percentage>[,<source>]

<percentage> ::= percent of spectral power occupied bandwidth is
   measured for (in NR3 format)

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

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

The :MEASure:FFT:OBW command installs an FFT analysis Occupied Bandwidth measurement on screen.

Occupied Bandwidth measures the bandwidth (frequency range) containing some percent (usually 99%) of the total spectral power. While 99% is the industry norm, you can specify the percent you want to use in the measurement.

The center frequency used in the measurement is the one defined for the FFT function, and the FFT span represents the total spectral power.

When this measurement is tracked with cursors, the cursors show the measured bandwidth (frequency range).

Query Syntax

```
:MEASure:FFT:OBW? <percentage>[,<source>]
```

The :MEASure:FFT:OBW? query returns the measured Occupied Bandwidth value.

Return Format

```
<return_value><NL>

<return_value> ::= occupied bandwidth, Hz in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 471
- ":MEASure:FFT:CPOWer" on page 472
- ":MEASure:FFT:THD" on page 474

:MEASure:FFT:THD

N (see page 1354)

Command Syntax

```
:MEASure:FFT:THD <tracking>[,<fundamental_freq>][,<source>]

<tracking> ::= {AUTO | MANual}

<fundamental_freq> ::= in NR3 format, required if <tracking> is MANual

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform)

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

The :MEASure:FFT:THD command installs an FFT analysis Total Harmonic Distortion measurement on screen.

Total Harmonic Distortion (THD) is the ratio of power in the fundamental frequency to the power contained in the rest of the harmonics and noise. THD is a measure of signal purity.

Total Harmonic Distortion (THD) measures the power contained in the bands surrounding each harmonic and compares it to the power in the band surrounding the fundamental frequency. The width of the bands measured is the same for the fundamental frequency and each harmonic. That width is 1/2 of the fundamental frequency.

You can either enter the fundamental frequency as a measurement parameter and have the fundamental frequency and harmonics be tracked manually, or you can allow the fundamental frequency and harmonics to be tracked automatically, where the highest peak is assumed to be the fundamental frequency.

When this measurement is tracked with cursors, the cursors show the band surrounding the fundamental frequency that is being measured (at ±1/4 of the fundamental frequency).

Query Syntax

```
:MEASure:FFT:THD? <tracking>[,<fundamental_freq>][,<source>]
```

The :MEASure:FFT:THD? query returns the measured Total Harmonic Distortion value.

Return Format

```
<return_value><NL>

<return_value> ::= total harmonic distortion ratio percent in NR3 format
```

See Also

- ":MEASure:FFT:ACPR" on page 471
- ":MEASure:FFT:CPOWer" on page 472
- ":MEASure:FFT:OBW" on page 473

:MEASure:FREQuency

(see page 1354)

Command Syntax

```
:MEASure:FREQuency [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:FREQuency command installs a screen measurement and starts a frequency measurement. If the optional source parameter is specified, the current source is modified.

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format

```
<source><NL>

<source> ::= frequency in Hertz in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:PERiod" on page 483

Example Code

- "Example Code" on page 495

:MEASure:NDUTy

N (see page 1354)

Command Syntax

```
:MEASure:NDUTy [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:NDUTy command installs a screen measurement and starts a negative duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NDUTy? [<source>]
```

The :MEASure:NDUTy? query measures and outputs the negative duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the negative pulse width to the period. The negative pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

-duty cycle = (-pulse width/period)*100

Return Format

```
<value><NL>

<value> ::= ratio of negative pulse width to period in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:PERiod" on page 483
- ":MEASure:NWIDth" on page 479
- ":MEASure:SOURce" on page 494
- ":MEASure:DUTYcycle" on page 469

:MEASure:NEDGes

(see page 1354)

Command Syntax

```
:MEASure:NEDGes [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:NEDGes command installs a falling edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NEDGes? [<source>]
```

The :MEASure:NEDGes? query measures and returns the on-screen falling edge count.

Return Format

```
<value><NL>

<value> ::= the falling edge count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:NPULses

(see page 1354)

Command Syntax

```
:MEASure:NPULses [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:NPULses command installs a falling pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NPULses? [<source>]
```

The :MEASure:NPULses? query measures and returns the on-screen falling pulse count.

Return Format

```
<value><NL>

<value> ::= the falling pulse count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:NWIDth

(see page 1354)

Command Syntax

```
:MEASure:NWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is not specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

Return Format

```
<value><NL>

<value> ::= negative pulse width in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:PWIDth" on page 487
- ":MEASure:PERiod" on page 483

:MEASure:OVERshoot

(see page 1354)

Command Syntax

```
:MEASure:OVERshoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

overshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

For a falling edge:

overshoot = ((Vbase-Vmin) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right after the chosen edge, halfway to the next edge. This more restricted definition is used instead of the normal one, because it is conceivable that a signal may have more preshoot than overshoot, and the normal extremum would then be dominated by the preshoot of the following edge.

Return Format

```
<overshoot><NL>

<overshoot>::= the percent of the overshoot of the selected waveform in
NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:PREShoot" on page 486
- ":MEASure:SOURce" on page 494

- ":MEASure:VMAX" on page 509
- ":MEASure:VTOP" on page 515
- ":MEASure:VBASe" on page 508
- ":MEASure:VMIN" on page 510

:MEASure:PEDGes

 (see page 1354)

Command Syntax

```
:MEASure:PEDGes [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:PEDGes command installs a rising edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PEDGes? [<source>]
```

The :MEASure:NEDGes? query measures and returns the on-screen rising edge count.

Return Format

```
<value><NL>

<value> ::= the rising edge count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:PERiod

(see page 1354)

Command Syntax

```
:MEASure:PERiod [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:PERiod command installs a screen measurement and starts the period measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

Return Format

```
<value><NL>

<value> ::= waveform period in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:NWIDth" on page 479
- ":MEASure:PWIDth" on page 487
- ":MEASure:FREQuency" on page 475

Example Code

- "Example Code" on page 495

:MEASure:PHASe

 (see page 1354)

Command Syntax

```
:MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r
>}

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

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

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

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax

```
:MEASure:PHASe? [<source1>][,<source2>]
```

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

phase = (delay / period of input 1) x 360

Return Format

```
<value><NL>

<value> ::= the phase angle value in degrees in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:DELay" on page 460
- ":MEASure:PERiod" on page 483
- ":MEASure:SOURce" on page 494

:MEASure:PPULses

(see page 1354)

Command Syntax

```
:MEASure:PPULses [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:PPULses command installs a rising pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PPULses? [<source>]
```

The :MEASure:PPULses? query measures and returns the on-screen rising pulse count.

Return Format

```
<value><NL>

<value> ::= the rising pulse count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:PREShoot

(see page 1354)

Command Syntax

```
:MEASure:PREShoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:PREShoot? [<source>]
```

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

preshoot = ((Vmin-Vbase) / (Vtop-Vbase)) x 100

For a falling edge:

preshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right before the chosen edge, halfway back to the previous edge. This more restricted definition is used instead of the normal one, because it is likely that a signal may have more overshoot than preshoot, and the normal extremum would then be dominated by the overshoot of the preceding edge.

Return Format

```
<value><NL>

<value> ::= the percent of preshoot of the selected waveform
            in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VMIN" on page 510
- ":MEASure:VMAX" on page 509
- ":MEASure:VTOP" on page 515
- ":MEASure:VBASe" on page 508

:MEASure:PWIDth

(see page 1354)

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

```
<value><NL>

<value> ::= width of positive pulse in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:NWIDth" on page 479
- ":MEASure:PERiod" on page 483

:MEASure:RESults

 (see page 1354)

Query Syntax

```
:MEASure:RESults?
```

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values.

If more than one measurement is running continuously, the :MEASure:RESults return values are duplicated for each continuous measurement from the first to last (top to bottom) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of 10 continuous measurements that can be continuously displayed at a time.

When no quick measurements are installed, the :MEASure:RESults? query returns nothing (empty string). When the count for any of the measurements is 0, the value of infinity (9.9E+37) is returned for the min, max, mean, and standard deviation.

Return Format

```
<result_list><NL>

<result_list> ::= comma-separated list of measurement results
```

The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measurement label	current	min	max	mean	std dev	count

Measurement label, current, min, max, mean, std dev, and count are only returned if :MEASure:STATistics is ON.

If :MEASure:STATistics is set to CURRent, MIN, MAX, MEAN, STDDev, or COUNt only that particular statistic value is returned for each measurement that is on.

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:STATistics" on page 496

Example Code

```
' This program shows the InfiniiVision oscilloscopes' measurement
' statistics commands.
' ----------------------------------------------------------------------

Option Explicit

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

```
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::10.112.94.136::hislip9-0.0::INSTR
")

  ' Initialize.
  myScope.IO.Clear   ' Clear the interface.
  myScope.WriteString "*RST"   ' Reset to the defaults.
  myScope.WriteString "*CLS"   ' Clear the status data structures.
  myScope.WriteString ":AUToscale"

  ' Install some measurements.
  myScope.WriteString ":MEASure:SOURce CHANnel1"   ' Input source.

  Dim MeasurementArray(3) As String
  MeasurementArray(0) = "FREQuency"
  MeasurementArray(1) = "DUTYcycle"
  MeasurementArray(2) = "VAMPlitude"
  MeasurementArray(3) = "VPP"
  Dim Measurement As Variant

  For Each Measurement In MeasurementArray
    myScope.WriteString ":MEASure:" + Measurement
    myScope.WriteString ":MEASure:" + Measurement + "?"
    varQueryResult = myScope.ReadNumber   ' Read measurement value.
    Debug.Print Measurement + ": " + FormatNumber(varQueryResult, 4)
  Next

  myScope.WriteString ":MEASure:STATistics:RESet"   ' Reset stats.
  Sleep 5000   ' Wait for 5 seconds.

  ' Select the statistics results type.
  Dim ResultsTypeArray(6) As String
  ResultsTypeArray(0) = "CURRent"
  ResultsTypeArray(1) = "MINimum"
  ResultsTypeArray(2) = "MAXimum"
  ResultsTypeArray(3) = "MEAN"
  ResultsTypeArray(4) = "STDDev"
  ResultsTypeArray(5) = "COUNt"
  ResultsTypeArray(6) = "ON"   ' All results.
  Dim ResultType As Variant

  Dim ResultsList()

  Dim ValueColumnArray(6) As String
  ValueColumnArray(0) = "Meas_Lbl"
  ValueColumnArray(1) = "Current"
  ValueColumnArray(2) = "Min"
  ValueColumnArray(3) = "Max"
  ValueColumnArray(4) = "Mean"
```

```
  ValueColumnArray(5) = "Std_Dev"
  ValueColumnArray(6) = "Count"
  Dim ValueColumn As Variant

  For Each ResultType In ResultsTypeArray
    myScope.WriteString ":MEASure:STATistics " + ResultType

    ' Get the statistics results.
    Dim intCounter As Integer
    intCounter = 0
    myScope.WriteString ":MEASure:RESults?"
    ResultsList() = myScope.ReadList

    For Each Measurement In MeasurementArray

      If ResultType = "ON" Then   ' All statistics.

        For Each ValueColumn In ValueColumnArray
          If VarType(ResultsList(intCounter)) <> vbString Then
            Debug.Print "Measure statistics result CH1, " + _
                Measurement + ", "; ValueColumn + ": " + _
                FormatNumber(ResultsList(intCounter), 4)

          Else   ' Result is a string (e.g., measurement label).
            Debug.Print "Measure statistics result CH1, " + _
                Measurement + ", "; ValueColumn + ": " + _
                ResultsList(intCounter)

          End If

          intCounter = intCounter + 1

        Next

      Else   ' Specific statistic (e.g., Current, Max, Min, etc.).

        Debug.Print "Measure statistics result CH1, " + _
            Measurement + ", "; ResultType + ": " + _
            FormatNumber(ResultsList(intCounter), 4)

        intCounter = intCounter + 1

      End If

    Next

  Next

  Exit Sub

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

End Sub
```

:MEASure:RISetime

(see page 1354)

Command Syntax

```
:MEASure:RISetime [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:RISetime? [<source>]
```

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

rise time = time at upper threshold - time at lower threshold

Return Format

```
<value><NL>

<value> ::= rise time in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:FALLtime" on page 470

:MEASure:SDEViation

(see page 1354)

Command Syntax

```
:MEASure:SDEViation [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1 to (# math functions) in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

NOTE This ":MEASure:VRMS DISPlay, AC" command is the preferred syntax for making standard deviation measurements.

The :MEASure:SDEViation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:SDEViation? [<source>]
```

The :MEASure:SDEViation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

Return Format

```
<value><NL>
<value> ::= calculated std deviation value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:VRMS" on page 513
- ":MEASure:SOURce" on page 494

:MEASure:SHOW

(see page 1354)

Command Syntax

```
:MEASure:SHOW <on_off>

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

The :MEASure:SHOW command enables markers for tracking measurements on the display.

Query Syntax

```
:MEASure:SHOW?
```

The :MEASure:SHOW? query returns the current state of the markers.

This can return OFF when :MARKer:MODE selects a mode other than MEASurement.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MARKer:MODE" on page 415

:MEASure:SOURce

(see page 1354)

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>
   | WMEMory<r>}

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

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

<r> ::= 1-2 in NR1 format
```

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion<m>, or MATH<m> will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 applies only to :MEASure:DELay and :MEASure:PHASe measurements.

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

NOTE FUNCtion or MATH (without the "<m>" math function number) is an alias for FUNCtion2 or MATH2. The query will return FUNC2 if the source is FUNCtion or MATH.

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= {CHAN<n> | FUNC<m> | WMEM<r> | NONE}
```

See Also:

- "Introduction to :MEASure Commands" on page 448
- ":MARKer:MODE" on page 415
- ":MARKer:X1Y1source" on page 418
- ":MARKer:X2Y2source" on page 421
- ":MEASure:DELay" on page 460

- ":MEASure:PHASe" on page 484

Example Code

```
' MEASURE - The commands in the MEASure subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString ":MEASure:SOURce CHANnel1"   ' Source to measure.
myScope.WriteString ":MEASure:FREQuency?"   ' Query for frequency.
varQueryResult = myScope.ReadNumber   ' Read frequency.
MsgBox "Frequency:" + vbCrLf _
      + FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASure:DUTYcycle?"   ' Query for duty cycle.
varQueryResult = myScope.ReadNumber   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf _
      + FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASure:RISetime?"   ' Query for risetime.
varQueryResult = myScope.ReadNumber   ' Read risetime.
MsgBox "Risetime:" + vbCrLf _
      + FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASure:VPP?"   ' Query for Pk to Pk voltage.
varQueryResult = myScope.ReadNumber   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASure:VMAX?"   ' Query for Vmax.
varQueryResult = myScope.ReadNumber   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:MEASure:STATistics

 (see page 1354)

Command Syntax

```
:MEASure:STATistics <type>

<type> ::= {{ON | 1} | CURRent | MINimum | MAXimum | MEAN | STDDev
           | COUNt}
```

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Query Syntax

```
:MEASure:STATistics?
```

The :MEASure:STATistics? query returns the current statistics mode.

Return Format

```
<type><NL>

<type> ::= {ON | CURR | MIN | MAX | MEAN | STDD | COUN}
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:RESults" on page 488
- ":MEASure:STATistics:DISPlay" on page 497
- ":MEASure:STATistics:RESet" on page 500
- ":MEASure:STATistics:INCRement" on page 498

Example Code

- "Example Code" on page 488

:MEASure:STATistics:DISPlay

 (see page 1354)

Command Syntax `:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}}`

The :MEASure:STATistics:DISPlay command disables or enables the display of the measurement statistics.

Query Syntax `:MEASure:STATistics:DISPlay?`

The :MEASure:STATistics:DISPlay? query returns the state of the measurement statistics display.

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

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:RESults" on page 488
- ":MEASure:STATistics" on page 496
- ":MEASure:STATistics:MCOunt" on page 499
- ":MEASure:STATistics:RESet" on page 500
- ":MEASure:STATistics:INCRement" on page 498
- ":MEASure:STATistics:RSDeviation" on page 501

:MEASure:STATistics:INCRement

 (see page 1354)

Command Syntax `:MEASure:STATistics:INCRement`

This command updates the statistics once (incrementing the count by one) using the current measurement values. It corresponds to the front panel **Increment Statistics** softkey in the Measurement Statistics Menu. This command lets you, for example, gather statistics over multiple pulses captured in a single acquisition. To do this, change the horizontal position and enter the command for each new pulse that is measured.

This command is only allowed when the oscilloscope is stopped and quick measurements are on.

The command is allowed in segmented acquisition mode even though the corresponding front panel softkey is not available.

See Also
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:STATistics" on page 496
- ":MEASure:STATistics:DISPlay" on page 497
- ":MEASure:STATistics:RESet" on page 500
- ":MEASure:RESults" on page 488

:MEASure:STATistics:MCOunt

(see page 1354)

Command Syntax

```
:MEASure:STATistics:MCOunt <setting>

<setting> ::= {INFinite | <count>}

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

The :MEASure:STATistics:MCOunt command specifies the maximum number of values used when calculating measurement statistics.

Query Syntax

```
:MEASure:STATistics:MCOunt?
```

The :MEASure:STATistics:MCOunt? query returns the current measurement statistics max count setting.

Return Format

```
<setting><NL>

<setting> ::= {INF | <count>}

<count> ::= 2 to 2000
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:RESults" on page 488
- ":MEASure:STATistics" on page 496
- ":MEASure:STATistics:DISPlay" on page 497
- ":MEASure:STATistics:RSDeviation" on page 501
- ":MEASure:STATistics:RESet" on page 500
- ":MEASure:STATistics:INCRement" on page 498

:MEASure:STATistics:RESet

 (see page 1354)

Command Syntax `:MEASure:STATistics:RESet`

This command resets the measurement statistics, zeroing the counts.

Note that the measurement (statistics) configuration is not deleted.

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:STATistics" on page 496
- ":MEASure:STATistics:DISPlay" on page 497
- ":MEASure:RESults" on page 488
- ":MEASure:STATistics:INCRement" on page 498

Example Code

- "Example Code" on page 488

:MEASure:STATistics:RSDeviation

 (see page 1354)

Command Syntax `:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}}`

The :MEASure:STATistics:RSDeviation command disables or enables relative standard deviations, that is, standard deviation/mean, in the measurement statistics.

Query Syntax `:MEASure:STATistics:RSDeviation?`

The :MEASure:STATistics:RSDeviation? query returns the current relative standard deviation setting.

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

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:RESults" on page 488
- ":MEASure:STATistics" on page 496
- ":MEASure:STATistics:DISPlay" on page 497
- ":MEASure:STATistics:MCOunt" on page 499
- ":MEASure:STATistics:RESet" on page 500
- ":MEASure:STATistics:INCRement" on page 498

:MEASure:TEDGe

N (see page 1354)

Query Syntax

```
:MEASure:TEDGe? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a
            space or plus sign (+). A falling edge is indicated by a
            minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number
                 is one, the first crossing from the left screen edge is
                 reported.  If the number is two, the second crossing is
                 reported, etc.

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The threshold voltage used for this measurement is at the 50% point with a bit of hysteresis added. You cannot change the threshold voltage used for this measurement with the :MEASure:DEFine command.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGe command:

Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel

Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see ":MEASure:TEDGe Code" on page 503.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>

<value> ::= time in seconds of the specified transition in NR3 format
```

:MEASure:TEDGe Code

```
' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.
' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:TVALue" on page 504
- ":MEASure:VTIMe" on page 514

:MEASure:TVALue

(see page 1354)

Query Syntax

```
:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross.  The
            value can be volts or a math function value such as dB,
            Vs, or V/s.

<slope> ::= direction of the waveform.  A rising slope is indicated
            by a plus sign (+).  A falling edge is indicated by a
            minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

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

```
<value> ::= time in seconds of the specified value crossing in
            NR3 format
```

See Also
- "Introduction to :MEASure Commands" on page 448
- ":MEASure:TEDGe" on page 502
- ":MEASure:VTIMe" on page 514

:MEASure:VAMPlitude

(see page 1354)

Command Syntax

```
:MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

vertical amplitude = Vtop - Vbase

Return Format

```
<value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VBASe" on page 508
- ":MEASure:VTOP" on page 515
- ":MEASure:VPP" on page 511

:MEASure:VAVerage

(see page 1354)

Command Syntax

```
:MEASure:VAVerage [<interval>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAVerage? [<interval>][,<source>]
```

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:VBASe

 (see page 1354)

Command Syntax

```
:MEASure:VBASe [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

```
<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in
                   NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VTOP" on page 515
- ":MEASure:VAMPlitude" on page 506
- ":MEASure:VMIN" on page 510

:MEASure:VMAX

(see page 1354)

Command Syntax

```
:MEASure:VMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMAX? [<source>]
```

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

Return Format

```
<value><NL>

<value> ::= maximum vertical value of the selected waveform in
            NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VMIN" on page 510
- ":MEASure:VPP" on page 511
- ":MEASure:VTOP" on page 515

:MEASure:VMIN

(see page 1354)

Command Syntax

```
:MEASure:VMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

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

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

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

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

Return Format

```
<value><NL>

<value> ::= minimum vertical value of the selected waveform in
            NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VBASe" on page 508
- ":MEASure:VMAX" on page 509
- ":MEASure:VPP" on page 511

:MEASure:VPP

(see page 1354)

Command Syntax

```
:MEASure:VPP [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

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

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

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

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

Vpp = Vmax - Vmin

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format

```
<value><NL>

<value> ::= vertical peak to peak value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VMAX" on page 509
- ":MEASure:VMIN" on page 510
- ":MEASure:VAMPlitude" on page 506

:MEASure:VRATio

(see page 1354)

Command Syntax

```
:MEASure:VRATio [<interval>][,<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure:VRATio command installs a ratio measurement on screen. Ratio measurements show the ratio of the ACRMS value of source1 to that of source2, expressed in dB.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRATio? [<interval>][<source1>][,<source2>]
```

The :MEASure:VRATio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

Return Format

```
<value><NL>

<value> ::= the ratio value in dB in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:VRMS" on page 513
- ":MEASure:SOURce" on page 494

:MEASure:VRMS

(see page 1354)

Command Syntax

```
:MEASure:VRMS [<interval>][,<type>][,<source>]

<interval> ::= {CYCLe | DISPlay}

<type> ::= {AC | DC}

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRMS? [<interval>][,<type>][,<source>]
```

The :MEASure:VRMS? query measures and outputs the RMS measurement value.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:VTIMe

 (see page 1354)

Query Syntax

```
:MEASure:VTIMe? <vtime_argument>[,<source>]

<vtime_argument> ::= time from trigger in seconds

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

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

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

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

The :MEASure:VTIMe? query returns the vertical value at a specified horizontal value on the source specified (see also :MEASure:SOURce). The specified horizontal value must be on the screen; when it is a time value, it is referenced to the trigger event. If the optional source parameter is specified, the measurement source is modified.

NOTE When the source is an FFT (Fast Fourier Transform) waveform, the <vtime_argument> is a frequency value instead of a time value.

Return Format

```
<value><NL>

<value> ::= vertical value at the specified horizontal location
            in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:TEDGe" on page 502
- ":MEASure:TVALue" on page 504

:MEASure:VTOP

(see page 1354)

Command Syntax

```
:MEASure:VTOP [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format

```
<value><NL>

<value> ::= vertical value at the top of the waveform in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494
- ":MEASure:VMAX" on page 509
- ":MEASure:VAMPlitude" on page 506
- ":MEASure:VBASe" on page 508

:MEASure:WINDow

 (see page 1354)

Command Syntax

```
:MEASure:WINDow <type>

<type> ::= {MAIN | ZOOM | AUTO | GATE}
```

The :MEASure:WINDow command lets you choose whether measurements are made in the Main window portion of the display, the Zoom window portion of the display (when the zoomed time base is displayed), or gated by the X1 and X2 cursors.

- MAIN – the measurement window is the Main window.
- ZOOM – the measurement window is the lower, Zoom window.
- AUTO – when the zoomed time base is displayed, the measurement is attempted in the lower, Zoom window; if it cannot be made there, or if the zoomed time base is not displayed, the Main window is used.
- GATE – the measurement window is between the X1 and X2 cursors. When the zoomed time base is displayed, the X1 and X2 cursors in the Zoom window portion of the display are used.

Query Syntax

```
:MEASure:WINDow?
```

The :MEASure:WINDow? query returns the current measurement window setting.

Return Format

```
<type><NL>

<type> ::= {MAIN | ZOOM | AUTO | GATE}
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:SOURce" on page 494

:MEASure:XMAX

N (see page 1354)

Command Syntax

```
:MEASure:XMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE :MEASure:XMAX is an alias for :MEASure:TMAX.

Query Syntax

```
:MEASure:XMAX? [<source>]
```

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the maximum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:XMIN" on page 518
- ":MEASure:TMAX" on page 1271

:MEASure:XMIN

N (see page 1354)

Command Syntax

```
:MEASure:XMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>}

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

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

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

The :MEASure:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE **:MEASure:XMIN is an alias for :MEASure:TMIN.**

Query Syntax

```
:MEASure:XMIN? [<source>]
```

The :MEASure:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the minimum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:XMAX" on page 517
- ":MEASure:TMIN" on page 1272

22 :MEASure Power Commands

These :MEASure commands are available when the DSOX3PWR power measurements and analysis application is licensed and enabled.

Table 97 :MEASure Power Commands Summary

Command	Query	Options and Query Returns
:MEASure:ANGLe [<source1>] [,<source2>] (see page 524)	:MEASure:ANGLe? [<source1>] [,<source2>] (see page 524)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the power phase angle in degrees in NR3 format
:MEASure:APParent [<source1>] [,<source2>] (see page 525)	:MEASure:APParent? [<source1>] [,<source2>] (see page 525)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the apparent power value in NR3 format
:MEASure:CPLoss [<source1>] [,<source2>] (see page 526)	:MEASure:CPLoss? [<source1>] [,<source2>] (see page 526)	<source1>, <source2> <source1> ::= {FUNCtion<m> | MATH<m>} <source2> ::= {CHANnel<n>} <m> ::= 1 to (# math functions) in NR1 format <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the switching loss per cycle watts value in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:CRESt [<source>] (see page 527)	:MEASure:CRESt? [<source>] (see page 527)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <return_value> ::= the crest factor value in NR3 format
:MEASure:EFFiciency (see page 528)	:MEASure:EFFiciency? (see page 528)	<return_value> ::= percent value in NR3 format
:MEASure:ELOSs [<source>] (see page 529)	:MEASure:ELOSs? [<source>] (see page 529)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the energy loss value in NR3 format
:MEASure:FACTor [<source1>][,<source2>] (see page 530)	:MEASure:FACTor? [<source1>][,<source2>] (see page 530)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the power factor value in NR3 format
:MEASure:IPOWer (see page 531)	:MEASure:IPOWer? (see page 531)	<return_value> ::= the input power value in NR3 format
:MEASure:OFFTime [<source1>][,<source2>] (see page 532)	:MEASure:OFFTime? [<source1>][,<source2>] (see page 532)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the time in seconds in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:ONTime [<source1>][,<source2>] (see page 533)	:MEASure:ONTime? [<source1>][,<source2>] (see page 533)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the time in seconds in NR3 format
:MEASure:OPOWer (see page 534)	:MEASure:OPOWer? (see page 534)	<return_value> ::= the output power value in NR3 format
:MEASure:PCURrent [<source>] (see page 535)	:MEASure:PCURrent? [<source>] (see page 535)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the peak current value in NR3 format
:MEASure:PLOSs [<source>] (see page 536)	:MEASure:PLOSs? [<source>] (see page 536)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the power loss value in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:RDSon [<source1>][,<source2>] (see page 537)	:MEASure:RDSon? [<source1>][,<source2>] (see page 537)	<source1>, <source2> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} \<n> ::= 1 to (# analog channels) in NR1 format \<m> ::= 1 to (# math functions) in NR1 format \<r> ::= 1 to (# ref waveforms) in NR1 format \<return_value> ::= the Rds(on) value in NR3 format
:MEASure:REACtive [<source1>][,<source2>] (see page 538)	:MEASure:REACtive? [<source1>][,<source2>] (see page 538)	<source1>, <source2> ::= {CHANnel<n>} \<n> ::= 1 to (# analog channels) in NR1 format \<return_value> ::= the reactive power value in NR3 format
:MEASure:REAL [<source>] (see page 539)	:MEASure:REAL? [<source>] (see page 539)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>} \<n> ::= 1 to (# analog channels) in NR1 format \<m> ::= 1 to (# math functions) in NR1 format \<return_value> ::= the real power value in NR3 format
:MEASure:RIPPle [<source>] (see page 540)	:MEASure:RIPPle? [<source>] (see page 540)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} \<n> ::= 1 to (# analog channels) in NR1 format \<m> ::= 1 to (# math functions) in NR1 format \<r> ::= 1 to (# ref waveforms) in NR1 format \<return_value> ::= the output ripple value in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:TRESponse [<source>] (see page 541)	:MEASure:TRESponse? [<source>] (see page 541)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= time in seconds for the overshoot to settle back into the band in NR3 format
:MEASure:VCESat [<source>] (see page 542)	:MEASure:VCESat? [<source>] (see page 542)	<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the Vce(sat) value in NR3 format

:MEASure:ANGLe

 (see page 1354)

Command Syntax

```
:MEASure:ANGLe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n>}

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

The :MEASure:ANGLe command installs a power phase angle 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.

Phase angle 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.

Query Syntax

```
:MEASure:ANGLe? [<source1>][,<source2>]
```

The :MEASure:ANGLe query returns the measured power phase angle in degrees.

Return Format

```
<return_value><NL>

<return_value> ::= the power phase angle in degrees in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:APParent

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:MEASure:APParent? [<source1>][,<source2>]
```

The :MEASure:APParent query returns the measured apparent power.

Return Format

```
<return_value><NL>

<return_value> ::= the apparent power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:CPLoss

N (see page 1354)

Command Syntax

```
:MEASure:CPLoss [<source1>][,<source2>]

<source1> ::= {FUNCtion<m> | MATH<m>}

<source2> ::= {CHANnel<n>}

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

<n> ::= 1 to (# analog channels) 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). 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 voltage of the switching signal.

Query Syntax

```
:MEASure:CPLoss? [<source1>][,<source2>]
```

The :MEASure:CPLoss query returns the switching loss per cycle in watts.

Return Format

```
<return_value><NL>

<return_value> ::= the switching loss per cycle value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:SWITch:APPLy" on page 669

:MEASure:CRESt

 (see page 1354)

Command Syntax

```
:MEASure:CRESt [<source>]

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

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

<m> ::= 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 Syntax

```
:MEASure:CRESt? [<source>]
```

The :MEASure:CRESt query returns the measured crest factor.

Return Format

```
<return_value><NL>

<return_value> ::= the crest factor value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:EFFiciency

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:MEASure:EFFiciency?
```

The :MEASure:EFFiciency query returns the measured efficiency as a percent value.

Return Format

```
<return_value><NL>

<return_value> ::= percent value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:EFFiciency:APPLy" on page 604

:MEASure:ELOSs

N (see page 1354)

Command Syntax

```
:MEASure:ELOSs [<source>]

<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure: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). 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 Syntax

```
:MEASure:ELOSs? [<source>]
```

The :MEASure:ELOSs query returns the switching loss in joules.

Return Format

```
<return_value><NL>

<return_value> ::= the energy loss value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:SWITch:APPLy" on page 669

:MEASure:FACTor

 (see page 1354)

Command Syntax

```
:MEASure:FACTor [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n>}

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

The :MEASure:FACTor command installs a power factor 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.

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 Syntax

```
:MEASure:FACTor? [<source1>][,<source2>]
```

The :MEASure:FACTor query returns the measured power factor.

Return Format

```
<return_value><NL>

<return_value> ::= the power factor value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:IPOWer

(see page 1354)

Command Syntax

```
: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 Syntax

```
:MEASure:IPOWer?
```

The :MEASure:IPOWer query returns the measured input power.

Return Format

```
<return_value><NL>

<return_value> ::= the input power value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:EFFiciency:APPLy" on page 604

:MEASure:OFFTime

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:MEASure:OFFTime? [<source1>][,<source2>]
```

The :MEASure:OFFTime query returns the measured turn off time.

Return Format

```
<return_value><NL>

<return_value> ::= the time in seconds in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:ONOFf:TEST" on page 629
- ":POWer:ONOFf:APPLy" on page 626

:MEASure:ONTime

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:MEASure:ONTime? [<source1>][,<source2>]
```

The :MEASure:ONTime query returns the measured turn off time.

Return Format

```
<return_value><NL>

<return_value> ::= the time in seconds in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:ONOFf:TEST" on page 629
- ":POWer:ONOFf:APPLy" on page 626

:MEASure:OPOWer

(see page 1354)

Command Syntax

```
:MEASure:OPOWer
```

The :MEASure:OPOWer 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 Syntax

```
:MEASure:OPOWer?
```

The :MEASure:OPOWer query returns the measured output power.

Return Format

```
<return_value><NL>

<return_value> ::= the output power value in NR3 format
```

See Also

- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:EFFiciency:APPLy" on page 604

:MEASure:PCURrent

N (see page 1354)

Command Syntax

```
:MEASure:PCURrent [<source>]

<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure: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 Syntax

```
:MEASure:PCURrent? [<source>]
```

The :MEASure:PCURrent query returns the measured peak current.

Return Format

```
<return_value><NL>

<return_value> ::= the peak current value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:INRush:APPLy" on page 619

:MEASure:PLOSs

(see page 1354)

Command Syntax

```
:MEASure:PLOSs [<source>]

<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure: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). 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 Syntax

```
:MEASure:PLOSs? [<source>]
```

The :MEASure:PLOSs query returns the switching loss in watts.

Return Format

```
<return_value><NL>

<return_value> ::= the power loss value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:SWITch:APPLy" on page 669

:MEASure:RDSon

 (see page 1354)

Command Syntax

```
:MEASure:RDSon [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>
   | WMEMory<r>}

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

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

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

The :MEASure: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 Syntax

```
:MEASure:RDSon? [<source1>][,<source2>]
```

The :MEASure:RDSon? query returns the measured Rds(on) value.

Return Format

```
<return_value><NL>

<return_value> ::= the Rds(on) value in NR3 format
```

See Also

- ":MEASure:VCESat" on page 542

:MEASure:REACtive

(see page 1354)

Command Syntax

```
:MEASure:REACtive [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n>}

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

The :MEASure:REACtive command installs a reactive 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.

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 Syntax

```
:MEASure:REACtive? [<source1>][,<source2>]
```

The :MEASure:REACtive query returns the measured reactive power.

Return Format

```
<return_value><NL>

<return_value> ::= the reactive power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:REAL

N (see page 1354)

Command Syntax

```
:MEASure:REAL [<source>]

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

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

<m> ::= 1 to (# math functions) 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). 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.

$$\text{Real Power} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} V_n I_n}$$

Query Syntax

```
:MEASure:REAL? [<source>]
```

The :MEASure:REAL query returns the measured real power.

Return Format

```
<return_value><NL>

<return_value> ::= the real power value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:QUALity:APPLy" on page 646

:MEASure:RIPPle

N (see page 1354)

Command Syntax

```
:MEASure:RIPPle [<source>]
<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1 to (# math functions) in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure: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 Syntax

```
:MEASure:RIPPle? [<source>]
```

The :MEASure:RIPPle query returns the measured output ripple.

Return Format

```
<return_value><NL>
<return_value> ::= the output ripple value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 494
- ":POWer:RIPPle:APPLy" on page 647

:MEASure:TRESponse

N (see page 1354)

Command Syntax

```
:MEASure:TRESponse [<source>]

<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure: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 Syntax

```
:MEASure:TRESponse? [<source>]
```

The :MEASure:TRESponse query returns the measured transient response time.

Return 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 494
- ":POWer:TRANsient:APPLy" on page 675

:MEASure:VCESat

(see page 1354)

Command Syntax

```
:MEASure:VCESat [<source>]

<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}

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

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

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

The :MEASure: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 Syntax

```
:MEASure:VCESat? [<source>]
```

The :MEASure:VCESat? query returns the measured Vce(sat) value.

Return Format

```
<return_value><NL>

<return_value> ::= the VCE(sat) value in NR3 format
```

See Also

- ":MEASure:RDSon" on page 537

23 :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "Introduction to :MTESt Commands" on page 545.

Table 98 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 548)	:MTESt:ALL? (see page 548)	{0 | 1}
:MTESt:AMASk:CREate (see page 549)	n/a	n/a
:MTESt:AMASk:SOURce <source> (see page 550)	:MTESt:AMASk:SOURce? (see page 550)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 551)	:MTESt:AMASk:UNITs? (see page 551)	<units> ::= {CURRent | DIVisions}
:MTESt:AMASk:XDELta <value> (see page 552)	:MTESt:AMASk:XDELta? (see page 552)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 553)	:MTESt:AMASk:YDELta? (see page 553)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 554)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 555)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 556)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 557)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 558)	:MTESt:DATA? (see page 558)	<mask> ::= data in IEEE 488.2 # format.

Table 98 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:DELete (see page 559)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 560)	:MTESt:ENABle? (see page 560)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 561)	:MTESt:LOCK? (see page 561)	{0 | 1}
:MTESt:RMODe <rmode> (see page 562)	:MTESt:RMODe? (see page 562)	<rmode> ::= {FOREver | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 563)	:MTESt:RMODe:FACTion:MEASure? (see page 563)	{0 | 1}
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 564)	:MTESt:RMODe:FACTion:SAVE? (see page 564)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 565)	:MTESt:RMODe:FACTion:STOP? (see page 565)	{0 | 1}
:MTESt:RMODe:SIGMa <level> (see page 566)	:MTESt:RMODe:SIGMa? (see page 566)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see page 567)	:MTESt:RMODe:TIME? (see page 567)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 568)	:MTESt:RMODe:WAVeforms? (see page 568)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 569)	:MTESt:SCALe:BIND? (see page 569)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 570)	:MTESt:SCALe:X1? (see page 570)	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALe:XDELta <xdelta_value> (see page 571)	:MTESt:SCALe:XDELta? (see page 571)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 572)	:MTESt:SCALe:Y1? (see page 572)	<y1_value> ::= Y1 value in NR3 format

Table 98 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:SCALe:Y2 <y2_value> (see page 573)	:MTESt:SCALe:Y2? (see page 573)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 574)	:MTESt:SOURce? (see page 574)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 575)	<title> ::= a string of up to 128 ASCII characters

Introduction to :MTESt Commands

Mask testing automatically compares the current displayed waveform with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

Reporting the Setup

Use :MTESt? to query setup information for the MTESt subsystem.

Return Format

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

```
:MTES:SOUR CHAN1;ENAB 0;LOCK 1;:MTES:AMAS:SOUR CHAN1;UNIT DIV;XDEL
+2.50000000E-001;YDEL +2.50000000E-001;:MTES:SCAL:X1 +200.000E-06;XDEL
+400.000E-06;Y1 -3.00000E+00;Y2 +3.00000E+00;BIND 0;:MTES:RMOD
FOR;RMOD:TIME +1E+00;WAV 1000;SIGM +6.0E+00;:MTES:RMOD:FACT:STOP
0;PRIN 0;SAVE 0
```

Example Code

```
' Mask testing commands example.
' ------------------------------------------------------------------

Option Explicit

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

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::10.112.94.136::hislip9-0.0::INSTR")
myScope.IO.Clear   ' Clear the interface.

' Make sure oscilloscope is running.
myScope.WriteString ":RUN"

' Set mask test termination conditions.
myScope.WriteString ":MTESt:RMODe SIGMa"
myScope.WriteString ":MTESt:RMODe?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test termination mode: " + strQueryResult

myScope.WriteString ":MTESt:RMODe:SIGMa 4.2"
myScope.WriteString ":MTESt:RMODe:SIGMa?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test termination 'test sigma': " + _
    FormatNumber(varQueryResult)

' Use auto-mask to create mask.
myScope.WriteString ":MTESt:AMASk:SOURce CHANnel1"
myScope.WriteString ":MTESt:AMASk:SOURce?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask source: " + strQueryResult

myScope.WriteString ":MTESt:AMASk:UNITs DIVisions"
myScope.WriteString ":MTESt:AMASk:UNITs?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask units: " + strQueryResult

myScope.WriteString ":MTESt:AMASk:XDELta 0.1"
myScope.WriteString ":MTESt:AMASk:XDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask X delta: " + _
    FormatNumber(varQueryResult)

myScope.WriteString ":MTESt:AMASk:YDELta 0.1"
myScope.WriteString ":MTESt:AMASk:YDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask Y delta: " + _
    FormatNumber(varQueryResult)

' Enable "Auto Mask Created" event (bit 10, &H400)
myScope.WriteString "*CLS"
myScope.WriteString ":MTEenable " + CStr(CInt("&H400"))

' Create mask.
myScope.WriteString ":MTESt:AMASk:CREate"
Debug.Print "Auto-mask created, mask test automatically enabled."

' Set up timeout variables.
Dim lngTimeout As Long   ' Max millisecs to wait.
Dim lngElapsed As Long
lngTimeout = 60000   ' 60 seconds.
```

```
  ' Wait until mask is created.
  lngElapsed = 0
  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERegister:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register MTE bit (bit 9, &H200).
    If (varQueryResult And &H200) <> 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Look for RUN bit = stopped (mask test termination).
  lngElapsed = 0
  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERegister:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Operation Status Condition Register RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get total waveforms, failed waveforms, and test time.
  myScope.WriteString ":MTESt:COUNt:WAVeforms?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test total waveforms: " + strQueryResult

  myScope.WriteString ":MTESt:COUNt:FWAVeforms?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test failed waveforms: " + strQueryResult

  myScope.WriteString ":MTESt:COUNt:TIME?"
  strQueryResult = myScope.ReadString
  Debug.Print "Mask test elapsed seconds: " + strQueryResult

  Exit Sub

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

End Sub
```

:MTESt:ALL

 (see page 1354)

Command Syntax

```
:MTESt:ALL <on_off>

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

The :MTESt:ALL command specifies the channel(s) that are included in the mask test:

- ON – All displayed analog channels are included in the mask test.
- OFF – Just the selected source channel is included in the test.

Query Syntax

```
:MTESt:ENABle?
```

The :MTESt:ENABle? query returns the current setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545

:MTESt:AMASk:CREate

 (see page 1354)

Command Syntax `:MTESt:AMASk:CREate`

The :MTESt:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the :MTESt:AMASk:XDELta, :MTESt:AMASk:YDELta, and :MTESt:AMASk:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTESt:SOURce command selects the channel and should be set before using this command.

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:XDELta" on page 552
- ":MTESt:AMASk:YDELta" on page 553
- ":MTESt:AMASk:UNITs" on page 551
- ":MTESt:AMASk:SOURce" on page 550
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:AMASk:SOURce

(see page 1354)

Command Syntax

```
:MTESt:AMASk:SOURce <source>

<source> ::= CHANnel<n>

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

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the :MTESt:AMASk:XDELta and :MTESt:AMASk:YDELta parameters when :MTESt:AMASk:UNITs is set to CURRent.

When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel<n>:UNITs command, of the selected source.

Suppose that UNITs are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

This command is the same as the :MTESt:SOURce command.

Query Syntax

```
:MTESt:AMASk:SOURce?
```

The :MTESt:AMASk:SOURce? query returns the currently set source.

Return Format

```
<source> ::= CHAN<n>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:XDELta" on page 552
- ":MTESt:AMASk:YDELta" on page 553
- ":MTESt:AMASk:UNITs" on page 551
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:AMASk:UNITs

N (see page 1354)

Command Syntax

```
:MTESt:AMASk:UNITs <units>

<units> ::= {CURRent | DIVisions}
```

The :MTESt:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by :MTESt:AMASk:XDELta and :MTESt:AMASk:YDELta commands.

- CURRent – the mask test subsystem uses the units as set by the :CHANnel<n>:UNITs command, usually time for ΔX and voltage for ΔY.
- DIVisions – the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURRent and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITs setting is changed.

Query Syntax

```
:MTESt:AMASk:UNITs?
```

The :MTESt:AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Return Format

```
<units><NL>

<units> ::= {CURR | DIV}
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:XDELta" on page 552
- ":MTESt:AMASk:YDELta" on page 553
- ":CHANnel<n>:UNITs" on page 293
- ":MTESt:AMASk:SOURce" on page 550
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:AMASk:XDELta

(see page 1354)

Command Syntax

```
:MTESt:AMASk:XDELta <value>

<value> ::= X delta value in NR3 format
```

The :MTESt:AMASk:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

The horizontal tolerance value is interpreted based on the setting specified by the :MTESt:AMASk:UNITs command; thus, if you specify 250-E3, the setting for :MTESt:AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for :MTESt:AMASk:UNITs is DIVisions, the same X delta value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Query Syntax

```
:MTESt:AMASk:XDELta?
```

The :MTEST:AMASk:XDELta? query returns the current setting of the ΔX tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the :MTESt:AMASk:UNITs query.

Return Format

```
<value><NL>

<value> ::= X delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:UNITs" on page 551
- ":MTESt:AMASk:YDELta" on page 553
- ":MTESt:AMASk:SOURce" on page 550
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:AMASk:YDELta

(see page 1354)

Command Syntax

```
:MTESt:AMASk:YDELta <value>

<value> ::= Y delta value in NR3 format
```

The :MTESt:AMASk:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

The vertical tolerance value is interpreted based on the setting specified by the :MTESt:AMASk:UNITs command; thus, if you specify 250-E3, the setting for :MTESt:AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for :MTESt:AMASk:UNITs is DIVisions, the same Y delta value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Query Syntax

```
:MTESt:AMASk:YDELta?
```

The :MTESt:AMASk:YDELta? query returns the current setting of the ΔY tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the :MTESt:AMASk:UNITs query.

Return Format

```
<value><NL>

<value> ::= Y delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:UNITs" on page 551
- ":MTESt:AMASk:XDELta" on page 552
- ":MTESt:AMASk:SOURce" on page 550
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:COUNt:FWAVeforms

(see page 1354)

Query Syntax

```
:MTESt:COUNt:FWAVeforms? [CHANnel<n>]

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

The :MTESt:COUNt:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms collected on the channel specified by the optional parameter or collected on the currently specified source channel (:MTESt:SOURce) if there is no parameter.

Return Format

```
<failed><NL>

<failed> ::= number of failed waveforms in NR1 format.
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:COUNt:WAVeforms" on page 557
- ":MTESt:COUNt:TIME" on page 556
- ":MTESt:COUNt:RESet" on page 555
- ":MTESt:SOURce" on page 574

Example Code

- "Example Code" on page 545

:MTESt:COUNt:RESet

(see page 1354)

Command Syntax `:MTESt:COUNt:RESet`

The :MTESt:COUNt:RESet command resets the mask statistics.

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:COUNt:WAVeforms" on page 557
- ":MTESt:COUNt:FWAVeforms" on page 554
- ":MTESt:COUNt:TIME" on page 556

:MTESt:COUNt:TIME

(see page 1354)

Query Syntax

```
:MTESt:COUNt:TIME?
```

The :MTESt:COUNt:TIME? query returns the elapsed time in the current mask test run.

Return Format

```
<time><NL>

<time> ::= elapsed seconds in NR3 format.
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:COUNt:WAVeforms" on page 557
- ":MTESt:COUNt:FWAVeforms" on page 554
- ":MTESt:COUNt:RESet" on page 555

Example Code

- "Example Code" on page 545

:MTESt:COUNt:WAVeforms

(see page 1354)

Query Syntax

```
:MTESt:COUNt:WAVeforms?
```

The :MTESt:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run.

Return Format

```
<count><NL>

<count> ::= number of waveforms in NR1 format.
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:COUNt:FWAVeforms" on page 554
- ":MTESt:COUNt:TIME" on page 556
- ":MTESt:COUNt:RESet" on page 555

Example Code

- "Example Code" on page 545

:MTESt:DATA

(see page 1354)

Command Syntax

```
:MTESt:DATA <mask>

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

The :MTESt:DATA command loads a mask from binary block data.

Query Syntax

```
:MTESt:DATA?
```

The :MTESt:DATA? query returns a mask in binary block data format. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

```
<mask><NL>

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

See Also

- ":SAVE:MASK[:STARt]" on page 703
- ":RECall:MASK[:STARt]" on page 687

:MTESt:DELete

 (see page 1354)

Command Syntax `:MTESt:DELete`

The :MTESt:DELete command clears the currently loaded mask.

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:CREate" on page 549

:MTESt:ENABle

(see page 1354)

Command Syntax

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

The :MTESt:ENABle command enables or disables the mask test features.

- ON – Enables the mask test features.
- OFF – Disables the mask test features.

Query Syntax

```
:MTESt:ENABle?
```

The :MTESt:ENABle? query returns the current state of mask test features.

Return Format

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

See Also

- "Introduction to :MTESt Commands" on page 545

:MTESt:LOCK

(see page 1354)

Command Syntax

```
:MTESt:LOCK <on_off>

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

The :MTESt:LOCK command enables or disables the mask lock feature:

- ON – Locks a mask to the SOURce. As the vertical or horizontal scaling or position of the SOURce changes, the mask is redrawn accordingly.
- OFF – The mask is static and does not move.

Query Syntax

```
:MTESt:LOCK?
```

The :MTESt:LOCK? query returns the current mask lock setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SOURce" on page 574

:MTESt:RMODe

(see page 1354)

Command Syntax

```
:MTESt:RMODe <rmode>

<rmode> ::= {FORever | SIGMa | TIME | WAVeforms}
```

The :MTESt:RMODe command specifies the termination conditions for the mask test:

- FORever – the mask test runs until it is turned off.
- SIGMa – the mask test runs until the Sigma level is reached. This level is set by the ":MTESt:RMODe:SIGMa" on page 566 command.
- TIME – the mask test runs for a fixed amount of time. The amount of time is set by the ":MTESt:RMODe:TIME" on page 567 command.
- WAVeforms – the mask test runs until a fixed number of waveforms are acquired. The number of waveforms is set by the ":MTESt:RMODe:WAVeforms" on page 568 command.

Query Syntax

```
:MTESt:RMODe?
```

The :MTESt:RMODe? query returns the currently set termination condition.

Return Format

```
<rmode><NL>

<rmode> ::= {FOR | SIGM | TIME | WAV}
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe:SIGMa" on page 566
- ":MTESt:RMODe:TIME" on page 567
- ":MTESt:RMODe:WAVeforms" on page 568

Example Code

- "Example Code" on page 545

:MTESt:RMODe:FACTion:MEASure

(see page 1354)

Command Syntax

```
:MTESt:RMODe:FACTion:MEASure <on_off>

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

The :MTESt:RMODe:FACTion:MEASure command sets measuring only mask failures on or off.

When ON, measurements and measurement statistics run only on waveforms that contain a mask violation; passing waveforms do not affect measurements and measurement statistics.

This mode is not available when the acquisition mode is set to Averaging.

Query Syntax

```
:MTESt:RMODe:FACTion:MEASure?
```

The :MTESt:RMODe:FACTion:MEASure? query returns the current mask failure measure setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe:FACTion:SAVE" on page 564
- ":MTESt:RMODe:FACTion:STOP" on page 565

:MTESt:RMODe:FACTion:SAVE

(see page 1354)

Command Syntax

```
:MTESt:RMODe:FACTion:SAVE <on_off>

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

The :MTESt:RMODe:FACTion:SAVE command sets saving on mask failures on or off.

NOTE **Setting :MTESt:RMODe:FACTion:SAVE ON automatically sets :MTESt:RMODe:FACTion:PRINt OFF.**

See Chapter 26, ":SAVE Commands," starting on page 691 for more information on save options.

Query Syntax

```
:MTESt:RMODe:FACTion:SAVE?
```

The :MTESt:RMODe:FACTion:SAVE? query returns the current mask failure save setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe:FACTion:MEASure" on page 563
- ":MTESt:RMODe:FACTion:STOP" on page 565

:MTESt:RMODe:FACTion:STOP

 (see page 1354)

Command Syntax

```
:MTESt:RMODe:FACTion:STOP <on_off>

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

The :MTESt:RMODe:FACTion:STOP command sets stopping on a mask failure on or off. When this setting is ON and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Query Syntax

```
:MTESt:RMODe:FACTion:STOP?
```

The :MTESt:RMODe:FACTion:STOP? query returns the current mask failure stop setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe:FACTion:MEASure" on page 563
- ":MTESt:RMODe:FACTion:SAVE" on page 564

:MTESt:RMODe:SIGMa

(see page 1354)

Command Syntax

```
:MTESt:RMODe:SIGMa <level>

<level> ::= from 0.1 to 9.3 in NR3 format
```

When the :MTESt:RMODe command is set to SIGMa, the :MTESt:RMODe:SIGMa command sets the *test sigma* level to which a mask test runs. *Test sigma* is the best achievable process sigma, assuming no failures. (*Process sigma* is calculated using the number of failures per test.) The *test sigma* level indirectly specifies the number of waveforms that must be tested (in order to reach the sigma level).

Query Syntax

```
:MTESt:RMODe:SIGMa?
```

The :MTESt:RMODe:SIGMa? query returns the current Sigma level setting.

Return Format

```
<level><NL>

<level> ::= from 0.1 to 9.3 in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe" on page 562

Example Code

- "Example Code" on page 545

:MTESt:RMODe:TIME

(see page 1354)

Command Syntax

```
:MTESt:RMODe:TIME <seconds>

<seconds> ::= from 1 to 86400 in NR3 format
```

When the :MTESt:RMODe command is set to TIME, the :MTESt:RMODe:TIME command sets the number of seconds for a mask test to run.

Query Syntax

```
:MTESt:RMODe:TIME?
```

The :MTESt:RMODe:TIME? query returns the number of seconds currently set.

Return Format

```
<seconds><NL>

<seconds> ::= from 1 to 86400 in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe" on page 562

:MTESt:RMODe:WAVeforms

(see page 1354)

Command Syntax

```
:MTESt:RMODe:WAVeforms <count>

<count> ::= number of waveforms in NR1 format
            from 1 to 2,000,000,000
```

When the :MTESt:RMODe command is set to WAVeforms, the :MTESt:RMODe:WAVeforms command sets the number of waveform acquisitions that are mask tested.

Query Syntax

```
:MTESt:RMODe:WAVeforms?
```

The :MTESt:RMODe:WAVeforms? query returns the number of waveforms currently set.

Return Format

```
<count><NL>

<count> ::= number of waveforms in NR1 format
            from 1 to 2,000,000,000
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RMODe" on page 562

:MTESt:SCALe:BIND

(see page 1354)

Command Syntax

```
:MTESt:SCALe:BIND <on_off>

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

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control:

- ON –

  If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

  If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size.

- OFF –

  If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

  If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

Query Syntax

```
:MTESt:SCALe:BIND?
```

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SCALe:X1" on page 570
- ":MTESt:SCALe:XDELta" on page 571
- ":MTESt:SCALe:Y1" on page 572
- ":MTESt:SCALe:Y2" on page 573

:MTESt:SCALe:X1

 (see page 1354)

Command Syntax

```
:MTESt:SCALe:X1 <x1_value>

<x1_value> ::= X1 value in NR3 format
```

The :MTESt:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the :MTESt:SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$$X = (X * \Delta X) + X1$$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

The X1 value is a time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Query Syntax

```
:MTESt:SCALe:X1?
```

The :MTESt:SCALe:X1? query returns the current X1 coordinate setting.

Return Format

```
<x1_value><NL>

<x1_value> ::= X1 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SCALe:BIND" on page 569
- ":MTESt:SCALe:XDELta" on page 571
- ":MTESt:SCALe:Y1" on page 572
- ":MTESt:SCALe:Y2" on page 573

:MTESt:SCALe:XDELta

N (see page 1354)

Command Syntax

```
:MTESt:SCALe:XDELta <xdelta_value>

<xdelta_value> ::= X delta value in NR3 format
```

The :MTESt:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then, a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

```
X = (X * ΔX) + X1
```

The X delta value is a time value specifying the distance of the X2 marker with respect to the X1 marker.

For example, if the period of the waveform you wish to test is 1 ms, setting ΔX to 1 ms ensures that the waveform's period is between the X1 and X2 markers.

Query Syntax

```
:MTESt:SCALe:XDELta?
```

The :MTESt:SCALe:XDELta? query returns the current value of ΔX.

Return Format

```
<xdelta_value><NL>

<xdelta_value> ::= X delta value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SCALe:BIND" on page 569
- ":MTESt:SCALe:X1" on page 570
- ":MTESt:SCALe:Y1" on page 572
- ":MTESt:SCALe:Y2" on page 573

:MTESt:SCALe:Y1

 (see page 1354)

Command Syntax

```
:MTESt:SCALe:Y1 <y1_value>

<y1_value> ::= Y1 value in NR3 format
```

The :MTESt:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation:

```
Y = (Y * (Y2 - Y1)) + Y1
```

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y1 value is a voltage value specifying the point at which Y=0.

Query Syntax

```
:MTESt:SCALe:Y1?
```

The :MTESt:SCALe:Y1? query returns the current setting of the Y1 marker.

Return Format

```
<y1_value><NL>

<y1_value> ::= Y1 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SCALe:BIND" on page 569
- ":MTESt:SCALe:X1" on page 570
- ":MTESt:SCALe:XDELta" on page 571
- ":MTESt:SCALe:Y2" on page 573

:MTESt:SCALe:Y2

 (see page 1354)

Command Syntax

```
:MTESt:SCALe:Y2 <y2_value>

<y2_value> ::= Y2 value in NR3 format
```

The :MTESt:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation:

```
Y = (Y * (Y2 - Y1)) + Y1
```

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

The Y2 value is a voltage value specifying the location of the Y2 marker.

Query Syntax

```
:MTESt:SCALe:Y2?
```

The :MTESt:SCALe:Y2? query returns the current setting of the Y2 marker.

Return Format

```
<y2_value><NL>

<y2_value> ::= Y2 value in NR3 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:SCALe:BIND" on page 569
- ":MTESt:SCALe:X1" on page 570
- ":MTESt:SCALe:XDELta" on page 571
- ":MTESt:SCALe:Y1" on page 572

:MTESt:SOURce

(see page 1354)

Command Syntax

```
:MTESt:SOURce <source>

<source> ::= CHANnel<n>

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

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

Query Syntax

```
:MTESt:SOURce?
```

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Return Format

```
<source><NL>

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

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

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:SOURce" on page 550

:MTESt:TITLe

N (see page 1354)

Query Syntax `:MTESt:TITLe?`

The :MTESt:TITLe? query returns the mask title which is a string of up to 128 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

Return Format `<title><NL>`

`<title> ::= a string of up to 128 ASCII characters.`

See Also

- "Introduction to :MTESt Commands" on page 545

24 :POWer Commands

These :POWer commands are available when the DSOX3PWR power measurements and analysis application is licensed and enabled.

Table 99 :POWer Commands Summary

Command	Query	Options and Query Returns
n/a	:POWer:CLResponse? (see page 585)	n/a
:POWer:CLResponse:APPLy (see page 586)	n/a	n/a
n/a	:POWer:CLResponse:DATA? [SWEep | SINGle] (see page 587)	<binary_block> ::= comma-separated data with newlines at the end of each row
n/a	:POWer:CLResponse:DATA:GMARgin? (see page 588)	<gain_margin> ::= gain margin in dB in NR3 format.
n/a	:POWer:CLResponse:DATA:GMARgin:FREQuency? (see page 589)	<frequency> ::= 0 degrees phase crossover frequency in Hz in NR3 format
n/a	:POWer:CLResponse:DATA:PMARgin? (see page 590)	<phase_margin> ::= phase margin in degrees in NR3 format.
n/a	:POWer:CLResponse:DATA:PMARgin:FREQuency? (see page 591)	<frequency> ::= 0dB gain crossover frequency in Hz in NR3 format.
:POWer:CLResponse:FREQuency:MODE <mode> (see page 592)	:POWer:CLResponse:FREQuency:MODE? (see page 592)	<mode> ::= {SWEep | SINGle}
:POWer:CLResponse:FREQuency:SINGle <value>[suffix] (see page 593)	:POWer:CLResponse:FREQuency:SINGle? (see page 593)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:CLResponse:FREQuency:STARt <value>[suffix] (see page 594)	:POWer:CLResponse:FREQuency:STARt? (see page 594)	<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:CLResponse:FREQuency:STOP <value>[suffix] (see page 595)	:POWer:CLResponse:FREQuency:STOP? (see page 595)	<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:CLResponse:PPDecade <pts> (see page 596)	:POWer:CLResponse:PPDecade? (see page 596)	<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:POWer:CLResponse:SOURce:INPut <source> (see page 597)	:POWer:CLResponse:SOURce:INPut? (see page 597)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:CLResponse:SOURce:OUTPut <source> (see page 598)	:POWer:CLResponse:SOURce:OUTPut? (see page 598)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:CLResponse:TRACe <selection> (see page 599)	:POWer:CLResponse:TRACe? (see page 599)	<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
:POWer:CLResponse:WGEN:LOAD <impedance> (see page 600)	:POWer:CLResponse:WGEN:LOAD? (see page 600)	<impedance> ::= {ONEMeg | FIFTy}
:POWer:CLResponse:WGEN:VOLTage <amplitude>[,<range>] (see page 601)	:POWer:CLResponse:WGEN:VOLTage? [<range>] (see page 601)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:POWer:CLResponse:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 602)	:POWer:CLResponse:WGEN:VOLTage:PROFile? (see page 602)	{0 | 1}
:POWer:DESKew (see page 603)	n/a	n/a
:POWer:EFFiciency:APPLy (see page 604)	n/a	n/a
:POWer:EFFiciency:TYPE <type> (see page 605)	:POWer:EFFiciency:TYPE? (see page 605)	<type> ::= {DCDC | DCAC | ACDC | ACAC}

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:ENABle {{0 | OFF} | {1 | ON}} (see page 606)	:POWer:ENABle? (see page 606)	{0 | 1}
:POWer:HARMonics:APPLy (see page 607)	n/a	n/a
n/a	:POWer:HARMonics:DATA? (see page 608)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISPlay <display> (see page 609)	:POWer:HARMonics:DISPlay? (see page 609)	<display> ::= {TABLe | BAR | OFF}
n/a	:POWer:HARMonics:FAILcount? (see page 610)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 611)	:POWer:HARMonics:LINE? (see page 611)	<frequency> ::= {F50 | F60 | F400 | AUTO}
n/a	:POWer:HARMonics:POWerfactor? (see page 612)	<value> ::= Class C power factor in NR3 format
:POWer:HARMonics:RPOWer <source> (see page 613)	:POWer:HARMonics:RPOWer? (see page 613)	<source> ::= {MEASured | USER}
:POWer:HARMonics:RPOWer:USER <value> (see page 614)	:POWer:HARMonics:RPOWer:USER? (see page 614)	<value> ::= Watts from 1.0 to 600.0 in NR3 format
n/a	:POWer:HARMonics:RUNCount? (see page 615)	<count> ::= integer in NR1 format
:POWer:HARMonics:STANdard <class> (see page 616)	:POWer:HARMonics:STANdard? (see page 616)	<class> ::= {A | B | C | D}
n/a	:POWer:HARMonics:STATus? (see page 617)	<status> ::= {PASS | FAIL | UNTested}
n/a	:POWer:HARMonics:THD? (see page 618)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLy (see page 619)	n/a	n/a
:POWer:INRush:EXIT (see page 620)	n/a	n/a

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:INRush:NEXT (see page 621)	n/a	n/a
:POWer:ITYPe <type> (see page 622)	:POWer:ITYPe? (see page 622)	<type> ::= {DC | AC}
:POWer:MODulation:APPLy (see page 623)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 624)	:POWer:MODulation:SOURce? (see page 624)	<source> ::= {V | I}
:POWer:MODulation:TYPE <modulation> (see page 625)	:POWer:MODulation:TYPE? (see page 625)	<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDith | NWIDth | DUTYcycle | RISetime | FALLtime}
:POWer:ONOFf:APPLy (see page 626)	n/a	n/a
:POWer:ONOFf:EXIT (see page 627)	n/a	n/a
:POWer:ONOFf:NEXT (see page 628)	n/a	n/a
:POWer:ONOFf:TEST {{0 | OFF} | {1 | ON}} (see page 629)	:POWer:ONOFf:TEST? (see page 629)	{0 | 1}
:POWer:ONOFf:THResholds <type>, <input_thr>, <output_thr> (see page 630)	:POWer:ONOFf:THResholds? <type> (see page 630)	<type> ::= {0 | 1} <input_thr> ::= percent from 0-100 in NR1 format <output_thr> ::= percent from 0-100 in NR1 format
n/a	:POWer:PSRR? (see page 632)	n/a
:POWer:PSRR:APPLy (see page 633)	n/a	n/a
n/a	:POWer:PSRR:DATA? [SWEep | SINGle] (see page 634)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:PSRR:FREQuency:MAXimum <value>[suffix] (see page 635)	:POWer:PSRR:FREQuency:MAXimum? (see page 635)	<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:FREQuency :MINimum <value>[suffix] (see page 636)	:POWer:PSRR:FREQuency :MINimum? (see page 636)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:FREQuency :MODE <mode> (see page 637)	:POWer:PSRR:FREQuency :MODE? (see page 637)	<mode> ::= {SWEep | SINGle}
:POWer:PSRR:FREQuency :SINGle <value>[suffix] (see page 638)	:POWer:PSRR:FREQuency :SINGle? (see page 638)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 2000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:PPDecade <pts> (see page 639)	:POWer:PSRR:PPDecade? (see page 639)	<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
:POWer:PSRR:SOURce:INPut <source> (see page 640)	:POWer:PSRR:SOURce:INPut? (see page 640)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:PSRR:SOURce:OUTPut <source> (see page 641)	:POWer:PSRR:SOURce:OUTPut? (see page 641)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:PSRR:TRACe <selection> (see page 642)	:POWer:PSRR:TRACe? (see page 642)	<selection> ::= {NONE | GAIN}
:POWer:PSRR:WGEN:LOAD <impedance> (see page 643)	:POWer:PSRR:WGEN:LOAD? (see page 643)	<impedance> ::= {ONEMeg | FIFTy}
:POWer:PSRR:WGEN:VOLTage <amplitude>[,<range>] (see page 644)	:POWer:PSRR:WGEN:VOLTage? [<range>] (see page 644)	<amplitude> ::= amplitude in volts in NR3 format <range> ::= {F20HZ | F100HZ | F1KHZ | F10KHZ | F100KHZ | F1MHZ | F10MHZ | F20MHZ}
:POWer:PSRR:WGEN:VOLTage:PROFile {{0 | OFF} | {1 | ON}} (see page 645)	:POWer:PSRR:WGEN:VOLTage:PROFile? (see page 645)	{0 | 1}
:POWer:QUALity:APPLy (see page 646)	n/a	n/a
:POWer:RIPPle:APPLy (see page 647)	n/a	n/a

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:AUTosetup <analysis> (see page 648)	n/a	<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity | SLEW | SWITch | RDSVce}
:POWer:SIGNals:CYCLes:HARMonics <count> (see page 649)	:POWer:SIGNals:CYCLes:HARMonics? (see page 649)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:CYCLes:QUALity <count> (see page 650)	:POWer:SIGNals:CYCLes:QUALity? (see page 650)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURation:EFFiciency <value>[suffix] (see page 651)	:POWer:SIGNals:DURation:EFFiciency? (see page 651)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:MODulation <value>[suffix] (see page 652)	:POWer:SIGNals:DURation:MODulation? (see page 652)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:ONOFf:OFF <value>[suffix] (see page 653)	:POWer:SIGNals:DURation:ONOFf:OFF? (see page 653)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:ONOFf:ON <value>[suffix] (see page 654)	:POWer:SIGNals:DURation:ONOFf:ON? (see page 654)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:RIPPle <value>[suffix] (see page 655)	:POWer:SIGNals:DURation:RIPPle? (see page 655)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:DURation:TRANsient <value>[suffix] (see page 656)	:POWer:SIGNals:DURation:TRANsient? (see page 656)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:IEXPected <value>[suffix] (see page 657)	:POWer:SIGNals:IEXPected? (see page 657)	<value> ::= Expected current value in NR3 format [suffix] ::= {A | mA}
:POWer:SIGNals:OVERshoot <percent> (see page 658)	:POWer:SIGNals:OVERshoot? (see page 658)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V | mV}}

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:VMAXimum:INRush <value>[suffix] (see page 659)	:POWer:SIGNals:VMAXimum:INRush? (see page 659)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VMAXimum:ONOFf:OFF <value>[suffix] (see page 660)	:POWer:SIGNals:VMAXimum:ONOFf:OFF? (see page 660)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VMAXimum:ONOFf:ON <value>[suffix] (see page 661)	:POWer:SIGNals:VMAXimum:ONOFf:ON? (see page 661)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady:ONOFf:OFF <value>[suffix] (see page 662)	:POWer:SIGNals:VSTeady:ONOFf:OFF? (see page 662)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady:ONOFf:ON <value>[suffix] (see page 663)	:POWer:SIGNals:VSTeady:ONOFf:ON? (see page 663)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady:TRANsient <value>[suffix] (see page 664)	:POWer:SIGNals:VSTeady:TRANsient? (see page 664)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:SOURce:CURRent<i> <source> (see page 665)	:POWer:SIGNals:SOURce:CURRent<i>? (see page 665)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SIGNals:SOURce:VOLTage<i> <source> (see page 666)	:POWer:SIGNals:SOURce:VOLTage<i>? (see page 666)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SLEW:APPLy (see page 667)	n/a	n/a
:POWer:SLEW:SOURce <source> (see page 668)	:POWer:SLEW:SOURce? (see page 668)	<source> ::= {V | I}
:POWer:SWITch:APPLy (see page 669)	n/a	n/a

Table 99 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SWITch:CONDuction <conduction> (see page 670)	:POWer:SWITch:CONDuction? (see page 670)	<conduction> ::= {WAVeform | RDS | VCE}
:POWer:SWITch:IREFerence <percent> (see page 671)	:POWer:SWITch:IREFerence? (see page 671)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 672)	:POWer:SWITch:RDS? (see page 672)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM | mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 673)	:POWer:SWITch:VCE? (see page 673)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V | mV}
:POWer:SWITch:VREFerence <percent> (see page 674)	:POWer:SWITch:VREFerence? (see page 674)	<percent> ::= percent in NR1 format
:POWer:TRANsient:APPLy (see page 675)	n/a	n/a
:POWer:TRANsient:EXIT (see page 676)	n/a	n/a
:POWer:TRANsient:IINitial <value>[suffix] (see page 677)	:POWer:TRANsient:IINitial? (see page 677)	<value> ::= Initial current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:INEW <value>[suffix] (see page 678)	:POWer:TRANsient:INEW? (see page 678)	<value> ::= New current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:NEXT (see page 679)	n/a	n/a

:POWer:CLResponse

N (see page 1354)

Query Syntax `:POWer:CLResponse?`

The :POWer:CLResponse? query returns the Control Loop Response (Bode) power analysis settings.

Return Format `<settings_string><NL>`

For example, the query returns the following string when issued after the *RST command.

```
:POW:CLR:SOUR:INP CHAN1;OUTP CHAN2;:POW:CLR:FREQ:STAR +100E+00;
STOP +20.000000E+06;:POW:CLR:WGEN:VOLT +200.0E-03;LOAD FIFT
```

See Also

- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:APPLy

 (see page 1354)

Command Syntax `: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.

You can use the :POWer:CLResponse:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

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 (Standard Event Status Register)" on page 181
- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:SINGle" on page 593
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:TRACe" on page 599
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:DATA

(see page 1354)

Query Syntax

```
: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.

You can use the :POWer:CLResponse:TRACe command to specify whether to include gain, phase, both gain and phase, or neither in the frequency response analysis results.

The SWEep or SINGle option specifies whether to get the data from a sweep or single-frequency analysis (see :POWer:CLResponse:FREQuency:MODE). If this option is not specified, the data from the sweep analysis is returned by default.

Return Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA:GMARgin" on page 588
- ":POWer:CLResponse:DATA:GMARgin:FREQuency" on page 589
- ":POWer:CLResponse:DATA:PMARgin" on page 590
- ":POWer:CLResponse:DATA:PMARgin:FREQuency" on page 591
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:SINGle" on page 593
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:TRACe" on page 599
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:DATA:GMARgin

N (see page 1354)

Query Syntax

```
: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.

Return 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 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:DATA:GMARgin:FREQuency" on page 589
- ":POWer:CLResponse:DATA:PMARgin" on page 590
- ":POWer:CLResponse:DATA:PMARgin:FREQuency" on page 591

:POWer:CLResponse:DATA:GMARgin:FREQuency

N (see page 1354)

Query Syntax

```
: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.

Return 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 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:DATA:GMARgin" on page 588
- ":POWer:CLResponse:DATA:PMARgin" on page 590
- ":POWer:CLResponse:DATA:PMARgin:FREQuency" on page 591

:POWer:CLResponse:DATA:PMARgin

 (see page 1354)

Query Syntax `: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.

Return 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 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:DATA:GMARgin" on page 588
- ":POWer:CLResponse:DATA:GMARgin:FREQuency" on page 589
- ":POWer:CLResponse:DATA:PMARgin:FREQuency" on page 591

:POWer:CLResponse:DATA:PMARgin:FREQuency

N (see page 1354)

Query Syntax

```
: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.

Return 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 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:DATA:GMARgin" on page 588
- ":POWer:CLResponse:DATA:GMARgin:FREQuency" on page 589
- ":POWer:CLResponse:DATA:PMARgin" on page 590

:POWer:CLResponse:FREQuency:MODE

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:FREQuency:MODE?
```

The :POWer:CLResponse:FREQuency:MODE? query returns the frequency mode setting.

Return Format

```
<mode><NL>

<mode> ::= {SWE | SING}
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:SINGle" on page 593
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:FREQuency:SINGle

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:FREQuency:SINGle?
```

The :POWer:CLResponse:FREQuency:SINGle? query returns the single frequency setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:FREQuency:STARt

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:FREQuency:STARt?
```

The :POWer:CLResponse:FREQuency:STARt? query returns the frequency sweep start setting.

Return Format

```
<value><NL>

<value> ::= {20 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:FREQuency:STOP

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:FREQuency:STOP?
```

The :POWer:CLResponse:FREQuency:STOP? query returns the frequency sweep stop setting.

Return Format

```
<value><NL>

<value> ::= {100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000
}
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:PPDecade

 (see page 1354)

Command Syntax

```
:POWer:CLResponse:PPDecade <pts>

<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

The :POWer:CLResponse:PPDecade command selects the number of frequency test points per decade (in the log scale).

Query Syntax

```
:POWer:CLResponse:PPDecade?
```

The :POWer:CLResponse:PPDecade? query returns the points per decade setting.

Return Format

```
<pts><NL>

<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:SOURce:INPut

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:SOURce:INPut?
```

The :POWer:CLResponse:SOURce:INPut? query returns the channel selection.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:SOURce:OUTPut

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:SOURce:OUTPut?
```

The :POWer:CLResponse:SOURce:OUTPut? query returns the channel selection.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:TRACe

(see page 1354)

Command Syntax

```
:POWer:CLResponse:TRACe <selection>

<selection> ::= {NONE | ALL | GAIN | PHASe}[,{GAIN | PHASe}]
```

The :POWer:CLResponse:TRACe command specifies whether to include gain, phase, both gain and phase, or neither in the control loop response analysis results.

NOTE This command affects the oscilloscope's front panel graphical user interface (plot and table) as well as when saving analysis data.

Query Syntax

```
:POWer:CLResponse:TRACe?
```

The :POWer:CLResponse:TRACe? query returns a comma-separated list of the types of data that are currently included in the control loop response analysis results, or "NONE" if neither gain nor phase data is included.

Return Format

```
<selection_list><NL>

<selection_list> ::= {"NONE" | "GAIN" | "PHASe" | "GAIN,PHASe"}
```

See Also

- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587

:POWer:CLResponse:WGEN:LOAD

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:WGEN:LOAD?
```

The :POWer:CLResponse:WGEN:LOAD? query returns the waveform generator expected output load impedance setting.

Return Format

```
<impedance><NL>

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

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:VOLTage" on page 601
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:WGEN:VOLTage

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:WGEN:VOLTage? [<range>]
```

The :POWer:CLResponse:WGEN:VOLTage? query returns the waveform generator output amplitude setting(s).

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage:PROFile" on page 602

:POWer:CLResponse:WGEN:VOLTage:PROFile

N (see page 1354)

Command Syntax `: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 Syntax `:POWer:CLResponse:WGEN:VOLTage:PROFile?`

The :POWer:CLResponse:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Return Format `<setting><NL>`

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

See Also

- ":POWer:CLResponse" on page 585
- ":POWer:CLResponse:APPLy" on page 586
- ":POWer:CLResponse:DATA" on page 587
- ":POWer:CLResponse:FREQuency:MODE" on page 592
- ":POWer:CLResponse:FREQuency:STARt" on page 594
- ":POWer:CLResponse:FREQuency:STOP" on page 595
- ":POWer:CLResponse:PPDecade" on page 596
- ":POWer:CLResponse:SOURce:INPut" on page 597
- ":POWer:CLResponse:SOURce:OUTPut" on page 598
- ":POWer:CLResponse:WGEN:LOAD" on page 600
- ":POWer:CLResponse:WGEN:VOLTage" on page 601

:POWer:DESKew

N (see page 1354)

Command Syntax `:POWer:DESKew`

The :POWer:DESKew command launches the auto deskew process on the oscilloscope.

Before sending this command:

1. Demagnetize and zero-adjust the current probe.

   Refer to the current probe's documentation for instructions on how to do this.
2. Make connections to the U1880A deskew fixture as described in the oscilloscope's connection dialog or in the *DSOX4PWR Power Measurement Application User's Guide*.
3. Make sure the voltage probe and current probe channels are specified appropriately using the :POWer:SIGNals:SOURce:VOLTage1 and :POWer:SIGNals:SOURce:CURRent1 commands.

NOTE **Use the lowest attenuation setting on the high voltage differential probes whenever possible because the voltage levels on the deskew fixture are very small. Using a higher attenuation setting may yield inaccurate skew values (and affect the measurements made) because the noise level is magnified as well.**

The deskew values are saved in the oscilloscope until a factory default or secure erase is performed. The next time you run the Power Application, you can use the saved deskew values or perform the deskew again.

Generally, you need to perform the deskew again when part of the test setup changes (for example, a different probe, different oscilloscope channel, etc.) or when the ambient temperature has changed.

See Also

- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665

:POWer:EFFiciency:APPLy

(see page 1354)

Command Syntax `: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 605
- ":MEASure:EFFiciency" on page 528
- ":MEASure:IPOWer" on page 531
- ":MEASure:OPOWer" on page 534

:POWer:EFFiciency:TYPE

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:EFFiciency:TYPE?
```

The :POWer:EFFiciency:TYPE? query returns the currently specified type setting.

Return Format

```
<type><NL>

<type> ::= {DCDC | DCAC | ACDC | ACAC}
```

See Also

- ":POWer:EFFiciency:APPLy" on page 604
- ":MEASure:EFFiciency" on page 528
- ":MEASure:IPOWer" on page 531
- ":MEASure:OPOWer" on page 534

:POWer:ENABle

 (see page 1354)

Command Syntax `:POWer:ENABle {{0 | OFF} | {1 | ON}}`

The :POWer:ENABle command enables or disables power analysis.

Query Syntax `:POWer:ENABle?`

The :POWer:ENABle query returns a 1 or a 0 showing whether power analysis is enabled or disabled, respectively.

Return Format `{0 | 1}`

See Also

- Chapter 22, ":MEASure Power Commands," starting on page 519

:POWer:HARMonics:APPLy

 (see page 1354)

Command Syntax `: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 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:DATA

 (see page 1354)

Query Syntax `:POWer:HARMonics:DATA?`

The :POWer:HARMonics:DATA query returns the power harmonics results table data.

Return Format `<binary_block> ::= comma-separated data with newlines at the end of each row`

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:DISPlay

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:HARMonics:DISPlay?
```

The :POWer:HARMonics:DISPlay query returns the display setting.

Return Format

```
<display><NL>

<display> ::= {TABL | BAR | OFF}
```

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:FAILcount

 (see page 1354)

Query Syntax `: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.

Return Format `<count><NL>`

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

See Also

- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:LINE

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:HARMonics:LINE?
```

The :POWer:HARMonics:LINE query returns the line frequency setting.

Return Format

```
<frequency><NL>

<frequency> ::= {F50 | F60 | F400 | AUTO}
```

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:POWerfactor

(see page 1354)

Query Syntax `:POWer:HARMonics:POWerfactor?`

The :POWer:HARMonics:POWerfactor query returns the power factor for IEC 61000-3-2 Standard Class C power factor value.

Return Format `<value> ::= Class C power factor in NR3 format`

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:RPOWer

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:HARMonics:RPOWer?
```

The :POWer:HARMonics:RPOWer? query returns the Real Power source setting.

Return Format

```
<source><NL>

<source> ::= {MEAS | USER}
```

See Also

- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:RPOWer:USER" on page 614

:POWer:HARMonics:RPOWer:USER

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:HARMonics:RPOWer:USER?
```

The :POWer:HARMonics:RPOWer:USER? query returns the user-defined Real Power value.

Return Format

```
<value><NL>

<value> ::= Watts from 1.0 to 600.0 in NR3 format
```

See Also

- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:RPOWer" on page 613

:POWer:HARMonics:RUNCount

 (see page 1354)

Query Syntax `: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.

Return Format `<count><NL>`

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

See Also

- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:STANdard

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:HARMonics:STANdard?
```

The :POWer:HARMonics:STANdard query returns the currently set IEC 61000-3-2 standard.

Return Format

```
<class><NL>

<class> ::= {A | B | C | D}
```

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STATus" on page 617
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:STATus

(see page 1354)

Query Syntax `:POWer:HARMonics:STATus?`

The :POWer:HARMonics:STATus query returns the overall pass/fail status of the current harmonics analysis.

Return Format `<status> ::= {PASS | FAIL | UNTested}`

See Also

- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:THD" on page 618

:POWer:HARMonics:THD

 (see page 1354)

Query Syntax `:POWer:HARMonics:THD?`

The :POWer:HARMonics:THD query returns the Total Harmonics Distortion (THD) results of the current harmonics analysis.

Return Format `<value> ::= Total Harmonics Distortion in NR3 format`

See Also

- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:HARMonics:DATA" on page 608
- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:FAILcount" on page 610
- ":POWer:HARMonics:LINE" on page 611
- ":POWer:HARMonics:POWerfactor" on page 612
- ":POWer:HARMonics:RUNCount" on page 615
- ":POWer:HARMonics:STANdard" on page 616
- ":POWer:HARMonics:STATus" on page 617

:POWer:INRush:APPLy

(see page 1354)

Command Syntax `: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 622
- ":POWer:INRush:EXIT" on page 620
- ":POWer:INRush:NEXT" on page 621
- ":MEASure:PCURrent" on page 535

:POWer:INRush:EXIT

(see page 1354)

Command Syntax `:POWer:INRush:EXIT`

The :POWer:INRush:EXIT command exits (stops) the inrush current power analysis.

This command is equivalent to pressing the **Exit** softkey on the oscilloscope front panel during the analysis.

See Also

- ":POWer:INRush:APPLy" on page 619
- ":POWer:INRush:NEXT" on page 621
- ":POWer:ITYPe" on page 622

:POWer:INRush:NEXT

 (see page 1354)

Command Syntax `: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 619
- ":POWer:INRush:EXIT" on page 620
- ":POWer:ITYPe" on page 622

:POWer:ITYPe

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:ITYPe?
```

The :POWer:ITYPe? query returns the input power type setting.

Return Format

```
<type><NL>

<type> ::= {DC | AC}
```

See Also

- ":POWer:INRush:APPLy" on page 619
- ":POWer:ONOFf:APPLy" on page 626

:POWer:MODulation:APPLy

 (see page 1354)

Command Syntax `: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 624
- ":POWer:MODulation:TYPE" on page 625
- ":MEASure:VAVerage" on page 507
- ":MEASure:VRMS" on page 513
- ":MEASure:VRATio" on page 512
- ":MEASure:PERiod" on page 483
- ":MEASure:FREQuency" on page 475
- ":MEASure:PWIDth" on page 487
- ":MEASure:NWIDth" on page 479
- ":MEASure:DUTYcycle" on page 469
- ":MEASure:RISetime" on page 491
- ":MEASure:FALLtime" on page 470

:POWer:MODulation:SOURce

 (see page 1354)

Command Syntax `: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 Syntax `:POWer:MODulation:SOURce?`

The :POWer:MODulation:SOURce query returns the selected source for the modulation analysis.

Return Format `<source><NL>`

`<source> ::= {V | I}`

See Also

- ":POWer:MODulation:APPLy" on page 623
- ":POWer:MODulation:TYPE" on page 625

:POWer:MODulation:TYPE

(see page 1354)

Command Syntax

```
:POWer:MODulation:TYPE <modulation>

<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency
                  | PWIDith | 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 Syntax

```
:POWer:MODulation:TYPE?
```

The :POWer:MODulation:TYPE query returns the modulation type setting.

Return Format

```
<modulation><NL>

<modulation> ::= {VAV | ACRM | VRAT | PER | FREQ | PWID | NWID | DUTY
                  | RIS | FALL}
```

See Also

- ":POWer:MODulation:SOURce" on page 624
- ":POWer:MODulation:APPLy" on page 623
- ":MEASure:VAVerage" on page 507
- ":MEASure:VRMS" on page 513
- ":MEASure:VRATio" on page 512
- ":MEASure:PERiod" on page 483
- ":MEASure:FREQuency" on page 475
- ":MEASure:PWIDth" on page 487
- ":MEASure:NWIDth" on page 479
- ":MEASure:DUTYcycle" on page 469
- ":MEASure:RISetime" on page 491
- ":MEASure:FALLtime" on page 470

:POWer:ONOFf:APPLy

 (see page 1354)

Command Syntax `: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 662
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:THResholds" on page 630
- ":POWer:ONOFf:TEST" on page 629
- ":POWer:ONOFf:EXIT" on page 627
- ":POWer:ONOFf:NEXT" on page 628
- ":MEASure:ONTime" on page 533
- ":MEASure:OFFTime" on page 532

:POWer:ONOFf:EXIT

 (see page 1354)

Command Syntax `:POWer:ONOFf:EXIT`

The :POWer:ONOFf:EXIT command exits (stops) the turn on time/turn off time analysis.

This command is equivalent to pressing the **Exit** softkey on the oscilloscope front panel during the analysis.

See Also

- ":POWer:ONOFf:THResholds" on page 630
- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:ONOFf:NEXT" on page 628
- ":POWer:ONOFf:TEST" on page 629

:POWer:ONOFf:NEXT

 (see page 1354)

Command Syntax `: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 630
- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:ONOFf:EXIT" on page 627
- ":POWer:ONOFf:TEST" on page 629

:POWer:ONOFf:TEST

 (see page 1354)

Command Syntax `: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 Syntax `:POWer:ONOFf:TEST?`

The :POWer:ONOFf:TEST query returns the selected test type.

Return Format `{0 | 1}`

See Also

- ":POWer:ONOFf:THResholds" on page 630
- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:ONOFf:EXIT" on page 627
- ":POWer:ONOFf:NEXT" on page 628

:POWer:ONOFf:THResholds

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:ONOFf:THResholds? <type>
```

The :POWer:ONOFf:THResholds? query returns the input and output threshold settings for the turn on/turn off analysis type.

Return 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 662
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:ONOFf:TEST" on page 629
- ":POWer:ONOFf:EXIT" on page 627
- ":POWer:ONOFf:NEXT" on page 628

- ":MEASure:ONTime" on page 533
- ":MEASure:OFFTime" on page 532

:POWer:PSRR

(see page 1354)

Query Syntax `:POWer:PSRR?`

The :POWer:PSRR? query returns the Power Supply Rejection Ratio (PSRR) power analysis settings.

Return Format `<settings_string><NL>`

For example, the query returns the following string when issued after the *RST command.

```
:POW:PSRR:SOUR:INP CHAN1;OUTP CHAN2;:POW:PSRR:FREQ:STAR +100E+00;
STOP +20.000000E+06;:POW:PSRR:WGEN:VOLT +200.0E-03;LOAD FIFT
```

See Also

- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:APPLy

(see page 1354)

Command Syntax `: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.

You can use the :POWer:PSRR:TRACe command to specify whether to include gain data in the PSRR analysis results.

See Also

- "*ESR (Standard Event Status Register)" on page 181
- ":POWer:PSRR" on page 632
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:FREQuency:SINGle" on page 638
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:TRACe" on page 642
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:DATA

 (see page 1354)

Query Syntax

```
: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.

You can use the :POWer:PSRR:TRACe command to specify whether to include gain data in the PSRR analysis results.

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.

Return Format

```
<binary_block><NL>

<binary_block> ::= comma-separated data with newlines at the end of each
 row
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:FREQuency:SINGle" on page 638
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:TRACe" on page 642
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:FREQuency:MAXimum

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:FREQuency:MAXimum?
```

The :POWer:PSRR:FREQuency:MAXimum query returns the maximum sweep frequency setting.

Return Format

```
<value><NL>

<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 20000000}
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:FREQuency:MINimum

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:FREQuency:MINimum?
```

The :POWer:PSRR:FREQuency:MINimum query returns the minimum sweep frequency setting.

Return Format

```
<value><NL>

<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000}
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:FREQuency:MODE

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:FREQuency:MODE?
```

The :POWer:PSRR:FREQuency:MODE? query returns the frequency mode setting.

Return Format

```
<mode><NL>

<mode> ::= {SWE | SING}
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:SINGle" on page 638
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:FREQuency:SINGle

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:FREQuency:SINGle?
```

The :POWer:PSRR:FREQuency:SINGle query returns the single frequency setting.

Return Format

```
<value><NL>

<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000
   | 2000000}
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:PPDecade

(see page 1354)

Command Syntax

```
:POWer:PSRR:PPDecade <pts>

<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

The :POWer:PSRR:PPDecade command selects the number of frequency test points per decade (in the log scale).

Query Syntax

```
:POWer:CLResponse:PPDecade?
```

The :POWer:PSRR:PPDecade? query returns the points per decade setting.

Return Format

```
<pts><NL>

<pts> ::= {10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100}
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:SOURce:INPut

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:SOURce:INPut?
```

The :POWer:PSRR:SOURce:INPut? query returns the channel selection.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:SOURce:OUTPut

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:SOURce:OUTPut?
```

The :POWer:PSRR:SOURce:OUTPut? query returns the channel selection.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:TRACe

(see page 1354)

Command Syntax

```
:POWer:PSRR:TRACe <selection>

<selection> ::= {NONE | GAIN}
```

The :POWer:PSRR:TRACe command specifies whether to include gain (PSRR) data in the power supply rejection ratio analysis results.

NOTE This command affects the oscilloscope's front panel graphical user interface (plot and table) as well as when saving analysis data.

Query Syntax

```
:POWer:PSRR:TRACe?
```

The :POWer:PSRR:TRACe? query returns the type of data that is currently included in the PSRR analysis results, or "NONE" if the gain data is not included.

Return Format

```
<selection_list><NL>

<selection_list> ::= {"NONE" | "GAIN"}
```

See Also

- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634

:POWer:PSRR:WGEN:LOAD

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:WGEN:LOAD?
```

The :POWer:PSRR:WGEN:LOAD? query returns the waveform generator expected output load impedance setting.

Return Format

```
<impedance><NL>

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

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:VOLTage" on page 644
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:WGEN:VOLTage

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:PSRR:WGEN:VOLTage? [<range>]
```

The :POWer:PSRR:WGEN:VOLTage? query returns the waveform generator output amplitude setting(s).

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage:PROFile" on page 645

:POWer:PSRR:WGEN:VOLTage:PROFile

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:CLResponse:WGEN:VOLTage:PROFile?
```

The :POWer:PSRR:WGEN:VOLTage:PROFile? query returns the voltage profile setting.

Return Format

```
<setting><NL>

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

See Also

- ":POWer:PSRR" on page 632
- ":POWer:PSRR:APPLy" on page 633
- ":POWer:PSRR:DATA" on page 634
- ":POWer:PSRR:FREQuency:MAXimum" on page 635
- ":POWer:PSRR:FREQuency:MINimum" on page 636
- ":POWer:PSRR:FREQuency:MODE" on page 637
- ":POWer:PSRR:PPDecade" on page 639
- ":POWer:PSRR:SOURce:INPut" on page 640
- ":POWer:PSRR:SOURce:OUTPut" on page 641
- ":POWer:PSRR:WGEN:LOAD" on page 643
- ":POWer:PSRR:WGEN:VOLTage" on page 644

:POWer:QUALity:APPLy

 (see page 1354)

Command Syntax `:POWer:QUALity:APPLy`

The :POWer:QUALity:APPLy command applies the selected power quality analysis type (:POWer:QUALity:TYPE).

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 530
- ":MEASure:REAL" on page 539
- ":MEASure:APParent" on page 525
- ":MEASure:REACtive" on page 538
- ":MEASure:CRESt" on page 527
- ":MEASure:ANGLe" on page 524

:POWer:RIPPle:APPLy

(see page 1354)

Command Syntax `:POWer:RIPPle:APPLy`

The :POWer:RIPPle:APPLy command applies the output ripple analysis.

See Also

- ":MEASure:RIPPle" on page 540

:POWer:SIGNals:AUTosetup

 (see page 1354)

Command Syntax

```
:POWer:SIGNals:AUTosetup <analysis>

<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity
   | SLEW | SWITch | RDSVce}
```

The :POWer:SIGNals:AUTosetup command performs automated oscilloscope setup for the signals in the specified type of power analysis.

See Also

- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:EFFiciency:APPLy" on page 604
- ":POWer:RIPPle:APPLy" on page 647
- ":POWer:MODulation:APPLy" on page 623
- ":POWer:QUALity:APPLy" on page 646
- ":POWer:SLEW:APPLy" on page 667
- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SIGNals:CYCLes:HARMonics" on page 649
- ":POWer:SIGNals:CYCLes:QUALity" on page 650
- ":POWer:SIGNals:DURation:EFFiciency" on page 651
- ":POWer:SIGNals:DURation:MODulation" on page 652
- ":POWer:SIGNals:DURation:RIPPle" on page 655
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:CYCLes:HARMonics

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:CYCLes:HARMonics?
```

The :POWer:SIGNals:CYCLes:HARMonics query returns the number of cycles currently set.

Return Format

```
<count><NL>

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

See Also

- ":POWer:HARMonics:DISPlay" on page 609
- ":POWer:HARMonics:APPLy" on page 607
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:CYCLes:QUALity

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:CYCLes:QUALity?
```

The :POWer:SIGNals:CYCLes:QUALity query returns the number of cycles currently set.

Return Format

```
<count><NL>

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

See Also

- ":POWer:QUALity:APPLy" on page 646
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:EFFiciency

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:EFFiciency?
```

The :POWer:SIGNals:DURation:EFFiciency query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:EFFiciency:APPLy" on page 604
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:MODulation

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:MODulation?
```

The :POWer:SIGNals:DURation:MODulation query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:MODulation:APPLy" on page 623
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:ONOFf:OFF

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:ONOFf:OFF?
```

The :POWer:SIGNals:DURation:ONOFf:OFF query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 660
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 662
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:ONOFf:ON

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:ONOFf:ON?
```

The :POWer:SIGNals:DURation:ONOFf:ON query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 661
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:RIPPle

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:RIPPle?
```

The :POWer:SIGNals:DURation:RIPPle query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:RIPPle:APPLy" on page 647
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:DURation:TRANsient

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:DURation:TRANsient?
```

The :POWer:SIGNals:DURation:TRANsient query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VSTeady:TRANsient" on page 664
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:IEXPected

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:IEXPected?
```

The :POWer:SIGNals:IEXPected query returns the expected inrush current setting.

Return Format

```
<value><NL>

<value> ::= Expected current value in NR3 format
```

See Also

- ":POWer:INRush:APPLy" on page 619
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:INRush" on page 659
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:OVERshoot

(see page 1354)

Command Syntax

```
:POWer:SIGNals:OVERshoot <percent>

<percent> ::= percent of overshoot value in NR1 format
```

The :POWer:SIGNals:OVERshoot command specifies the percent of overshoot 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.

Query Syntax

```
:POWer:SIGNals:OVERshoot?
```

The :POWer:SIGNals:OVERshoot query returns the overshoot percent setting.

Return Format

```
<percent><NL>

<percent> ::= percent of overshoot value in NR1 format
```

See Also

- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:TRANsient" on page 656
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:VSTeady:TRANsient" on page 664
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VMAXimum:INRush

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VMAXimum:INRush?
```

The :POWer:SIGNals:VMAXimum:INRush query returns the expected maximum input voltage setting.

Return Format

```
<value><NL>

<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 622
- ":POWer:INRush:APPLy" on page 619
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VMAXimum:ONOFf:OFF

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VMAXimum:ONOFf:OFF?
```

The :POWer:SIGNals:VMAXimum:ONOFf:OFF query returns the expected maximum input voltage setting.

Return Format

```
<value><NL>

<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 653
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 662
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VMAXimum:ONOFf:ON

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VMAXimum:ONOFf:ON?
```

The :POWer:SIGNals:VMAXimum:ONOFf:ON query returns the expected maximum input voltage setting.

Return Format

```
<value><NL>

<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:ITYPe" on page 622
- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 654
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VSTeady:ONOFf:OFF

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VSTeady:ONOFf:OFF?
```

The :POWer:SIGNals:VSTeady:ONOFf:OFF query returns the expected steady state voltage setting.

Return Format

```
<value><NL>

<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 653
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 660
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VSTeady:ONOFf:ON

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VSTeady:ONOFf:ON?
```

The :POWer:SIGNals:VSTeady:ONOFf:ON query returns the expected steady state voltage setting.

Return Format

```
<value><NL>

<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:ONOFf:APPLy" on page 626
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 654
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 661
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:VSTeady:TRANsient

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:VSTeady:TRANsient?
```

The :POWer:SIGNals:VSTeady:TRANsient query returns the expected steady state voltage setting.

Return Format

```
<value><NL>

<value> ::= Expected steady state output Voltage value in NR3 format
```

See Also

- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:DURation:TRANsient" on page 656
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:SOURce:CURRent<i>

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:SOURce:CURRent<i>?
```

The :POWer:SIGNals:SOURce:CURRent<i> query returns the current source channel setting.

Return Format

```
<source><NL>

<source> ::= CHANnel<n>

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

See Also

- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:CYCLes:HARMonics" on page 649
- ":POWer:SIGNals:CYCLes:QUALity" on page 650
- ":POWer:SIGNals:DURation:EFFiciency" on page 651
- ":POWer:SIGNals:DURation:MODulation" on page 652
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 653
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 654
- ":POWer:SIGNals:DURation:RIPPle" on page 655
- ":POWer:SIGNals:DURation:TRANsient" on page 656
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:INRush" on page 659
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 660
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 661
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 662
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:SIGNals:VSTeady:TRANsient" on page 664
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 666

:POWer:SIGNals:SOURce:VOLTage<i>

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SIGNals:SOURce:VOLTage<i>?
```

The :POWer:SIGNals:SOURce:VOLTage<i> query returns the voltage source channel setting.

Return Format

```
<source><NL>

<source> ::= CHANnel<n>

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

See Also

- ":POWer:SIGNals:AUTosetup" on page 648
- ":POWer:SIGNals:CYCLes:HARMonics" on page 649
- ":POWer:SIGNals:CYCLes:QUALity" on page 650
- ":POWer:SIGNals:DURation:EFFiciency" on page 651
- ":POWer:SIGNals:DURation:MODulation" on page 652
- ":POWer:SIGNals:DURation:ONOFf:OFF" on page 653
- ":POWer:SIGNals:DURation:ONOFf:ON" on page 654
- ":POWer:SIGNals:DURation:RIPPle" on page 655
- ":POWer:SIGNals:DURation:TRANsient" on page 656
- ":POWer:SIGNals:IEXPected" on page 657
- ":POWer:SIGNals:OVERshoot" on page 658
- ":POWer:SIGNals:VMAXimum:INRush" on page 659
- ":POWer:SIGNals:VMAXimum:ONOFf:OFF" on page 660
- ":POWer:SIGNals:VMAXimum:ONOFf:ON" on page 661
- ":POWer:SIGNals:VSTeady:ONOFf:OFF" on page 662
- ":POWer:SIGNals:VSTeady:ONOFf:ON" on page 663
- ":POWer:SIGNals:VSTeady:TRANsient" on page 664
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 665

:POWer:SLEW:APPLy

 (see page 1354)

Command Syntax `:POWer:SLEW:APPLy`

The :POWer:SLEW:APPLy command applies the slew rate analysis.

See Also

- ":POWer:SLEW:SOURce" on page 668

:POWer:SLEW:SOURce

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SLEW:SOURce?
```

The :POWer:SLEW:SOURce query returns the selected source for the slew rate analysis.

Return Format

```
<source><NL>

<source> ::= {V | I}
```

See Also

- ":POWer:SLEW:APPLy" on page 667

:POWer:SWITch:APPLy

 (see page 1354)

Command Syntax `: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 670
- ":POWer:SWITch:IREFerence" on page 671
- ":POWer:SWITch:RDS" on page 672
- ":POWer:SWITch:VCE" on page 673
- ":POWer:SWITch:VREFerence" on page 674
- ":MEASure:ELOSs" on page 529
- ":MEASure:PLOSs" on page 536

:POWer:SWITch:CONDuction

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SWITch:CONDuction?
```

The :POWer:SWITch:CONDuction query returns the conduction calculation method.

Return Format

```
<conduction><NL>

<conduction> ::= {WAV | RDS | VCE}
```

See Also

- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SWITch:IREFerence" on page 671
- ":POWer:SWITch:RDS" on page 672
- ":POWer:SWITch:VCE" on page 673
- ":POWer:SWITch:VREFerence" on page 674

:POWer:SWITch:IREFerence

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SWITch:IREFerence?
```

The :POWer:SWITch:IREFerence query returns the current switching level percent value.

Return Format

```
<percent><NL>

<percent> ::= percent in NR1 format
```

See Also

- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SWITch:CONDuction" on page 670
- ":POWer:SWITch:RDS" on page 672
- ":POWer:SWITch:VCE" on page 673
- ":POWer:SWITch:VREFerence" on page 674

:POWer:SWITch:RDS

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SWITch:RDS?
```

The :POWer:SWITch:RDS query returns the Rds(on) value.

Return Format

```
<value><NL>

<value> ::= Rds(on) value in NR3 format
```

See Also

- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SWITch:CONDuction" on page 670
- ":POWer:SWITch:IREFerence" on page 671
- ":POWer:SWITch:VCE" on page 673
- ":POWer:SWITch:VREFerence" on page 674

:POWer:SWITch:VCE

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SWITch:VCE?
```

The :POWer:SWITch:VCE query returns the Vce(sat) value.

Return Format

```
<value><NL>

<value> ::= Vce(sat) value in NR3 format
```

See Also

- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SWITch:CONDuction" on page 670
- ":POWer:SWITch:IREFerence" on page 671
- ":POWer:SWITch:RDS" on page 672
- ":POWer:SWITch:VREFerence" on page 674

:POWer:SWITch:VREFerence

(see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:SWITch:VREFerence?
```

The :POWer:SWITch:VREFerence query returns the voltage switching level percent value.

Return Format

```
<percent><NL>

<percent> ::= percent in NR1 format
```

See Also

- ":POWer:SWITch:APPLy" on page 669
- ":POWer:SWITch:CONDuction" on page 670
- ":POWer:SWITch:IREFerence" on page 671
- ":POWer:SWITch:RDS" on page 672
- ":POWer:SWITch:VCE" on page 673

:POWer:TRANsient:APPLy

 (see page 1354)

Command Syntax `:POWer:TRANsient:APPLy`

The :POWer:TRANsient:APPLy command applies the transient analysis using the initial current and new current settings.

See Also

- ":POWer:TRANsient:EXIT" on page 676
- ":POWer:TRANsient:IINitial" on page 677
- ":POWer:TRANsient:INEW" on page 678
- ":POWer:TRANsient:NEXT" on page 679
- ":MEASure:TRESponse" on page 541

:POWer:TRANsient:EXIT

 (see page 1354)

Command Syntax `:POWer:TRANsient:EXIT`

The :POWer:TRANsient:EXIT command exits (stops) the transient analysis.

This command is equivalent to pressing the **Exit** softkey on the oscilloscope front panel during the analysis.

See Also

- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:TRANsient:IINitial" on page 677
- ":POWer:TRANsient:INEW" on page 678
- ":POWer:TRANsient:NEXT" on page 679

:POWer:TRANsient:IINitial

N (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:TRANsient:IINitial?
```

The :POWer:TRANsient:IINitial query returns the initial load current value.

Return Format

```
<value><NL>

<value> ::= Initial current value in NR3 format
```

See Also

- ":POWer:SIGNals:VSTeady:TRANsient" on page 664
- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:TRANsient:EXIT" on page 676
- ":POWer:TRANsient:INEW" on page 678
- ":POWer:TRANsient:NEXT" on page 679

:POWer:TRANsient:INEW

 (see page 1354)

Command Syntax

```
: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 Syntax

```
:POWer:TRANsient:INEW?
```

The :POWer:TRANsient:INEW query returns the new load current value.

Return Format

```
<value><NL>

<value> ::= New current value in NR3 format
```

See Also

- ":POWer:TRANsient:APPLy" on page 675
- ":POWer:TRANsient:EXIT" on page 676
- ":POWer:TRANsient:IINitial" on page 677
- ":POWer:TRANsient:NEXT" on page 679

:POWer:TRANsient:NEXT

 (see page 1354)

Command Syntax `: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 675
- ":POWer:TRANsient:EXIT" on page 676
- ":POWer:TRANsient:IINitial" on page 677
- ":POWer:TRANsient:INEW" on page 678

25 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

Table 100 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[STARt] [<file_spec>][,<column>][,<wavegen_id>] (see page 683)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <column> ::= Column in CSV file to load. Column number starts from 1. <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= WGEN1
:RECall:DBC[:STARt] [<file_name>] [,<serialbus>] (see page 684)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".dbc". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:FILename <base_name> (see page 685)	:RECall:FILename? (see page 685)	<base_name> ::= quoted ASCII string
:RECall:LDF[:STARt] [<file_name>] [,<serialbus>] (see page 686)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".ldf". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format

Table 100 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:MASK[:STARt] [<file_spec>] (see page 687)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:PWD <path_name> (see page 688)	:RECall:PWD? (see page 688)	<path_name> ::= quoted ASCII string
:RECall:SETup[:STARt] [<file_spec>] (see page 689)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>] (see page 690)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5". <data> ::= binary block data in IEEE 488.2 # format

Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups, reference waveforms, and masks.

Reporting the Setup

Use :RECall? to query setup information for the RECall subsystem.

Return Format

The following is a sample response from the :RECall? query. In this case, the query was issued following the *RST command.

```
:REC:FIL "scope_0"
```

:RECall:ARBitrary[:STARt]

N (see page 1354)

Command Syntax

```
:RECall:ARBitrary:[STARt] [<file_spec>][, <column>][, <wavegen_id>]
<file_spec> ::= {<internal_loc> | <file_name>}
<column> ::= Column in CSV file to load. Column number starts from 1.
<wavegen_id> ::= WGEN1 - specifies which wavegen
<internal_loc> ::= 0-3; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :RECall:ARBitrary:[STARt] command recalls an arbitrary waveform.

NOTE **If a file extension is provided as part of a specified <file_name>, it must be ".csv".**

For internal locations, the <column> parameter is ignored.

For external (USB storage device) files, the column parameter is optional. If no <column> parameter is entered, and it is a 2-column file, the 2nd column (assumed to be voltage) is automatically be selected. If the <column> parameter is entered, and that column does not exist in the file, the operation fails.

When recalling arbitrary waveforms (from an external USB storage device) that were not saved from the oscilloscope, be aware that the oscilloscope uses a maximum of 8192 points for an arbitrary waveform. For more efficient recalls, make sure your arbitrary waveforms are 8192 points or less.

The <wavegen_id> parameter specifies which waveform generator to recall the arbitrary waveform into.

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":RECall:PWD" on page 688
- ":SAVE:ARBitrary[:STARt]" on page 695

:RECall:DBC[:STARt]

(see page 1354)

Command Syntax

```
:RECall:DBC[:STARt] [<file_name>] [, <serialbus>]

<file_name> ::= quoted ASCII string

<serialbus> ::= {SBUS<n>}

<n> ::= 1 to (# of serial bus) in NR1 format
```

The :RECall:DBC[:STARt] command loads a CAN DBC (communication database) symbolic data file into the oscilloscope.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".dbc".

The <serialbus> parameter specifies which serial decode waveform the CAN symbolic data will be loaded for.

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 772
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 773
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 774
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 977
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 978
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 979

:RECall:FILename

(see page 1354)

Command Syntax

```
:RECall:FILename <base_name>
<base_name> ::= quoted ASCII string
```

The :RECall:FILename command specifies the source for any RECall operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:RECall:FILename?
```

The :RECall:FILename? query returns the current RECall filename.

Return Format

```
<base_name><NL>
<base_name> ::= quoted ASCII string
```

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:SETup[:STARt]" on page 689
- ":SAVE:FILename" on page 696

:RECall:LDF[:STARt]

(see page 1354)

Command Syntax

```
:RECall:LDF[:STARt] [<file_name>] [, <serialbus>]

<file_name> ::= quoted ASCII string

<serialbus> ::= {SBUS<n>}

<n> ::= 1 to (# of serial bus) in NR1 format
```

The :RECall:LDF[:STARt] command loads a LIN description file (LDF) symbolic data file into the oscilloscope.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".ldf".

The <serialbus> parameter specifies which serial decode waveform the LIN symbolic data will be loaded for.

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 818
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 819
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 820
- ":SEARch:SERial:LIN:MODE" on page 990
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 994
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 995
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 996

:RECall:MASK[:STARt]

(see page 1354)

Command Syntax

```
:RECall:MASK[:STARt] [<file_spec>]
<file_spec> ::= {<internal_loc> | <file_name>}
<internal_loc> ::= 0-3; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :RECall:MASK[:STARt] command recalls a mask.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".msk".

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":RECall:PWD" on page 688
- ":SAVE:MASK[:STARt]" on page 703
- ":MTESt:DATA" on page 558

:RECall:PWD

 (see page 1354)

Command Syntax

```
:RECall:PWD <path_name>

<path_name> ::= quoted ASCII string
```

The :RECall:PWD command sets the present working directory for recall operations.

Query Syntax

```
:RECall:PWD?
```

The :RECall:PWD? query returns the currently set working directory for recall operations.

Return Format

```
<path_name><NL>

<path_name> ::= quoted ASCII string
```

See Also

- "Introduction to :RECall Commands" on page 682
- ":SAVE:PWD" on page 706

:RECall:SETup[:STARt]

(see page 1354)

Command Syntax

```
:RECall:SETup[:STARt] [<file_spec>]
<file_spec> ::= {<internal_loc> | <file_name>}
<internal_loc> ::= 0-9; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :RECall:SETup[:STARt] command recalls an oscilloscope setup.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":RECall:PWD" on page 688
- ":SAVE[:SETup[:STARt]]" on page 713

:RECall:WMEMory<r>[:STARt]

 (see page 1354)

Command Syntax

```
:RECall:WMEMory<r>[:STARt] [<file_name> | <data>]

<r> ::= 1 to (# ref waveforms) in NR1 format

<file_name> ::= quoted ASCII string

<data> ::= binary block data in IEEE 488.2 # format
```

The :RECall:WMEMory<r>[:STARt] command recalls a reference waveform.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

The <data> option lets you recall a reference waveform from a local file on the controller PC (instead of from a USB storage device connected to the oscilloscope). In this case, your remote program reads data from the local ".h5" format reference waveform file in the same way that setup files are restored to the oscilloscope (see ":SYSTem:SETup" on page 1034).

See Also

- "Introduction to :RECall Commands" on page 682
- ":RECall:FILename" on page 685
- ":SAVE:WMEMory[:STARt]" on page 720

26 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "Introduction to :SAVE Commands" on page 694.

Table 101 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [<file_spec>][,<wavegen_id>] (see page 695)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= WGEN1
:SAVE:FILename <base_name> (see page 696)	:SAVE:FILename? (see page 696)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 697)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 698)	:SAVE:IMAGe:FACTors? (see page 698)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 699)	:SAVE:IMAGe:FORMat? (see page 699)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 700)	:SAVE:IMAGe:INKSaver? (see page 700)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 701)	:SAVE:IMAGe:PALette? (see page 701)	<palette> ::= {COLor | GRAYscale}

Table 101 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:LISTer[:STARt] [<file_name>] (see page 702)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:STARt] [<file_spec>] (see page 703)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:MULTi[:STARt] [<file_name>] (see page 704)	n/a	<file_name> ::= quoted ASCII string
:SAVE:POWer[:STARt] [<file_name>] (see page 705)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 706)	:SAVE:PWD? (see page 706)	<path_name> ::= quoted ASCII string
:SAVE:RESults:[STARt] [<file_spec>] (see page 707)	n/a	<file_name> ::= quoted ASCII string
:SAVE:RESults:FORMat: CURSor {{0 | OFF} | {1 | ON}} (see page 708)	:SAVE:RESults:FORMat: CURSor? (see page 708)	{0 | 1}
:SAVE:RESults:FORMat: MASK {{0 | OFF} | {1 | ON}} (see page 709)	:SAVE:RESults:FORMat: MASK? (see page 709)	{0 | 1}
:SAVE:RESults:FORMat: MEASurement {{0 | OFF} | {1 | ON}} (see page 710)	:SAVE:RESults:FORMat: MEASurement? (see page 710)	{0 | 1}
:SAVE:RESults:FORMat: SEARch {{0 | OFF} | {1 | ON}} (see page 711)	:SAVE:RESults:FORMat: SEARch? (see page 711)	{0 | 1}
:SAVE:RESults:FORMat: SEGMented {{0 | OFF} | {1 | ON}} (see page 712)	:SAVE:RESults:FORMat: SEGMented? (see page 712)	{0 | 1}

Table 101 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETup[:STARt] [<file_spec>] (see page 713)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:STARt] [<file_name>] (see page 714)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 715)	:SAVE:WAVeform:FORMat? (see page 715)	<format> ::= {ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 716)	:SAVE:WAVeform:LENGth? (see page 716)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 717)	:SAVE:WAVeform:LENGth:MAX? (see page 717)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 718)	:SAVE:WAVeform:SEGMented? (see page 718)	<option> ::= {ALL | CURRent}
:SAVE:WMEMory:SOURce <source> (see page 719)	:SAVE:WMEMory:SOURce? (see page 719)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:STARt] [<file_name>] (see page 720)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Introduction to :SAVE Commands

The :SAVE subsystem provides commands to save oscilloscope setups, screen images, and data.

:SAV is an acceptable short form for :SAVE.

Reporting the Setup

Use :SAVE? to query setup information for the SAVE subsystem.

Return Format

The following is a sample response from the :SAVE? query. In this case, the query was issued following the *RST command.

```
:SAVE:FIL "";:SAVE:IMAG:AREA GRAT;FACT 0;FORM TIFF;INKS 0;PAL
MON;:SAVE:PWD "C:/setups/";:SAVE:WAV:FORM NONE;LENG 1000;SEGM CURR
```

:SAVE:ARBitrary[:STARt]

 (see page 1354)

Command Syntax

```
:SAVE:ARBitrary:[STARt] [<file_spec>][, <wavegen_id>]

<file_spec> ::= {<internal_loc> | <file_name>}

<internal_loc> ::= 0-3; an integer in NR1 format

<file_name> ::= quoted ASCII string

<wavegen_id> ::= WGEN1
```

The :SAVE:ARBitrary:[STARt] command saves the current arbitrary waveform to an internal location or a file on a USB storage device.

The <wavegen_id> parameter specifies which waveform generator to save the arbitrary waveform from.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- ":SAVE:PWD" on page 706
- ":RECall:ARBitrary[:STARt]" on page 683

:SAVE:FILename

(see page 1354)

Command Syntax

```
:SAVE:FILename <base_name>
<base_name> ::= quoted ASCII string
```

The :SAVE:FILename command specifies the source for any SAVE operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:SAVE:FILename?
```

The :SAVE:FILename? query returns the current SAVE filename.

Return Format

```
<base_name><NL>
<base_name> ::= quoted ASCII string
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE[:SETup[:STARt]]" on page 713
- ":SAVE:WAVeform[:STARt]" on page 714
- ":SAVE:PWD" on page 706
- ":RECall:FILename" on page 685

:SAVE:IMAGe[:STARt]

(see page 1354)

Command Syntax

```
:SAVE:IMAGe[:STARt] [<file_name>]
<file_name> ::= quoted ASCII string
```

The :SAVE:IMAGe[:STARt] command saves an image.

NOTE Be sure to set the :SAVE:IMAGe:FORMat before saving an image. If the format is NONE, the save image command will not succeed.

NOTE If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:IMAGe:FORMat, the format will be changed if the extension is a valid image file extension.

NOTE If the extension ".bmp" is used and the current :SAVE:IMAGe:FORMat is not BMP or BMP8, the format will be changed to BMP.

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe:FACTors" on page 698
- ":SAVE:IMAGe:FORMat" on page 699
- ":SAVE:IMAGe:INKSaver" on page 700
- ":SAVE:IMAGe:PALette" on page 701
- ":SAVE:FILename" on page 696

:SAVE:IMAGe:FACTors

(see page 1354)

Command Syntax

```
:SAVE:IMAGe:FACTors <factors>

<factors> ::= {{OFF | 0} | {ON | 1}}
```

The :SAVE:IMAGe:FACTors command controls whether the oscilloscope factors are output along with the image.

NOTE Factors are written to a separate file with the same path and base name but with the ".txt" extension.

Query Syntax

```
:SAVE:IMAGe:FACTors?
```

The :SAVE:IMAGe:FACTors? query returns a flag indicating whether oscilloscope factors are output along with the image.

Return Format

```
<factors><NL>

<factors> ::= {0 | 1}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE:IMAGe:FORMat" on page 699
- ":SAVE:IMAGe:INKSaver" on page 700
- ":SAVE:IMAGe:PALette" on page 701

:SAVE:IMAGe:FORMat

(see page 1354)

Command Syntax

```
:SAVE:IMAGe:FORMat <format>

<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}
```

The :SAVE:IMAGe:FORMat command sets the image format type.

Query Syntax

```
:SAVE:IMAGe:FORMat?
```

The :SAVE:IMAGe:FORMat? query returns the selected image format type.

Return Format

```
<format><NL>

<format> ::= {BMP | BMP8 | PNG | NONE}
```

When NONE is returned, it indicates that a waveform data file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE:IMAGe:FACTors" on page 698
- ":SAVE:IMAGe:INKSaver" on page 700
- ":SAVE:IMAGe:PALette" on page 701
- ":SAVE:WAVeform:FORMat" on page 715

:SAVE:IMAGe:INKSaver

(see page 1354)

Command Syntax

```
:SAVE:IMAGe:INKSaver <value>

<value> ::= {{OFF | 0} | {ON | 1}}
```

The :SAVE:IMAGe:INKSaver command controls whether the graticule colors are inverted or not.

Query Syntax

```
:SAVE:IMAGe:INKSaver?
```

The :SAVE:IMAGe:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE:IMAGe:FACTors" on page 698
- ":SAVE:IMAGe:FORMat" on page 699
- ":SAVE:IMAGe:PALette" on page 701

:SAVE:IMAGe:PALette

N (see page 1354)

Command Syntax

```
:SAVE:IMAGe:PALette <palette>

<palette> ::= {COLor | GRAYscale}
```

The :SAVE:IMAGe:PALette command sets the image palette color.

Query Syntax

```
:SAVE:IMAGe:PALette?
```

The :SAVE:IMAGe:PALette? query returns the selected image palette color.

Return Format

```
<palette><NL>

<palette> ::= {COL | GRAY}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE:IMAGe:FACTors" on page 698
- ":SAVE:IMAGe:FORMat" on page 699
- ":SAVE:IMAGe:INKSaver" on page 700

:SAVE:LISTer[:STARt]

(see page 1354)

Command Syntax

```
:SAVE:LISTer[:STARt] [<file_name>]
<file_name> ::= quoted ASCII string
```

The :SAVE:LISTer[:STARt] command saves the Lister display data to a file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- Chapter 19, ":LISTer Commands," starting on page 407

:SAVE:MASK[:STARt]

 (see page 1354)

Command Syntax

```
:SAVE:MASK[:STARt] [<file_spec>]
<file_spec> ::= {<internal_loc> | <file_name>}
<internal_loc> ::= 0-3; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :SAVE:MASK[:STARt] command saves a mask.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".msk".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- ":SAVE:PWD" on page 706
- ":RECall:MASK[:STARt]" on page 687
- ":MTESt:DATA" on page 558

:SAVE:MULTi[:STARt]

 (see page 1354)

Command Syntax

```
:SAVE:MULTi[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string
```

The :SAVE:MULTi[:STARt] command saves multi-channel waveform data to a file. This file can be opened by the N8900A Infiniium Offline oscilloscope analysis software.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- ":SAVE:PWD" on page 706

:SAVE:POWer[:STARt]

 (see page 1354)

Command Syntax

```
:SAVE:POWer[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string
```

The :SAVE:POWer[:STARt] command saves the power measurement application's current harmonics analysis results to a file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".csv".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- Chapter 24, ":POWer Commands," starting on page 577

:SAVE:PWD

 (see page 1354)

Command Syntax

```
:SAVE:PWD <path_name>

<path_name> ::= quoted ASCII string
```

The :SAVE:PWD command sets the present working directory for save operations.

Query Syntax

```
:SAVE:PWD?
```

The :SAVE:PWD? query returns the currently set working directory for save operations.

Return Format

```
<path_name><NL>

<path_name> ::= quoted ASCII string
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- ":RECall:PWD" on page 688

:SAVE:RESults:[STARt]

(see page 1354)

Command Syntax

```
:SAVE:RESults:[STARt] [<file_spec>]

<file_name> ::= quoted ASCII string
```

The :SAVE:RESults:[STARt] command saves analysis results to a comma-separated values (*.csv) file on a USB storage device.

Use the :SAVE:RESults:FORMat commands to specify the analysis types whose results are saved to the file.

When multiple types of analysis results are selected, they are all saved to the same file and separated by a blank line.

See Also

- ":SAVE:RESults:FORMat:CURSor" on page 708
- ":SAVE:RESults:FORMat:MASK" on page 709
- ":SAVE:RESults:FORMat:MEASurement" on page 710
- ":SAVE:RESults:FORMat:SEARch" on page 711
- ":SAVE:RESults:FORMat:SEGMented" on page 712

:SAVE:RESults:FORMat:CURSor

(see page 1354)

Command Syntax `:SAVE:RESults:FORMat:CURSor {{0 | OFF} | {1 | ON}}`

The :SAVE:RESults:FORMat:CURSor command specifies whether cursor values will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax `:SAVE:RESults:FORMat:CURSor?`

The :SAVE:RESults:FORMat:CURSor? query returns whether cursor values will be included when analysis results are saved.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":SAVE:RESults:[STARt]" on page 707
- ":SAVE:RESults:FORMat:MASK" on page 709
- ":SAVE:RESults:FORMat:MEASurement" on page 710
- ":SAVE:RESults:FORMat:SEARch" on page 711
- ":SAVE:RESults:FORMat:SEGMented" on page 712

:SAVE:RESults:FORMat:MASK

(see page 1354)

Command Syntax

```
:SAVE:RESults:FORMat:MASK {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:MASK command specifies whether mask statistics will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:MASK?
```

The :SAVE:RESults:FORMat:MASK? query returns whether mask statistics will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults:[STARt]" on page 707
- ":SAVE:RESults:FORMat:CURSor" on page 708
- ":SAVE:RESults:FORMat:MEASurement" on page 710
- ":SAVE:RESults:FORMat:SEARch" on page 711
- ":SAVE:RESults:FORMat:SEGMented" on page 712

:SAVE:RESults:FORMat:MEASurement

 (see page 1354)

Command Syntax

```
:SAVE:RESults:FORMat:MEASurement {{0 | OFF} | {1 | ON}}
```

The :SAVE:RESults:FORMat:MEASurement command specifies whether measurement results will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax

```
:SAVE:RESults:FORMat:MEASurement?
```

The :SAVE:RESults:FORMat:MEASurement? query returns whether measurement results will be included when analysis results are saved.

Return Format

```
<off_on><NL>
```

```
{0 | 1}
```

See Also

- ":SAVE:RESults:[STARt]" on page 707
- ":SAVE:RESults:FORMat:CURSor" on page 708
- ":SAVE:RESults:FORMat:MASK" on page 709
- ":SAVE:RESults:FORMat:SEARch" on page 711
- ":SAVE:RESults:FORMat:SEGMented" on page 712

:SAVE:RESults:FORMat:SEARch

N (see page 1354)

Command Syntax `:SAVE:RESults:FORMat:SEARch {{0 | OFF} | {1 | ON}}`

The :SAVE:RESults:FORMat:SEARch command specifies whether found search event times will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax `:SAVE:RESults:FORMat:SEARch?`

The :SAVE:RESults:FORMat:SEARch? query returns whether found search event times will be included when analysis results are saved.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":SAVE:RESults:[STARt]" on page 707
- ":SAVE:RESults:FORMat:CURSor" on page 708
- ":SAVE:RESults:FORMat:MASK" on page 709
- ":SAVE:RESults:FORMat:MEASurement" on page 710
- ":SAVE:RESults:FORMat:SEGMented" on page 712

:SAVE:RESults:FORMat:SEGMented

(see page 1354)

Command Syntax `:SAVE:RESults:FORMat:SEGMented {{0 | OFF} | {1 | ON}}`

The :SAVE:RESults:FORMat:SEGMented command specifies whether segmented memory acquisition times will be included when analysis results are saved.

Analysis results are saved using the :SAVE:RESults:[STARt] command.

Other :SAVE:RESults:FORMat commands specify whether other types of analysis results are also saved.

When multiple types of analysis results are saved, they are all saved to the same file and separated by a blank line.

Query Syntax `:SAVE:RESults:FORMat:SEGMented?`

The :SAVE:RESults:FORMat:SEGMented? query returns whether segmented memory acquisition times will be included when analysis results are saved.

Return Format `<off_on><NL>`

`{0 | 1}`

See Also

- ":SAVE:RESults:[STARt]" on page 707
- ":SAVE:RESults:FORMat:CURSor" on page 708
- ":SAVE:RESults:FORMat:MASK" on page 709
- ":SAVE:RESults:FORMat:MEASurement" on page 710
- ":SAVE:RESults:FORMat:SEARch" on page 711

:SAVE[:SETup[:STARt]]

(see page 1354)

Command Syntax

```
:SAVE[:SETup[:STARt]] [<file_spec>]
<file_spec> ::= {<internal_loc> | <file_name>}
<internal_loc> ::= 0-9; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :SAVE[:SETup[:STARt]] command saves an oscilloscope setup.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:FILename" on page 696
- ":SAVE:PWD" on page 706
- ":RECall:SETup[:STARt]" on page 689

:SAVE:WAVeform[:STARt]

(see page 1354)

Command Syntax

```
:SAVE:WAVeform[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string
```

The :SAVE:WAVeform[:STARt] command saves oscilloscope waveform data to a file.

NOTE Be sure to set the :SAVE:WAVeform:FORMat before saving waveform data. If the format is NONE, the save waveform command will not succeed.

NOTE If a file extension is provided as part of a specified <file_name>, and it does not match the extension expected by the format specified in :SAVE:WAVeform:FORMat, the format will be changed if the extension is a valid waveform file extension.

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WAVeform:FORMat" on page 715
- ":SAVE:WAVeform:LENGth" on page 716
- ":SAVE:FILename" on page 696
- ":RECall:SETup[:STARt]" on page 689

:SAVE:WAVeform:FORMat

(see page 1354)

Command Syntax

```
:SAVE:WAVeform:FORMat <format>

<format> ::= {ASCiixy | CSV | BINary}
```

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ASCiixy – creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV – creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINary – creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

Query Syntax

```
:SAVE:WAVeform:FORMat?
```

The :SAVE:WAVeform:FORMat? query returns the selected waveform data format type.

Return Format

```
<format><NL>

<format> ::= {ASC | CSV | BIN | NONE}
```

When NONE is returned, it indicates that an image file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WAVeform[:STARt]" on page 714
- ":SAVE:WAVeform:LENGth" on page 716
- ":SAVE:IMAGe:FORMat" on page 699

:SAVE:WAVeform:LENGth

 (see page 1354)

Command Syntax

```
:SAVE:WAVeform:LENGth <length>

<length> ::= 100 to max. length; an integer in NR1 format
```

When the :SAVE:WAVeform:LENGth:MAX setting is OFF, the :SAVE:WAVeform:LENGth command sets the waveform data length (that is, the number of points saved).

When the :SAVE:WAVeform:LENGth:MAX setting is ON, the :SAVE:WAVeform:LENGth setting has no effect.

Query Syntax

```
:SAVE:WAVeform:LENGth?
```

The :SAVE:WAVeform:LENGth? query returns the current waveform data length setting.

Return Format

```
<length><NL>

<length> ::= 100 to max. length; an integer in NR1 format
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WAVeform:LENGth:MAX" on page 717
- ":SAVE:WAVeform[:STARt]" on page 714
- ":WAVeform:POINts" on page 1170
- ":SAVE:WAVeform:FORMat" on page 715

:SAVE:WAVeform:LENGth:MAX

 (see page 1354)

Command Syntax

```
:SAVE:WAVeform:LENGth:MAX <setting>

<setting> ::= {{OFF | 0} | {ON | 1}}
```

The :SAVE:WAVeform:LENGth:MAX command specifies whether maximum number of waveform data points is saved.

When OFF, the :SAVE:WAVeform:LENGth command specifies the number of waveform data points saved.

Query Syntax

```
:SAVE:WAVeform:LENGth:MAX?
```

The :SAVE:WAVeform:LENGth:MAX? query returns the current setting.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WAVeform[:STARt]" on page 714
- ":SAVE:WAVeform:LENGth" on page 716

:SAVE:WAVeform:SEGMented

(see page 1354)

Command Syntax

```
:SAVE:WAVeform:SEGMented <option>

<option> ::= {ALL | CURRent}
```

When segmented memory is used for acquisitions, the :SAVE:WAVeform:SEGMented command specifies which segments are included when the waveform is saved:

- ALL – all acquired segments are saved.
- CURRent – only the currently selected segment is saved.

Query Syntax

```
:SAVE:WAVeform:SEGMented?
```

The :SAVE:WAVeform:SEGMented? query returns the current segmented waveform save option setting.

Return Format

```
<option><NL>

<option> ::= {ALL | CURR}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WAVeform[:STARt]" on page 714
- ":SAVE:WAVeform:FORMat" on page 715
- ":SAVE:WAVeform:LENGth" on page 716

:SAVE:WMEMory:SOURce

(see page 1354)

Command Syntax

```
:SAVE:WMEMory:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :SAVE:WMEMory:SOURce command selects the source to be saved as a reference waveform file.

NOTE Only ADD or SUBtract math operations can be saved as reference waveforms.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:SAVE:WMEMory:SOURce?
```

The :SAVE:WMEMory:SOURce? query returns the source to be saved as a reference waveform file.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}
```

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WMEMory[:STARt]" on page 720
- ":RECall:WMEMory<r>[:STARt]" on page 690

:SAVE:WMEMory[:STARt]

(see page 1354)

Command Syntax

```
:SAVE:WMEMory[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string
```

The :SAVE:WMEMory[:STARt] command saves oscilloscope waveform data to a reference waveform file.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".h5".

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:WMEMory:SOURce" on page 719
- ":RECall:WMEMory<r>[:STARt]" on page 690

27 :SBUS<n> Commands

Control the modes and parameters for each serial bus decode/trigger type. See:

Introduction to :SBUS<n> Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE **These commands are only valid on oscilloscope models when a serial decode option has been licensed.**

The following serial bus decode/trigger types are available (see ":TRIGger:MODE" on page 1066).

- **CAN (Controller Area Network) triggering**– will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. You can trigger on CAN data and identifier patterns and you can set the bit sample point.
- **CXPI triggering** (with CXPI license) – lets you trigger on CXPI serial data.

- **IIC (Inter-IC bus) triggering**— consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering**— will trigger on LIN sync break at the beginning of a message frame.You can trigger on Sync Break, Frame IDs, or Frame IDs and Data.
- **UART/RS-232 triggering** (with COMP license) — lets you trigger on RS-232 serial data.
- **SENT triggering** (with SENSOR license) — lets you trigger on SENT serial data.
- **USB PD triggering** (with USBPD license) — lets you trigger on USB PD serial data.

Reporting the Setup

Use :SBUS<n>? to query setup information for the :SBUS<n> subsystem.

Return Format

The following is a sample response from the :SBUS1? query. In this case, the query was issued following a *RST command.

```
:SBUS1:DISP 0;MODE IIC;:SBUS1:IIC:ASIZ BIT7;:SBUS1:IIC:TRIG:TYPE
STAR;QUAL EQU;:SBUS1:IIC:SOUR:CLOC CHAN1;DATA
CHAN2;:SBUS1:IIC:TRIG:PATT:ADDR -1;DATA -1;DATA2 -1
```

General :SBUS<n> Commands

Table 102 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 724)	:SBUS<n>:DISPlay? (see page 724)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 725)	:SBUS<n>:MODE? (see page 725)	<mode> ::= {A429 | CAN | CXPI | IIC | LIN | M1553 | MANChester | NRZ | SENT | UART | USBPd}

:SBUS<n>:DISPlay

(see page 1354)

Command Syntax

```
:SBUS<n>:DISPlay <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS<n>:DISPlay command turns displaying of the serial decode bus on or off.

NOTE This command is only valid when a serial decode option has been licensed.

NOTE Two I2S buses or two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Query Syntax

```
:SBUS<n>:DISPlay?
```

The :SBUS<n>:DISPlay? query returns the current display setting of the serial decode bus.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":CHANnel<n>:DISPlay" on page 277
- ":VIEW" on page 239
- ":BLANk" on page 210
- ":STATus" on page 236

:SBUS<n>:MODE

(see page 1354)

Command Syntax

```
:SBUS<n>:MODE <mode>

<mode> ::= {A429 | CAN | CXPI | IIC | LIN | M1553 | MANChester | NRZ
   | SENT | UART | USBPd}
```

The :SBUS<n>:MODE command determines the decode mode for the serial bus.

NOTE This command is only valid when a serial decode option has been licensed.

Query Syntax

```
:SBUS<n>:MODE?
```

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Return Format

```
<mode><NL>

<mode> ::= {A429 | CAN | IIC | LIN | M1553 | MANC | NRZ | SENT | UART
   | USBP | NONE}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:A429 Commands" on page 726
- ":SBUS<n>:CAN Commands" on page 745
- ":SBUS<n>:IIC Commands" on page 792
- ":SBUS<n>:LIN Commands" on page 802
- ":SBUS<n>:M1553 Commands" on page 821
- ":SBUS<n>:SENT Commands" on page 866
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:A429 Commands

NOTE These commands are valid when the MIL-STD-1553 and ARINC 429 triggering and serial decode option (AERO license) has been installed.

Table 103 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUTosetup (see page 728)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 729)	:SBUS<n>:A429:BASE? (see page 729)	<base> ::= {BINary | HEX}
:SBUS<n>:A429:BAUDrate <baudrate> (see page 730)	:SBUS<n>:A429:BAUDrate? (see page 730)	<baudrate> ::= integer from 10000 to 1000000
n/a	:SBUS<n>:A429:COUNt:ERRor? (see page 731)	<error_count> ::= integer in NR1 format
:SBUS<n>:A429:COUNt:RESet (see page 732)	n/a	n/a
n/a	:SBUS<n>:A429:COUNt:WORD? (see page 733)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMat <format> (see page 734)	:SBUS<n>:A429:FORMat? (see page 734)	<format> ::= {LDSDi | LDSSm | LDATa}
:SBUS<n>:A429:SIGNal <signal> (see page 735)	:SBUS<n>:A429:SIGNal? (see page 735)	<signal> ::= {A | B | DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 736)	:SBUS<n>:A429:SOURce? (see page 736)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEed <speed> (see page 737)	:SBUS<n>:A429:SPEed? (see page 737)	<speed> ::= {LOW | HIGH | USER}

Table 103 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :LABel <value> (see page 738)	:SBUS<n>:A429:TRIGger :LABel? (see page 738)	<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 or "0xXX" (don't care) <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SBUS<n>:A429:TRIGger :PATTern:DATA <string> (see page 739)	:SBUS<n>:A429:TRIGger :PATTern:DATA? (see page 739)	<string> ::= "nn...n" where n ::= {0 | 1 | X}, length depends on FORMat
:SBUS<n>:A429:TRIGger :PATTern:SDI <string> (see page 740)	:SBUS<n>:A429:TRIGger :PATTern:SDI? (see page 740)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :PATTern:SSM <string> (see page 741)	:SBUS<n>:A429:TRIGger :PATTern:SSM? (see page 741)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :RANGe <min>,<max> (see page 742)	:SBUS<n>:A429:TRIGger :RANGe? (see page 742)	<min> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <max> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SBUS<n>:A429:TRIGger :TYPE <condition> (see page 743)	:SBUS<n>:A429:TRIGger :TYPE? (see page 743)	<condition> ::= {WSTArt | WSTOp | LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors | LRANge | ABITs | AOBits | AZBits}

:SBUS<n>:A429:AUTosetup

(see page 1354)

Command Syntax `:SBUS<n>:A429:AUTosetup`

The :SBUS<n>:A429:AUTosetup command automatically sets these options for decoding and triggering on ARINC 429 signals:

- High Trigger Threshold: 3.0 V.
- Low Trigger Threshold: -3.0 V.
- Noise Reject: Off.
- Probe Attenuation: 10.0.
- Vertical Scale: 4 V/div.
- Serial Decode: On.
- Base (:SBUS<n>:A429:BASE): HEX.
- Word Format (:SBUS<n>:A429:FORMat): LDSDi (Label/SDI/Data/SSM).
- Trigger: the specified serial bus (n of SBUS<n>).
- Trigger Mode (:SBUS<n>:A429:TRIGger:TYPE): WSTArt.

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:A429:BASE" on page 729
- ":SBUS<n>:A429:FORMat" on page 734
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429 Commands" on page 726

:SBUS<n>:A429:BASE

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:BASE <base>

<base> ::= {BINary | HEX}
```

The :SBUS<n>:A429:BASE command selects between hexadecimal and binary display of the decoded data.

The BASE command has no effect on the SDI and SSM fields, which are always displayed in binary, nor the Label field, which is always displayed in octal.

Query Syntax

```
:SBUS<n>:A429:BASE?
```

The :SBUS<n>:A429:BASE? query returns the current ARINC 429 base setting.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:FORMat" on page 734

:SBUS<n>:A429:BAUDrate

 (see page 1354)

Command Syntax

```
:SBUS<n>:A429:BAUDrate <baudrate>

<baudrate> ::= integer from 10000 to 1000000
```

When a user-defined baud rate is selected (with the ":SBUS<n>:A429:SPEed USER" command), the :SBUS<n>:A429:BAUDrate command specifies the user-defined baud rate. The baud rate can be set in 100 b/s increments between 10000 and 100000 and in 1000 b/s ingrements between 100000 and 1000000.

Query Syntax

```
:SBUS<n>:A429:BAUDrate?
```

The :SBUS<n>:A429:BAUDrate? query returns the user-defined baud rate setting.

Return Format

```
<baudrate><NL>
```

See Also

- ":SBUS<n>:A429:SPEed" on page 737

:SBUS<n>:A429:COUNt:ERRor

(see page 1354)

Query Syntax `:SBUS<n>:A429:COUNt:ERRor?`

Returns the error count.

Return Format `<error_count><NL>`

`<error_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:A429:COUNt:RESet" on page 732
- ":SBUS<n>:A429:COUNt:WORD" on page 733
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429 Commands" on page 726

:SBUS<n>:A429:COUNt:RESet

(see page 1354)

Command Syntax `:SBUS<n>:A429:COUNt:RESet`

Resets the word and error counters.

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:A429:COUNt:WORD" on page 733
- ":SBUS<n>:A429:COUNt:ERRor" on page 731
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429 Commands" on page 726

:SBUS<n>:A429:COUNt:WORD

N (see page 1354)

Query Syntax `:SBUS<n>:A429:COUNt:WORD?`

Returns the word count.

Return Format `<word_count><NL>`

`<word_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:A429:COUNt:RESet" on page 732
- ":SBUS<n>:A429:COUNt:ERRor" on page 731
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429 Commands" on page 726

:SBUS<n>:A429:FORMat

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:FORMat <format>

<format> ::= {LDSDi | LDSSm | LDATa}
```

The :SBUS<n>:A429:FORMat command specifies the word decode format:

- LDSDi:
  - Label - 8 bits.
  - SDI - 2 bits.
  - Data - 19 bits.
  - SSM - 2 bits.
- LDSSm:
  - Label - 8 bits.
  - Data - 21 bits.
  - SSM - 2 bits.
- LDATa:
  - Label - 8 bits.
  - Data - 23 bits.

Query Syntax

```
:SBUS<n>:A429:FORMat?
```

The :SBUS<n>:A429:FORMat? query returns the current ARINC 429 word decode format setting.

Return Format

```
<format><NL>

<format> ::= {LDSD | LDSS | LDAT}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 739
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 740
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 741
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- ":SBUS<n>:A429:SIGNal" on page 735
- ":SBUS<n>:A429:SPEed" on page 737
- ":SBUS<n>:A429:BASE" on page 729
- ":SBUS<n>:A429:SOURce" on page 736

:SBUS<n>:A429:SIGNal

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:SIGNal <signal>

<signal> ::= {A | B | DIFFerential}
```

The :SBUS<n>:A429:SIGNal command specifies the signal type:

- A – Line A (non-inverted).
- B – Line B (inverted).
- DIFFerential – Differential (A-B).

Query Syntax

```
:SBUS<n>:A429:SIGNal?
```

The :SBUS<n>:A429:SIGNal? query returns the current ARINC 429 signal type setting.

Return Format

```
<signal><NL>

<signal> ::= {A | B | DIFF}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:FORMat" on page 734
- ":SBUS<n>:A429:SPEed" on page 737
- ":SBUS<n>:A429:SOURce" on page 736

:SBUS<n>:A429:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:A429:SOURce command sets the source of the ARINC 429 signal.

Query Syntax

```
:SBUS<n>:A429:SOURce?
```

The :SBUS<n>:A429:SOURce? query returns the currently set source of the ARINC 429 signal.

Use the :TRIGger:LEVel:HIGH and :TRIGger:LEVel:LOW commands to set the threshold levels for the selected source.

Return Format

```
<source><NL>
```

See Also

- ":TRIGger:LEVel:HIGH" on page 1064
- ":TRIGger:LEVel:LOW" on page 1065
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- ":SBUS<n>:A429:SIGNal" on page 735
- ":SBUS<n>:A429:SPEed" on page 737
- ":SBUS<n>:A429:FORMat" on page 734
- "Introduction to :TRIGger Commands" on page 1053

:SBUS<n>:A429:SPEed

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:SPEed <speed>

<speed> ::= {LOW | HIGH | USER}
```

The :SBUS<n>:A429:SPEed command specifies the signal speed:

- LOW – 12.5 kb/s.
- HIGH – 100 kb/s.
- USER – lets you specify a user-defined baud rate using the :SBUS<n>:A429:BAUDrate command.

Query Syntax

```
:SBUS<n>:A429:SPEed?
```

The :SBUS<n>:A429:SPEed? query returns the current ARINC 429 signal speed setting.

Return Format

```
<speed><NL>

<speed> ::= {LOW | HIGH | USER}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:A429:BAUDrate" on page 730
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:SIGNal" on page 735
- ":SBUS<n>:A429:FORMat" on page 734
- ":SBUS<n>:A429:SOURce" on page 736

:SBUS<n>:A429:TRIGger:LABel

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:LABel <value>

<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string>
            from 0-255 or "0xXX" (don't care)

<hex> ::= #Hnn where n ::= {0,..,9 | A,..,F}

<octal> ::= #Qnnn where n ::= {0,..,7}

<string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
```

The :SBUS<n>:A429:TRIGger:LABel command defines the ARINC 429 label value when labels are used in the selected trigger type.

To set the label value to don't cares (0xXX), set the value to -1.

Query Syntax

```
:SBUS<n>:A429:TRIGger:LABel?
```

The :SBUS<n>:A429:TRIGger:LABel? query returns the current label value in decimal format.

Return Format

```
<value><NL> in decimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743

:SBUS<n>:A429:TRIGger:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}, length depends on FORMat
```

The :SBUS<n>:A429:TRIGger:PATTern:DATA command defines the ARINC 429 data pattern resource according to the string parameter. This pattern controls the data pattern searched for in each ARINC 429 word.

NOTE If more bits are sent for <string> than specified by the :SBUS<n>:A429:FORMat command, the most significant bits will be truncated.

Query Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:DATA?
```

The :SBUS<n>:A429:TRIGger:PATTern:DATA? query returns the current settings of the specified ARINC 429 data pattern resource in the binary string format.

Return Format

```
<string><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 740
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 741

:SBUS<n>:A429:TRIGger:PATTern:SDI

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:SDI <string>

<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
```

The :SBUS<n>:A429:TRIGger:PATTern:SDI command defines the ARINC 429 two-bit SDI pattern resource according to the string parameter. This pattern controls the SDI pattern searched for in each ARINC 429 word.

The specified SDI is only used if the :SBUS<n>:A429:FORMat includes the SDI field.

Query Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:SDI?
```

The :SBUS<n>:A429:TRIGger:PATTern:SDI? query returns the current settings of the specified ARINC 429 two-bit SDI pattern resource in the binary string format.

Return Format

```
<string><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:FORMat" on page 734
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 739
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 741

:SBUS<n>:A429:TRIGger:PATTern:SSM

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:SSM <string>

<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
```

The :SBUS<n>:A429:TRIGger:PATTern:SSM command defines the ARINC 429 two-bit SSM pattern resource according to the string parameter. This pattern controls the SSM pattern searched for in each ARINC 429 word.

The specified SSM is only used if the :SBUS<n>:A429:FORMat includes the SSM field.

Query Syntax

```
:SBUS<n>:A429:TRIGger:PATTern:SSM?
```

The :SBUS<n>:A429:TRIGger:PATTern:SSM? query returns the current settings of the specified ARINC 429 two-bit SSM pattern resource in the binary string format.

Return Format

```
<string><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:FORMat" on page 734
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 739
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 740

:SBUS<n>:A429:TRIGger:RANGe

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:RANGe <min>,<max>

<min> ::= 8-bit integer in decimal, <hex>, <octal>, or <string>
          from 0-255

<max> ::= 8-bit integer in decimal, <hex>, <octal>, or <string>
          from 0-255

<hex> ::= #Hnn where n ::= {0,..,9 | A,..,F}

<octal> ::= #Qnnn where n ::= {0,..,7}

<string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
```

The :SBUS<n>:A429:TRIGger:RANGe command defines a range of ARINC 429 label values. This range is used when the LRANge trigger type is selected.

Query Syntax

```
:SBUS<n>:A429:TRIGger:RANGe?
```

The :SBUS<n>:A429:TRIGger:RANGe? query returns the current label values in decimal format.

Return Format

```
<min>,<max><NL> in decimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:TRIGger:TYPE" on page 743

:SBUS<n>:A429:TRIGger:TYPE

N (see page 1354)

Command Syntax

```
:SBUS<n>:A429:TRIGger:TYPE <condition>

<condition> ::= {WSTArt | WSTOp | LABel | LBITs | PERRor | WERRor
                 | GERRor | WGERrors | ALLerrors | LRANge | ABITs
                 | AOBits | AZBits}
```

The :SBUS<n>:A429:TRIGger command sets the ARINC 429 trigger on condition:

- WSTArt – triggers on the start of a word.
- WSTOp – triggers at the end of a word.
- LABel – triggers on the specified label value.
- LBITs – triggers on the label and the other word fields as specified.
- LRANge – triggers on a label within a min/max range.
- PERRor – triggers on words with a parity error.
- WERRor – triggers on an intra-word coding error.
- GERRor – triggers on an inter-word gap error.
- WGERrors – triggers on either a Word or Gap Error.
- ALLerrors – triggers on any of the above errors.
- ABITs – triggers on any bit, which will therefore form an eye diagram.
- AZBits – triggers on any bit with a value of zero.
- AOBits – triggers on any bit with a value of one.

Query Syntax

```
:SBUS<n>:A429:TRIGger:TYPE?
```

The :SBUS<n>:A429:TRIGger:TYPE? query returns the current ARINC 429 trigger on condition.

Return Format

```
<condition><NL>

<condition> ::= {WSTA | WSTO | LAB | LBIT | PERR | WERR | GERR | WGER
                 | ALL | LRAN | ABIT | AOB | AZB}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:A429:TRIGger:LABel" on page 738
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 739
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 740
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 741
- ":SBUS<n>:A429:TRIGger:RANGe" on page 742

- ":SBUS<n>:A429:SOURce" on page 736

:SBUS<n>:CAN Commands

NOTE These commands are valid when the automotive CAN and LIN serial decode option (AUTO license) has been installed.

Table 104 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 748)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 749)	<frame_count> ::= 0 in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 750)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:SPEC? (see page 751)	<spec_error_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 752)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 753)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPlay <type> (see page 754)	:SBUS<n>:CAN:DISPlay? (see page 754)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:CAN:FDSPoint <value> (see page 755)	:SBUS<n>:CAN:FDSPoint? (see page 755)	<value> ::= even numbered percentages from 30 to 90 in NR3 format.
:SBUS<n>:CAN:FDSTandard <std> (see page 756)	:SBUS<n>:CAN:FDSTandard? (see page 756)	<std> ::= {ISO | NISO}
:SBUS<n>:CAN:SAMPlepoint <percent> (see page 757)	:SBUS<n>:CAN:SAMPlepoint? (see page 757)	<percent> ::= 30.0 to 90.0 in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 758)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 758)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 759)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 759)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate> (see page 760)	:SBUS<n>:CAN:SIGNal:FDBaudrate? (see page 760)	<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.

Table 104 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:SOURce <source> (see page 761)	:SBUS<n>:CAN:SOURce? (see page 761)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 762)	:SBUS<n>:CAN:TRIGger? (see page 763)	<condition> ::= {SOF | EOF | IDData | DATA | FDData | IDRemote | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive | OVERload | MESSage | MSIGnal | FDMSignal}
:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}} (see page 765)	:SBUS<n>:CAN:TRIGger:IDFilter? (see page 765)	{0 | 1}
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 766)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 766)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc> (see page 767)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? (see page 767)	<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 768)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 768)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start> (see page 769)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? (see page 769)	<start> ::= integer between 0 and 63, in NR1 format.
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 770)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 770)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value> (see page 771)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 771)	<value> ::= {STANdard | EXTended}
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name> (see page 772)	:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? (see page 772)	<name> ::= quoted ASCII string

Table 104 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name> (see page 773)	:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? (see page 773)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data> (see page 774)	:SBUS<n>:CAN:TRIGger:SYMBolic:VALue? (see page 774)	<data> ::= value in NR3 format

:SBUS<n>:CAN:COUNt:ERRor

(see page 1354)

Query Syntax `:SBUS<n>:CAN:COUNt:ERRor?`

Returns the error frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 750
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:COUNt:OVERload

N (see page 1354)

Query Syntax `:SBUS<n>:CAN:COUNt:OVERload?`

Returns the overload frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= 0 in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 750
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:COUNt:RESet

(see page 1354)

Command Syntax `:SBUS<n>:CAN:COUNt:RESet`

Resets the frame counters.

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:ERRor" on page 748
- ":SBUS<n>:CAN:COUNt:OVERload" on page 749
- ":SBUS<n>:CAN:COUNt:TOTal" on page 752
- ":SBUS<n>:CAN:COUNt:UTILization" on page 753
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:COUNt:SPEC

N (see page 1354)

Query Syntax `:SBUS<n>:CAN:COUNt:SPEC?`

Returns the Spec error (Ack + Form + Stuff + CRC errors) count.

Return Format `<spec_error_count><NL>`

`<spec_error_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 750
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:COUNt:TOTal

(see page 1354)

Query Syntax `:SBUS<n>:CAN:COUNt:TOTal?`

Returns the total frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 750
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:COUNt:UTILization

(see page 1354)

Query Syntax `:SBUS<n>:CAN:COUNt:UTILization?`

Returns the percent utilization.

Return Format `<percent><NL>`

`<percent> ::= floating-point in NR3 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 750
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN Commands" on page 745

:SBUS<n>:CAN:DISPlay

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:DISPlay <type>

<type> ::= {HEXadecimal | SYMBolic}
```

The :SBUS<n>:CAN:DISPlay command specifies, when CAN symbolic data is loaded into the oscilloscope, whether symbolic values (from the DBC file) or hexadecimal values are displayed in the decode waveform and the Lister window.

Query Syntax

```
:SBUS<n>:CAN:DISPlay?
```

The :SBUS<n>:CAN:DISPlay? query returns the CAN decode display type.

Return Format

```
<type><NL>

<type> ::= {HEX | SYMB}
```

See Also

- ":RECall:DBC[:STARt]" on page 684

:SBUS<n>:CAN:FDSPoint

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:FDSPoint <value>

<value> ::= even numbered percentages from 30 to 90 in NR3 format.
```

The :SBUS<n>:CAN:FDSPoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax

```
:SBUS<n>:CAN:FDSPoint?
```

The :SBUS<n>:CAN:FDSPoint? query returns the current CAN FD sample point setting.

Return Format

```
<value><NL>

<value> ::= even numbered percentages from 30 to 90 in NR3 format.
```

See Also

- ":SBUS<n>:CAN:SIGNal:FDBaudrate" on page 760

:SBUS<n>:CAN:FDSTandard

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:FDSTandard <std>

<std> ::= {ISO | NISO}
```

The :SBUS<n>:CAN:FDSTandard command lets you pick the standard that will be used when decoding or triggering on FD frames, ISO, or non-ISO.

This setting has no effect on the processing of non-FD (classical) frames.

Query Syntax

```
:SBUS<n>:CAN:FDSTandard?
```

The :SBUS<n>:CAN:FDSTandard? query returns the selected CAN FD frame decode standard.

Return Format

```
<std<NL>

<std> ::= {ISO | NISO}
```

See Also

- ":SBUS<n>:CAN:FDSPoint" on page 755

:SBUS<n>:CAN:SAMPlepoint

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:SAMPlepoint <percent>

<percent><NL>

<percent> ::= 30.0 to 90.0 in NR3 format
```

The :SBUS<n>:CAN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax

```
:SBUS<n>:CAN:SAMPlepoint?
```

The :SBUS<n>:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Return Format

```
<percent><NL>

<percent> ::= 30.0 to 90.0 in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:TRIGger" on page 762

:SBUS<n>:CAN:SIGNal:BAUDrate

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

The :SBUS<n>:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 4 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

You can also set the baud rate of the CAN signal to 5 Mb/s. Fractional baud rates between 4 Mb/s and 5 Mb/s are not allowed.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:BAUDrate?
```

The :SBUS<n>:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments,
               or 5000000
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 759
- ":SBUS<n>:CAN:SOURce" on page 761

:SBUS<n>:CAN:SIGNal:DEFinition

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:DEFinition <value>

<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
```

The :SBUS<n>:CAN:SIGNal:DEFinition command sets the CAN signal type when :SBUS<n>:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signals:

- CANH – the actual CAN_H differential bus signal.
- DIFH – the CAN differential (H-L) bus signal connected to an analog source channel using a differential probe.

Dominant low signals:

- CANL – the actual CAN_L differential bus signal.
- RX – the Receive signal from the CAN bus transceiver.
- TX – the Transmit signal to the CAN bus transceiver.
- DIFL – the CAN differential (L-H) bus signal connected to an analog source channel using a differential probe.
- DIFFerential – the CAN differential bus signal connected to an analog source channel using a differential probe. This is the same as DIFL.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:DEFinition?
```

The :SBUS<n>:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Return Format

```
<value><NL>

<value> ::= {CANH | CANL | RX | TX | DIFL | DIFH}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 758
- ":SBUS<n>:CAN:SOURce" on page 761
- ":SBUS<n>:CAN:TRIGger" on page 762

:SBUS<n>:CAN:SIGNal:FDBaudrate

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:SIGNal:FDBaudrate <baudrate>

<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
```

The :SBUS<n>:CAN:SIGNal:FDBaudrate command sets the CAN FD baud rate from 10 kb/s to 10 Mb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

For CAN FD, both the standard rate settings (see :SBUS<n>:CAN:SIGNal:BAUDrate) and the FD rate settings must be set correctly; otherwise, false triggers may occur.

Query Syntax

```
:SBUS<n>:CAN:SIGNal:FDBaudrate?
```

The :SBUS<n>:CAN:SIGNal:FDBaudrate? query returns the current CAN FD baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 10000 to 10000000 in 100 b/s increments.
```

See Also

- ":SBUS<n>:CAN:FDSPoint" on page 755
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 758

:SBUS<n>:CAN:SOURce

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:CAN:SOURce command sets the source for the CAN signal.

Query Syntax

```
:SBUS<n>:CAN:SOURce?
```

The :SBUS<n>:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 759

:SBUS<n>:CAN:TRIGger

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger <condition>

<condition> ::= {SOF | EOF | IDData | DATA | FDData | IDRemote
    | IDEither | ERRor | ACKerror | FORMerror | STUFferror | CRCerror
    | SPECerror | ALLerrors | BRSBit | CRCDbit | EBActive | EBPassive
    | OVERload | MESSage | MSIGnal | FDMSignal}
```

The :SBUS<n>:CAN:TRIGger command sets the CAN trigger on condition:

Condition	Front-panel name	Description	Filter by ID*
SOF	SOF - Start of Frame	Triggers at the start bit for both data and overload frames.	
EOF	EOF - End of Frame	Triggers at the end of any frame.	X
IDEither	Frame ID	Triggers on any standard CAN (data or remote) or CAN FD frame at the end of the 11- or 29-bit ID field.	
IDData	Data Frame ID (non-FD)	Triggers on standard CAN data frames at the end of the 11- or 29-bit ID field.	
DATA	Data Frame ID and Data (non-FD)	Triggers on any standard CAN data frame at the end of the last data byte defined in the trigger. The DLC of the packet must must match the number of bytes specified.	
FDData	Data Frame ID and Data (FD)	triggers on CAN FD frames at the end of the last data byte defined in the trigger. You can trigger on up to 8 bytes of data anywhere within the CAN FD data, which can be up to 64 bytes long.	
IDRemote	Remote Frame ID	Triggers on standard CAN remote frames at the end of the 11- or 29-bit ID field.	
ERRor	Error Frame	Triggers after 6 consecutive 0s while in a data frame, at the EOF.	X
ACKerror	Acknowledge Error	Triggers on the acknowledge bit if the polarity is incorrect.	X
FORMerror	Form Error	Triggers on reserved bit errors.	X
STUFferror	Stuff Error	Triggers on 6 consecutive 1s or 6 consecutive 0s, while in a non-error or non overload frame.	X

Condition	Front-panel name	Description	Filter by ID*
CRCerror	CRC Field Error	Triggers when the calculated CRC does not match the transmitted CRC. In addition, for FD frames, will also trigger if the Stuff Count is in error.	X
SPECerror	Spec Error (Ack or Form or Stuff or CRC)	Triggers on Ack, Form, Stuff, or CRC errors.	X
ALLerrors	All Errors	Triggers on all Spec errors and error frames.	X
BRSBit	BRS Bit (FD)	Triggers on the BRS bit of CAN FD frames.	X
CRCDbit	CRC Delimiter Bit (FD)	Triggers on the CRC delimiter bit in CAN FD frames.	X
EBActive	ESI Bit Active (FD)	Triggers on the ESI bit if set active.	X
EBPassive	ESI Bit Passive (FD)	Triggers on the ESI bit if set passive.	X
OVERload	Overload Frame	Triggers on an overload frame.	
MESSage	Message	Triggers on a symbolic message.	
MSIGnal	Message and Signal (non-FD)	Triggers on a symbolic message and a signal value.	
FDMSignal	Message and Signal (FD, first 8 bytes only)	Triggers on a symbolic message and a signal value, limited to the first 8 bytes of FD data.	
* Filtering by CAN IDs is available for these trigger conditions (see :SBUS<n>:CAN:TRIGger:IDFilter).			

CAN Id specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:ID and:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE commands.

CAN Data specification is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA command.

CAN Data Length Code is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger?
```

The :SBUS<n>:CAN:TRIGger? query returns the current CAN trigger on condition.

Return Format

```
<condition><NL>

<condition> ::= {SOF | EOF | IDD | DATA | FDD | IDR | IDE | ERR | ACK
    | FORM | STUF | CRC | SPEC | ALL | BRSB | CRCD | EBA | EBP | OVER
    | MESS | MSIG | FDMS}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721

- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 768
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 770
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 771
- ":SBUS<n>:CAN:TRIGger:IDFilter" on page 765
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 759
- ":SBUS<n>:CAN:SOURce" on page 761
- ":RECall:DBC[:STARt]" on page 684
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 772
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 773
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 774

:SBUS<n>:CAN:TRIGger:IDFilter

(see page 1354)

Command Syntax `:SBUS<n>:CAN:TRIGger:IDFilter {{0 | OFF} | {1 | ON}}`

The :SBUS<n>:CAN:TRIGger:IDFilter command specifies, in certain error and bit trigger modes, whether triggers are filtered by CAN IDs.

Query Syntax `:SBUS<n>:CAN:TRIGger:IDFilter?`

The :SBUS<n>:CAN:TRIGger:IDFilter? query returns the CAN trigger ID filter setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SBUS<n>:CAN:TRIGger" on page 762

:SBUS<n>:CAN:TRIGger:PATTern:DATA

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA command defines the CAN data pattern resource according to the string parameter. This pattern, along with the data length (set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command), control the data pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

If more bits are sent for <string> than specified by the :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command, the most significant bits will be truncated. If the data length is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA? query returns the current settings of the specified CAN data pattern resource in the binary string format.

Return Format

```
<string><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 768
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 770

:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC <dlc>

<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC command specifies the DLC value to be used in the CAN FD data trigger mode. A specific valid FD value can be specified, or -1 can be specified to indicate "don't care".

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:DLC? query returns the currently set DLC value.

Return Format

```
<dlc><NL>

<dlc> ::= integer between -1 (don't care) and 64, in NR1 format.
```

See Also

- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766

:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth

 (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:CAN:TRIGger:PATTern:DATA command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? query returns the current CAN data pattern length setting.

Return Format

```
<count><NL>

<count> ::= integer from 1 to 8 in NR1 format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766
- ":SBUS<n>:CAN:SOURce" on page 761

:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt <start>

<start> ::= integer between 0 and 63, in NR1 format.
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt command specifies the starting byte position for CAN FD data triggers.

CAN FD frames can have up to 64 bytes of data. You can trigger on up to 8 bytes of data. The starting byte position setting lets you trigger on data anywhere within the frame.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt?
```

The :SBUS<n>:CAN:TRIGger:PATTern:DATA:STARt? query returns the starting byte position setting.

Return Format

```
<start><NL>

<start> ::= integer between 0 and 63, in NR1 format.
```

See Also

- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 768

:SBUS<n>:CAN:TRIGger:PATTern:ID

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID command defines the CAN identifier pattern resource according to the string parameter. This pattern, along with the identifier mode (set by the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE

The ID pattern resource string is always 29 bits. Only 11 of these bits are used when the :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE is STANdard.

A string longer than 29 bits is truncated to 29 bits when setting the ID pattern resource.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID?
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID? query returns the current settings of the specified CAN identifier pattern resource in the 29-bit binary string format.

Return Format

```
<string><NL> in 29-bit binary string format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 771
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766

:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE command sets the CAN identifier mode. STANdard selects the standard 11-bit identifier. EXTended selects the extended 29-bit identifier. The CAN identifier is set by the :SBUS<n>:CAN:TRIGger:PATTern:ID command.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE?
```

The :SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 766
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 768
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 770

:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage

(see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage <name>

<name> ::= quoted ASCII string
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSage command specifies the message to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MESSage or MSIGnal.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:MESSage?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSage? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 773
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 774

:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name>

<name> ::= quoted ASCII string
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal command specifies the signal to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 772
- ":SBUS<n>:CAN:TRIGger:SYMBolic:VALue" on page 774

:SBUS<n>:CAN:TRIGger:SYMBolic:VALue

N (see page 1354)

Command Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data>

<data> ::= value in NR3 format
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue command specifies the signal value to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SBUS<n>:CAN:TRIGger:SYMBolic:VALue?
```

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

<data> ::= value in NR3 format
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SBUS<n>:CAN:TRIGger" on page 762
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 772
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 773

:SBUS<n>:CXPI Commands

NOTE These commands are valid when the CXPI (Clock Extension Peripheral Interface) serial decode and triggering option has been licensed.

Table 105 :SBUS<n>:CXPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:CXPI:BAUDrate <baudrate> (see page 777)	:SBUS<n>:CXPI:BAUDrate? (see page 777)	<baudrate> ::= integer from 9600 to 40000 in 100 b/s increments.
:SBUS<n>:CXPI:PARity {{0 | OFF} | {1 | ON}} (see page 778)	:SBUS<n>:CXPI:PARity? (see page 778)	{0 | 1}
:SBUS<n>:CXPI:SOURce <source> (see page 779)	:SBUS<n>:CXPI:SOURce? (see page 779)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:CXPI:TOLerance <percent> (see page 780)	:SBUS<n>:CXPI:TOLerance? (see page 780)	<percent> ::= from 1-30, in NR1 format.
:SBUS<n>:CXPI:TRIGger <mode> (see page 781)	:SBUS<n>:CXPI:TRIGger? (see page 782)	<mode> ::= {SOF | EOF | PTYPe | ID | DATA | LDATa | CRCerror | PARityerror | IBSerror | IFSerror | FRAMingerror | DLENgtherror | SAMPleerror | ALLerrors | SLEepframe | WAKeuppulse}
:SBUS<n>:CXPI:TRIGger:IDFilter {{0 | OFF} | {1 | ON}} (see page 783)	:SBUS<n>:CXPI:TRIGger:IDFilter? (see page 783)	{0 | 1}
:SBUS<n>:CXPI:TRIGger:PTYPe {{0 | OFF} | {1 | ON}} (see page 784)	:SBUS<n>:CXPI:TRIGger:PTYPe? (see page 784)	{0 | 1}
:SBUS<n>:CXPI:TRIGger:PATTern:DATA <string> (see page 785)	:SBUS<n>:CXPI:TRIGger:PATTern:DATA? (see page 785)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth <length> (see page 786)	:SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth? (see page 786)	<start> ::= integer between 0 and 12, in NR1 format.

Table 105 :SBUS<n>:CXPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt <start> (see page 787)	:SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt? (see page 787)	<start> ::= integer between 0 and 124, in NR1 format.
:SBUS<n>:CXPI:TRIGger:PATTern:ID <string> (see page 788)	:SBUS<n>:CXPI:TRIGger:PATTern:ID? (see page 788)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT <string> (see page 789)	:SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT? (see page 789)	<string> ::= "nn" where n ::= {0 | 1 | X}
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC <dlc> (see page 790)	:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC? (see page 790)	<dlc> ::= integer between -1 (don't care) and 15, in NR1 format, when trigger is in DATA mode. <dlc> ::= integer between -1 (don't care) and 255, in NR1 format, when trigger is in LDATa mode.
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM <string> (see page 791)	:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM? (see page 791)	<string> ::= "nn" where n ::= {0 | 1 | X}

:SBUS<n>:CXPI:BAUDrate

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:BAUDrate <baudrate>

<baudrate> ::= integer from 9600 to 40000 in 100 b/s increments.
```

The :SBUS<n>:CXPI:BAUDrate command specifies the baud rate of the CXPI signal from your device under test.

The CXPI baud rate can be set from 9600 b/s to 40000 b/s in 100 b/s increments.

You must set the baud rate to match your device under test.

The default baud rate is 20 kb/s.

Query Syntax

```
:SBUS<n>:CXPI:BAUDrate?
```

The :SBUS<n>:CXPI:BAUDrate? query returns the baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 9600 to 40000 in 100 b/s increments.
```

See Also

- ":SBUS<n>:CXPI:PARity" on page 778
- ":SBUS<n>:CXPI:SOURce" on page 779
- ":SBUS<n>:CXPI:TOLerance" on page 780
- ":SBUS<n>:CXPI:TRIGger" on page 781

:SBUS<n>:CXPI:PARity

(see page 1354)

Command Syntax `:SBUS<n>:CXPI:PARity {{0 | OFF} | {1 | ON}}`

The :SBUS<n>:CXPI:PARity command specifies whether the parity bit should be displayed in the identifier field.

When OFF, the upper bit is masked. The parity is still checked, but it is not displayed unless a parity error occurs.

Query Syntax `:SBUS<n>:CXPI:PARity?`

The :SBUS<n>:CXPI:PARity? query returns the parity display setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SBUS<n>:CXPI:BAUDrate" on page 777
- ":SBUS<n>:CXPI:SOURce" on page 779
- ":SBUS<n>:CXPI:TOLerance" on page 780
- ":SBUS<n>:CXPI:TRIGger" on page 781

:SBUS<n>:CXPI:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:CXPI:SOURce command selects the oscilloscope channel connected to the CXPI signal line.

Query Syntax

```
:SBUS<n>:CXPI:SOURce?
```

The :SBUS<n>:CXPI:SOURce? query returns the selected oscilloscope channel source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":SBUS<n>:CXPI:BAUDrate" on page 777
- ":SBUS<n>:CXPI:PARity" on page 778
- ":SBUS<n>:CXPI:TOLerance" on page 780
- ":SBUS<n>:CXPI:TRIGger" on page 781

:SBUS<n>:CXPI:TOLerance

(see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TOLerance <percent>

<percent> ::= from 1-30, in NR1 format.
```

The :SBUS<n>:CXPI:TOLerance command specifies the tolerance as a percentage of the Tbit width.

Query Syntax

```
:SBUS<n>:CXPI:TOLerance?
```

The :SBUS<n>:CXPI:TOLerance? query returns the tolerance setting.

Return Format

```
<percent><NL>

<percent> ::= from 1-30, in NR1 format.
```

See Also

- ":SBUS<n>:CXPI:BAUDrate" on page 777
- ":SBUS<n>:CXPI:PARity" on page 778
- ":SBUS<n>:CXPI:SOURce" on page 779
- ":SBUS<n>:CXPI:TRIGger" on page 781

:SBUS<n>:CXPI:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger <mode>

<mode> ::= {SOF | EOF | PTYPe | ID | DATA | LDATa | CRCerror
   | PARityerror | IBSerror | IFSerror | FRAMingerror | DLENgtherror
   | SAMPleerror | ALLerrors | SLEepframe | WAKeuppulse}
```

The :SBUS<n>:CXPI:TRIGger command selects the CXPI trigger type:

- SOF – (Start of Frame) triggers at the start bit of any frame.
- EOF – (End of Frame) triggers at the end of any frame.
- PTYPe – triggers on any frame that starts with the special PTYPE byte.

  PTYPE frames begins with an extra PID byte with a Frame ID of 0000000b (reserved for only PTYPE frames). The PTYPE PID byte is then followed by a regular PID byte and the rest of the normal frame. The extra PTYPE byte is never included in the CRC calculation.
- ID – (Frame ID) triggers on a user-defined Frame ID at the end of the PID byte. The Frame ID value is user-defined, 7 bits, and has bitwise don't-cares. You can specify whether to trigger on PTYPE present or no PTYPE present.
- DATA – (Frame ID, Info and Data) triggers on CXPI frames at the end of the last data byte defined in the trigger. In addition to the PID value, you can specify the contents of the Frame Info byte with bitwise don't-cares. You can specify up to 12 data bytes on which to trigger with bitwise don't-cares.
- LDATa – (Frame ID, Info and Data (Long Frame)) triggers on CXPI frames at the end of the last data byte defined in the trigger. The standard DLC field will be locked to 1111b. You can specify up to 12 bytes of data on which to trigger and specify the start byte number as an offset. The offset can be up to 255.
- CRCerror – (CRC Field Error) triggers when the calculated CRC does not match the transmitted CRC. You can optionally filter by Frame ID and PTYPE as in the Frame ID trigger.
- PARityerror – triggers when the parity bit in the PID or PTYPE field is not correct.
- IBSerror – (Inter-Byte Space Error) triggers when there are more than 9 bits between consecutive bytes in a frame. You can optionally filter by Frame ID and PTYPE as in the Frame ID trigger.
- IFSerror – (Inter-Frame Space Error) triggers when there are fewer than 10 idle bits before a new frame begins.
- FRAMingerror – triggers when the stop bit of a byte is not logical 1. You can optionally filter by Frame ID and PTYPE as in the Frame ID trigger.

- DLENgtherror – (Data Length Error) triggers when there are more data bytes in a frame than is indicated by the DLC or Extended DLC field. You can optionally filter by Frame ID and PTYPE as in the Frame ID trigger.
- SAMPleerror – triggers when 10 consecutive logical 0s are detected.
- ALLerrors – triggers on all CRC, Parity, IBS, Stop Bit, Data Length, and Sample errors.
- SLEepframe – triggers when a normal frame is transmitted matching the definition of a sleep frame in the CXPI specification.
- WAKeuppulse – triggers when a wakeup pulse is detected.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger?
```

The :SBUS<n>:CXPI:TRIGger? query returns the CXPI trigger type setting.

Return Format

```
<mode><NL>

<mode> ::= {SOF | EOF | PTYP | ID | DATA | LDAT | CRC | PAR | IBS
    | IFS | FRAM | DLEN | SAMP | ALL | SLE | WAK}
```

See Also

- ":SBUS<n>:CXPI:BAUDrate" on page 777
- ":SBUS<n>:CXPI:PARity" on page 778
- ":SBUS<n>:CXPI:SOURce" on page 779
- ":SBUS<n>:CXPI:TOLerance" on page 780
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:IDFilter

N (see page 1354)

Command Syntax `:SBUS<n>:CXPI:TRIGger:IDFilter {{0 | OFF} | {1 | ON}}`

When triggering on CRC Field Errors, Inter-Byte Space Errors, Framing Errors, or Data Length Errors, the :SBUS<n>:CXPI:TRIGger:IDFilter command lets you enable/disable modification of the trigger so that it occurs only for a specified ID.

Query Syntax `:SBUS<n>:CXPI:TRIGger:IDFilter?`

The :SBUS<n>:CXPI:TRIGger:IDFilter? query returns the ID filter setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PTYPe

(see page 1354)

Command Syntax `:SBUS<n>:CXPI:TRIGger:PTYPe {{0 | OFF} | {1 | ON}}`

For the trigger types that let you trigger on data, the :SBUS<n>:CXPI:TRIGger:PTYPe command specifies whether you want to trigger when the special PTYPE byte is present (ON) or not present (OFF).

Query Syntax `:SBUS<n>:CXPI:TRIGger:PTYPe?`

The :SBUS<n>:CXPI:TRIGger:PTYPe? query returns the PTYPE trigger setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
```

For the trigger types that let you trigger on data, the :SBUS<n>:CXPI:TRIGger:PATTern:DATA command lets you specify the data value.

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth command specifies the length of the data to trigger on, from 0 to 12 bytes, limited by the data length code (DLC) setting of the :SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC command.

When triggering on long frames (with the LDATa trigger type) that can have up to 255 data bytes, the maximum number of data bytes you can include in the trigger specification is still only 12 bytes. In this case, you can use the :SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt command to specify the starting byte location where the data value should be found.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA? query returns the specified data value.

Returned data values are always quoted binary format strings.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X}
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer between 0 and 12, in NR1 format.
```

For the trigger types that let you trigger on data, the :SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth command specifies the length of the data to trigger on, from 0 to 12 bytes, limited by the data length code (DLC) setting of the :SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC command.

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA command lets you specify the data value to trigger on.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth? query returns the data length setting.

Return Format

```
<length><NL>

<length> ::= integer between 0 and 12, in NR1 format.
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt <start>

<start> ::= integer between 0 and 124, in NR1 format.
```

When triggering on long frames (with the LDATa trigger type) that can have up to 255 data bytes, the maximum number of data bytes you can include in the trigger specification is still only 12 bytes. In this case, you can use the :SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt command to specify the starting byte location where the data value should be found.

The starting byte location must be within the first 123 bytes when PTYPE is present or 124 bytes when PTYPE is not present.

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth command lets you specify the length of the data value to trigger on.

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA command lets you specify the data value to trigger on.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt? query returns the start byte setting.

Return Format

```
<start><NL>

<start> ::= integer between 0 and 124, in NR1 format.
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:ID

(see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:ID <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
```

For the trigger types that let you specify frame ID values in the trigger or allow filtering by the frame ID, the :SBUS<n>:CXPI:TRIGger:PATTern:ID command lets you specify the frame ID value.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:ID?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:ID? query returns the specified frame ID value.

Returned frame ID values are always quoted binary format strings.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X}
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT <string>

<string> ::= "nn" where n ::= {0 | 1 | X}
```

The command ...

For the trigger types that let you trigger on data, as well as frame ID and frame information bits, the :SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT command lets you specify the Count (CT) value of the CXPI frame you wish to trigger on. This is a two-bit binary value.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT? query returns the specified CT bits included in the trigger.

Return Format

```
<string><NL>

<string> ::= "nn" where n ::= {0 | 1 | X}
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC <dlc>

<dlc> ::= integer between -1 (don't care) and 15, in NR1 format,
   when trigger is in DATA mode.

<dlc> ::= integer between -1 (don't care) and 255, in NR1 format,
   when trigger is in LDATa mode.
```

For the trigger types that let you trigger on data, the :SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC command specifies the data length code of the CXPI frame you wish to trigger on.

This will also affect the number of data bytes you can specify in the trigger.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC? query returns the DLC trigger value setting.

Return Format

```
<dlc><NL>

<dlc> ::= integer between -1 (don't care) and 15, in NR1 format,
   when trigger is in DATA mode.

<dlc> ::= integer between -1 (don't care) and 255, in NR1 format,
   when trigger is in LDATa mode.
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM" on page 791

:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM

N (see page 1354)

Command Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM <string>

<string> ::= "nn" where n ::= {0 | 1 | X}
```

For the trigger types that let you trigger on data, as well as frame ID and frame information bits, the :SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM command lets you specify the Network Management (NM) value of the CXPI frame you wish to trigger on. This is a two-bit binary value.

Query Syntax

```
:SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM?
```

The :SBUS<n>:CXPI:TRIGger:PATTern:INFO:NM? query returns the specified NM bits included in the trigger.

Return Format

```
<string><NL>

<string> ::= "nn" where n ::= {0 | 1 | X}
```

See Also

- ":SBUS<n>:CXPI:TRIGger" on page 781
- ":SBUS<n>:CXPI:TRIGger:IDFilter" on page 783
- ":SBUS<n>:CXPI:TRIGger:PTYPe" on page 784
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA" on page 785
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:LENGth" on page 786
- ":SBUS<n>:CXPI:TRIGger:PATTern:DATA:STARt" on page 787
- ":SBUS<n>:CXPI:TRIGger:PATTern:ID" on page 788
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:CT" on page 789
- ":SBUS<n>:CXPI:TRIGger:PATTern:INFO:DLC" on page 790

:SBUS<n>:IIC Commands

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Table 106 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 793)	:SBUS<n>:IIC:ASIZe? (see page 793)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce]:CLOCk <source> (see page 794)	:SBUS<n>:IIC[:SOURce]:CLOCk? (see page 794)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC[:SOURce]:DATA <source> (see page 795)	:SBUS<n>:IIC[:SOURce]:DATA? (see page 795)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value> (see page 796)	:SBUS<n>:IIC:TRIGger:PATTern:ADDRess? (see page 796)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value> (see page 797)	:SBUS<n>:IIC:TRIGger:PATTern:DATA? (see page 797)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value> (see page 798)	:SBUS<n>:IIC:TRIGger:PATTern:DATa2? (see page 798)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:QUALifier <value> (see page 799)	:SBUS<n>:IIC:TRIGger:QUALifier? (see page 799)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 800)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 800)	<type> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk | NACKnowledge | READEprom | READ7 | WRITe7 | R7Data2 | W7Data2 | WRITe10}

:SBUS<n>:IIC:ASIZe

(see page 1354)

Command Syntax

```
:SBUS<n>:IIC:ASIZe <size>

<size> ::= {BIT7 | BIT8}
```

The :SBUS<n>:IIC:ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

Query Syntax

```
:SBUS<n>:IIC:ASIZe?
```

The :SBUS<n>:IIC:ASIZe? query returns the current IIC address width setting.

Return Format

```
<mode><NL>

<mode> ::= {BIT7 | BIT8}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:IIC Commands" on page 792

:SBUS<n>:IIC[:SOURce]:CLOCk

N (see page 1354)

Command Syntax

```
:SBUS<n>:IIC[:SOURce]:CLOCk <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:IIC[:SOURce]:CLOCk command sets the source for the IIC serial clock (SCL).

Query Syntax

```
:SBUS<n>:IIC[:SOURce]:CLOCk?
```

The :SBUS<n>:IIC[:SOURce]:CLOCk? query returns the current source for the IIC serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:IIC[:SOURce]:DATA" on page 795

:SBUS<n>:IIC[:SOURce]:DATA

(see page 1354)

Command Syntax

```
:SBUS<n>:IIC[:SOURce]:DATA <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:IIC[:SOURce]:DATA command sets the source for IIC serial data (SDA).

Query Syntax

```
:SBUS<n>:IIC[:SOURce]:DATA?
```

The :SBUS<n>:IIC[:SOURce]:DATA? query returns the current source for IIC serial data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:IIC[:SOURce]:CLOCk" on page 794

:SBUS<n>:IIC:TRIGger:PATTern:ADDRess

N (see page 1354)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :SBUS<n>:IIC:TRIGger:PATTern:ADDRess command sets the address for IIC data.The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess?
```

The :SBUS<n>:IIC:TRIGger:PATTern:ADDRess? query returns the current address for IIC data.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 797
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 798
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 800

:SBUS<n>:IIC:TRIGger:PATTern:DATA

(see page 1354)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATA command sets IIC data. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATA?
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATA? query returns the current pattern for IIC data.

Return Format

```
<value><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 796
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 798
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 800

:SBUS<n>:IIC:TRIGger:PATTern:DATa2

 (see page 1354)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATa2 command sets IIC data 2. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:PATTern:DATa2?
```

The :SBUS<n>:IIC:TRIGger:PATTern:DATa2? query returns the current pattern for IIC data 2.

Return Format

```
<value><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 796
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 797
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 800

:SBUS<n>:IIC:TRIGger:QUALifier

N (see page 1354)

Command Syntax

```
:SBUS<n>:IIC:TRIGger:QUALifier <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :SBUS<n>:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEprom.

Query Syntax

```
:SBUS<n>:IIC:TRIGger:QUALifier?
```

The :SBUS<n>:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

Return Format

```
<value><NL>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 800

:SBUS<n>:IIC:TRIGger[:TYPE]

N (see page 1354)

Command Syntax

```
:SBUS<n>:IIC:TRIGger[:TYPE] <value>

<value> ::= {STARt | STOP | RESTart | ADDRess | ANACk | DNACk
    | NACKnowledge | READEprom | READ7 | WRITe7 | R7Data2 | W7Data2
    | WRITe10}
```

The :SBUS<n>:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- STARt – Start condition.
- STOP – Stop condition.
- RESTart – Another start condition occurs before a stop condition.
- ADDRess – Triggers on the selected address. The R/W bit is ignored.
- ANACk – Address with no acknowledge.
- DNACk – Write data with no acknowledge.
- NACKnowledge – Missing acknowledge.
- READEprom – EEPROM data read.
- READ7 – 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- WRITe7 – 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- R7Data2 – 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- W7Data2 – 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- WRITe10 – 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).

NOTE **The short form of READ7 (READ7), READEprom (READE), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules (see page 1356).**

Query Syntax

```
:SBUS<n>:IIC:TRIGger[:TYPE]?
```

The :SBUS<n>:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

Return Format

```
<value><NL>

<value> ::= {STAR | STOP | REST | ADDR | ANAC | DNAC | NACK | READE
    | READ7 | WRIT7 | R7D2 | W7D2 | WRIT10}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 796
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 797
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 798
- ":SBUS<n>:IIC:TRIGger:QUALifier" on page 799
- "Long Form to Short Form Truncation Rules" on page 1356

:SBUS<n>:LIN Commands

NOTE These commands are valid when the automotive CAN and LIN serial decode option (AUTO license) has been installed.

Table 107 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:DISPlay <type> (see page 804)	:SBUS<n>:LIN:DISPlay? (see page 804)	<type> ::= {HEXadecimal | SYMBolic}
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 805)	:SBUS<n>:LIN:PARity? (see page 805)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 806)	:SBUS<n>:LIN:SAMPlepoint? (see page 806)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 807)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 807)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 808)	:SBUS<n>:LIN:SOURce? (see page 808)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 809)	:SBUS<n>:LIN:STANdard? (see page 809)	<std> ::= {LIN13 | LIN13NLC | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 810)	:SBUS<n>:LIN:SYNCbreak? (see page 810)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 811)	:SBUS<n>:LIN:TRIGger? (see page 811)	<condition> ::= {SYNCbreak | ID | DATA}
:SBUS<n>:LIN:TRIGger:ID <value> (see page 813)	:SBUS<n>:LIN:TRIGger:ID? (see page 813)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal

Table 107 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 814)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 814)	<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when <base> = DECimal <string> ::= "nn...n" where n ::= {0 | 1 | X | $} when <base> = BINary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length> (see page 816)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 816)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 817)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 817)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name> (see page 818)	:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? (see page 818)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name> (see page 819)	:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? (see page 819)	<name> ::= quoted ASCII string
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data> (see page 820)	:SBUS<n>:LIN:TRIGger:SYMBolic:VALue? (see page 820)	<data> ::= value in NR3 format

:SBUS<n>:LIN:DISPlay

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:DISPlay <type>

<type> ::= {HEXadecimal | SYMBolic}
```

The :SBUS<n>:LIN:DISPlay command specifies, when LIN symbolic data is loaded into the oscilloscope, whether symbolic values (from the LDF file) or hexadecimal values are displayed in the decode waveform and the Lister window.

Query Syntax

```
:SBUS<n>:LIN:DISPlay?
```

The :SBUS<n>:LIN:DISPlay? query returns the LIN decode display type.

Return Format

```
<type><NL>

<type> ::= {HEX | SYMB}
```

See Also

- ":RECall:LDF[:STARt]" on page 686

:SBUS<n>:LIN:PARity

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:PARity <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS<n>:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

Query Syntax

```
:SBUS<n>:LIN:PARity?
```

The :SBUS<n>:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:LIN Commands" on page 802

:SBUS<n>:LIN:SAMPlepoint

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:SAMPlepoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :SBUS<n>:LIN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

NOTE The sample point values are not limited by the baud rate.

Query Syntax

```
:SBUS<n>:LIN:SAMPlepoint?
```

The :SBUS<n>:LIN:SAMPlepoint? query returns the current LIN sample point setting.

Return Format

```
<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:TRIGger" on page 811

:SBUS<n>:LIN:SIGNal:BAUDrate

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
```

The :SBUS<n>:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal from 2400 b/s to 625 kb/s in 100 b/s increments. If you enter a baud rate that is not divisible by 100 b/s, the baud rate is set to the nearest baud rate divisible by 100 b/s.

Query Syntax

```
:SBUS<n>:LIN:SIGNal:BAUDrate?
```

The :SBUS<n>:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289
- ":SBUS<n>:LIN:SOURce" on page 808

:SBUS<n>:LIN:SOURce

 (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:LIN:SOURce command sets the source for the LIN signal.

Query Syntax

```
:SBUS<n>:LIN:SOURce?
```

The :SBUS<n>:LIN:SOURce? query returns the current source for the LIN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289

:SBUS<n>:LIN:STANdard

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:STANdard <std>

<std> ::= {LIN13 | LIN13NLC | LIN20}
```

The :SBUS<n>:LIN:STANdard command sets the LIN standard in effect for triggering and decoding:

- LIN13 – LIN 1.3.
- LIN13NLC – LIN 1.3 (no length control). Select this for systems where length control is not used and all nodes have knowledge of the data packet size. In LIN 1.3, the ID may or may not be used to indicate the number of bytes. (In LIN 2.X, there is no length control.)
- LIN20 – LIN 2.X.

For LIN 1.2 signals, use the LIN 1.3 setting. The LIN 1.3 setting assumes the signal follows the "Table of Valid ID Values" as shown in section A.2 of the LIN Specification dated December 12, 2002. If your signal does not comply with the table, use the LIN 2.X setting.

Query Syntax

```
:SBUS<n>:LIN:STANdard?
```

The :SBUS<n>:LIN:STANdard? query returns the current LIN standard setting.

Return Format

```
<std><NL>

<std> ::= {LIN13 | LIN13NLC | LIN20}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289
- ":SBUS<n>:LIN:SOURce" on page 808

:SBUS<n>:LIN:SYNCbreak

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:SYNCbreak <value>

<value> ::= integer = {11 | 12 | 13}
```

The :SBUS<n>:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11, 12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

Query Syntax

```
:SBUS<n>:LIN:SYNCbreak?
```

The :SBUS<n>:LIN:SYNCbreak? query returns the current LIN sync break setting.

Return Format

```
<value><NL>

<value> ::= {11 | 12 | 13}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289
- ":SBUS<n>:LIN:SOURce" on page 808

:SBUS<n>:LIN:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger <condition>

<condition> ::= {SYNCbreak | ID | DATA | PARityerror | CSUMerror
                 | FRAMe | FSIGnal}
```

The :SBUS<n>:LIN:TRIGger command sets the LIN trigger condition to be:

- SYNCbreak — Sync Break.
- ID — Frame ID.

  Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.
- DATA — Frame ID and Data.

  Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.

  Use the :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth and :SBUS<n>:LIN:TRIGger:PATTern:DATA commands to specify the data string length and value.
- PARityerror — parity errors.
- CSUMerror — checksum errors.
- FRAMe — Triggers on a symbolic frame.
- FSIGnal — Triggers on a symbolic frame and a signal value.

Query Syntax

```
:SBUS<n>:LIN:TRIGger?
```

The :SBUS<n>:LIN:TRIGger? query returns the current LIN trigger value.

Return Format

```
<condition><NL>

<condition> ::= {SYNC | ID | DATA | PAR | CSUM | FRAM | FSIG}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:TRIGger:ID" on page 813
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 816
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 814
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289
- ":SBUS<n>:LIN:SOURce" on page 808
- ":RECall:LDF[:STARt]" on page 686
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 818
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 819

- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 820

:SBUS<n>:LIN:TRIGger:ID

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
            from 0-63 or 0x00-0x3f

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :SBUS<n>:LIN:TRIGger:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:ID?
```

The :SBUS<n>:LIN:TRIGger:ID? query returns the current LIN identifier setting.

Return Format

```
<value><NL>

<value> ::= integer in decimal
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1289
- ":SBUS<n>:LIN:SOURce" on page 808

:SBUS<n>:LIN:TRIGger:PATTern:DATA

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string>

<string> ::= "n" where n ::= 32-bit integer in unsigned decimal when
             <base> = DECimal

<string> ::= "nn...n" where n ::= {0 | 1 | X | $} when
             <base> = BINary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when
             <base> = HEX
```

NOTE <base> is specified with the :SBUS<n>:LIN:TRIGger:PATTern:FORMat command. The default <base> is BINary.

The :SBUS<n>:LIN:TRIGger:PATTern:DATA command specifies the LIN trigger data pattern searched for in each LIN data field.

Set a <string> bit to "0" or "1" to set the corresponding bit in the data pattern to low or high, respectively.

Set a <string> bit to "X" to ignore (mask off) that bit in the data pattern.

Use the "$" character to indicate that the value of the corresponding bit will not be changed (the existing bit value is used).

When <base> = DECimal, the "X" and "$" characters cannot be entered. When queried, the "$" character is returned when any bits in the pattern have the value of "X" and <base> = DECimal. When any bits in a given nibble have the value of "X" and <base> = HEX, the "$" character is returned for the corresponding nibble.

NOTE The length of the trigger data value is determined by the :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth command.

NOTE If more bits are sent for <string> than the specified trigger pattern data length, the most significant bits will be truncated. If the data length size is changed after the <string> is programmed, the added or deleted bits will be added to or deleted from the least significant bits.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA?
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA? query returns the currently specified LIN trigger data pattern.

Return Format

```
<string><NL>
```

See Also
- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:LIN:TRIGger:PATTern:FORMat" on page 817
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 816

:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth

 (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the LIN data string. The number of bytes in the string can be anywhere from 1 bytes to 8 bytes (64 bits). The value for these bytes is set by the :SBUS<n>:LIN:TRIGger:PATTern:DATA command.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth?
```

The :SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? query returns the current LIN data pattern length setting.

Return Format

```
<count><NL>

<count> ::= integer from 1 to 8 in NR1 format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 814
- ":SBUS<n>:LIN:SOURce" on page 808

:SBUS<n>:LIN:TRIGger:PATTern:FORMat

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}
```

The :SBUS<n>:LIN:TRIGger:PATTern:FORMat command sets the entry (and query) number base used by the :SBUS<n>:LIN:TRIGger:PATTern:DATA command. The default <base> is BINary.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:PATTern:FORMat?
```

The :SBUS<n>:LIN:TRIGger:PATTern:FORMat? query returns the currently set number base for LIN pattern data.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX | DEC}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 814
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 816

:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe <name>

<name> ::= quoted ASCII string
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe command specifies the message to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FRAMe or FSIGnal.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 819
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 820

:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal

N (see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal <name>

<name> ::= quoted ASCII string
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal command specifies the signal to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FSIGnal.

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 818
- ":SBUS<n>:LIN:TRIGger:SYMBolic:VALue" on page 820

:SBUS<n>:LIN:TRIGger:SYMBolic:VALue

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue <data>

<data> ::= value in NR3 format
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:VALue command specifies the signal value to trigger on when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN trigger mode is set to FSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SBUS<n>:LIN:TRIGger:SYMBolic:VALue?
```

The :SBUS<n>:LIN:TRIGger:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

<data> ::= value in NR3 format
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SBUS<n>:LIN:TRIGger" on page 811
- ":SBUS<n>:LIN:TRIGger:SYMBolic:FRAMe" on page 818
- ":SBUS<n>:LIN:TRIGger:SYMBolic:SIGNal" on page 819

:SBUS<n>:M1553 Commands

NOTE These commands are valid when the MIL-STD-1553 and ARINC 429 triggering and serial decode option (AERO license) has been installed.

Table 108 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTosetup (see page 822)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 823)	:SBUS<n>:M1553:BASE? (see page 823)	<base> ::= {BINary | HEX}
:SBUS<n>:M1553:SOURce <source> (see page 824)	:SBUS<n>:M1553:SOURce? (see page 824)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGger:PATTern:DATA <string> (see page 825)	:SBUS<n>:M1553:TRIGger:PATTern:DATA? (see page 825)	<string> ::= "nn...n" where n ::= {0 | 1 | X}
:SBUS<n>:M1553:TRIGger:RTA <value> (see page 826)	:SBUS<n>:M1553:TRIGger:RTA? (see page 826)	<value> ::= 5-bit integer in decimal, <nondecimal>, or <string> from 0-31 <nondecimal> ::= #Hnn where n ::= {0,..,9|A,..,F} <string> ::= "0xnn" where n::= {0,..,9|A,..,F}
:SBUS<n>:M1553:TRIGger:TYPE <type> (see page 827)	:SBUS<n>:M1553:TRIGger:TYPE? (see page 827)	<type> ::= {DSTArt | DSTOp | CSTArt | CSTOp | RTA | PERRor | SERRor | MERRor | RTA11}

:SBUS<n>:M1553:AUTosetup

 (see page 1354)

Command Syntax `:SBUS<n>:M1553:TRIGger:AUTosetup`

The :SBUS<n>:M1553:AUTosetup command automatically sets these options for decoding and triggering on MIL-STD-1553 signals:

- High/Low Trigger Thresholds: to a voltage value equal to ±1/3 division based on the source channel's current V/div setting.
- Noise Reject: Off.
- Probe Attenuation: 10.0.
- Serial Decode: On.
- Trigger: the specified serial bus (n of SBUS<n>).

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:M1553:SOURce" on page 824

:SBUS<n>:M1553:BASE

(see page 1354)

Command Syntax

```
:SBUS<n>:M1553:BASE <base>

<base> ::= {BINary | HEX}
```

The :SBUS<n>:M1553:BASE command determines the base to use for the MIL-STD-1553 decode display.

Query Syntax

```
:SBUS<n>:M1553:BASE?
```

The :SBUS<n>:M1553:BASE? query returns the current MIL-STD-1553 display decode base.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:M1553 Commands" on page 821

:SBUS<n>:M1553:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:M1553:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:M1553:SOURce command sets the source of the MIL-STD 1553 signal.

Use the :TRIGger:LEVel:HIGH and :TRIGger:LEVel:LOW commands to set the threshold levels for the selected source.

Query Syntax

```
:SBUS<n>:M1553:TRIGger:SOURce?
```

The :SBUS<n>:M1553:SOURce? query returns the currently set source of the MIL-STD 1553 signal.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- ":TRIGger:LEVel:HIGH" on page 1064
- ":TRIGger:LEVel:LOW" on page 1065
- ":TRIGger:MODE" on page 1066
- "Introduction to :TRIGger Commands" on page 1053

:SBUS<n>:M1553:TRIGger:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:M1553:TRIGger:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}
```

The :SBUS<n>:M1553:TRIGger:PATTern:DATA command sets the 11 bits to trigger on if the trigger type has been set to RTA11 (RTA + 11 Bits) using the :SBUS<n>:M1553:TRIGger:TYPE command.

Query Syntax

```
:SBUS<n>:M1553:TRIGger:PATTern:DATA?
```

The :SBUS<n>:M1553:TRIGger:PATTern:DATA? query returns the current 11-bit setting.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:M1553:TRIGger:TYPE" on page 827
- ":SBUS<n>:M1553:TRIGger:RTA" on page 826

:SBUS<n>:M1553:TRIGger:RTA

 (see page 1354)

Command Syntax

```
:SBUS<n>:M1553:TRIGger:RTA <value>

<value> ::= 5-bit integer in decimal, <nondecimal>, or
            <string> from 0-31

<nondecimal> ::= #Hnn where n ::= {0,..,9|A,..,F}

<string> ::= "0xnn" where n::= {0,..,9|A,..,F}
```

The :SBUS<n>:M1553:TRIGger:RTA command sets the Remote Terminal Address (RTA) to trigger on when the trigger type has been set to RTA or RTA11 (using the :SBUS<n>:M1553:TRIGger:TYPE command).

To set the RTA value to don't cares (0xXX), set the value to -1.

Query Syntax

```
:SBUS<n>:M1553:TRIGger:RTA?
```

The :SBUS<n>:M1553:TRIGger:RTA? query returns the RTA value.

Return Format

```
<value><NL> in decimal format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:M1553:TRIGger:TYPE" on page 827

:SBUS<n>:M1553:TRIGger:TYPE

N (see page 1354)

Command Syntax

```
:SBUS<n>:M1553:TRIGger:TYPE <type>

<type> ::= {DSTArt | DSTOp | CSTArt | CSTOp | RTA | PERRor | SERRor
           | MERRor | RTA11}
```

The :SBUS<n>:M1553:TRIGger:TYPE command specifies the type of MIL-STD-1553 trigger to be used:

- DSTArt – (Data Word Start) triggers on the start of a Data word (at the end of a valid Data Sync pulse).
- DSTOp – (Data Word Stop) triggers on the end of a Data word.
- CSTArt – (Command/Status Word Start) triggers on the start of Command/Status word (at the end of a valid C/S Sync pulse).
- CSTOp – (Command/Status Word Stop) triggers on the end of a Command/Status word.
- RTA – (Remote Terminal Address) triggers if the RTA of the Command/Status word matches the specified value. The value is specified in hex.
- RTA11 – (RTA + 11 Bits) triggers if the RTA and the remaining 11 bits match the specified criteria. The RTA can be specified as a hex value, and the remaining 11 bits can be specified as a 1, 0, or X (don't care).
- PERRor – (Parity Error) triggers if the (odd) parity bit is incorrect for the data in the word.
- MERRor – (Manchester Error) triggers if a Manchester encoding error is detected.
- SERRor – (Sync Error) triggers if an invalid Sync pulse is found.

Query Syntax

```
:SBUS<n>:M1553:TRIGger:TYPE?
```

The :SBUS<n>:M1553:TRIGger:TYPE? query returns the currently set MIL-STD-1553 trigger type.

Return Format

```
<type><NL>

<type> ::= {DSTA | DSTO | CSTA | CSTO | RTA | PERR | SERR
           | MERR | RTA11}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:M1553:TRIGger:RTA" on page 826
- ":SBUS<n>:M1553:TRIGger:PATTern:DATA" on page 825
- ":TRIGger:MODE" on page 1066

:SBUS<n>:MANChester Commands

NOTE These commands are valid when the automotive MANChester serial decode and triggering option has been licensed.

Table 109 :SBUS<n>:MANChester Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:MANChester:BASE <base> (see page 830)	:SBUS<n>:MANChester:BASE? (see page 830)	<base> ::= {HEX | DECimal | ASCii}
:SBUS<n>:MANChester:BAUDrate <baudrate> (see page 831)	:SBUS<n>:MANChester:BAUDrate? (see page 831)	<baudrate> ::= integer from 500 to 5000000 in 100 b/s increments
:SBUS<n>:MANChester:BITorder <bitorder> (see page 832)	:SBUS<n>:MANChester:BITorder? (see page 832)	<bitorder> ::= {MSBFirst | LSBFirst}
:SBUS<n>:MANChester:DISPlay <format> (see page 833)	:SBUS<n>:MANChester:DISPlay? (see page 833)	<format> ::= {BIT | WORD}
:SBUS<n>:MANChester:DSIZe {AUTO | <#words>} (see page 834)	:SBUS<n>:MANChester:DSIZe? (see page 834)	<#words> ::= from 1-255, in NR1 format
:SBUS<n>:MANChester:HSIZe <#bits> (see page 835)	:SBUS<n>:MANChester:HSIZe? (see page 835)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:MANChester:IDLE:BITS <#bits> (see page 836)	:SBUS<n>:MANChester:IDLE:BITS? (see page 836)	<#bits> ::= minimum idle time in terms of bit width, from 1.50 to 32.0, in NR3 format.
:SBUS<n>:MANChester:LOGic <logic> (see page 837)	:SBUS<n>:MANChester:LOGic? (see page 837)	<logic> ::= {FALLing | RISing}
:SBUS<n>:MANChester:SOURce <source> (see page 838)	:SBUS<n>:MANChester:SOURce? (see page 838)	<source> ::= {CHANnel<n>}
:SBUS<n>:MANChester:SSIZe <#bits> (see page 839)	:SBUS<n>:MANChester:SSIZe? (see page 839)	<#bits> ::= from 0-255, in NR1 format
:SBUS<n>:MANChester:STARt <edge#> (see page 840)	:SBUS<n>:MANChester:STARt? (see page 840)	<edge#> ::= from 1-256, in NR1 format

Table 109 :SBUS<n>:MANChester Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:MANChester:TOLerance <percent> (see page 841)	:SBUS<n>:MANChester:TOLerance? (see page 841)	<percent> ::= from 1-30, in NR1 format
:SBUS<n>:MANChester:TRIGger <mode> (see page 842)	:SBUS<n>:MANChester:TRIGger? (see page 842)	<mode> ::= {SOF | VALue | MERRor}
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA <string> (see page 843)	:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA? (see page 843)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh <width> (see page 844)	:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh? (see page 844)	<width> ::= integer from 4 to 128 in NR1 format
:SBUS<n>:MANChester:TSIZe <#bits> (see page 845)	:SBUS<n>:MANChester:TSIZe? (see page 845)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:MANChester:WSIZe <#bits> (see page 846)	:SBUS<n>:MANChester:WSIZe? (see page 846)	<#bits> ::= from 2-32, in NR1 format

:SBUS<n>:MANChester:BASE

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:BASE <base>

<base> ::= {HEX | DECimal | ASCii | BINary}
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:BASE command specifies the base for the Manchester bus decode and Lister display.

- HEX – hexadecimal
- DECimal – unsigned decimal
- ASCii

When the display format is BIT, the only legal decode base value is BINary.

Query Syntax

```
:SBUS<n>:MANChester:BASE?
```

The :SBUS<n>:MANChester:BASE? query returns the decode number base setting.

Return Format

```
<base><NL>

<base> ::= {HEX | DEC | ASC | BIN}
```

See Also

- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:IDLE:BITS" on page 836
- ":SBUS<n>:MANChester:LOGic" on page 837
- ":SBUS<n>:MANChester:STARt" on page 840

:SBUS<n>:MANChester:BAUDrate

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:BAUDrate <baudrate>

<baudrate> ::= integer from 500 to 5000000 in 100 b/s increments
```

The :SBUS<n>:MANChester:BAUDrate command specifies the baud rate of the Manchester signal.

Query Syntax

```
:SBUS<n>:MANChester:BAUDrate?
```

The :SBUS<n>:MANChester:BAUDrate? query returns the specified baud rate.

Return Format

```
<baudrate><NL>
```

See Also

- ":SBUS<n>:MANChester:SOURce" on page 838
- ":SBUS<n>:MANChester:TOLerance" on page 841

:SBUS<n>:MANChester:BITorder

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:BITorder <bitorder>

<bitorder> ::= {MSBFirst | LSBFirst}
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:BITorder command specifies the order of transmission on the Manchester bus:

- MSBFirst – specifies the most significant bit is transmitted first.
- LSBFirst – specifies the least significant bit is transmitted first.

Query Syntax

```
:SBUS<n>:MANChester:BITorder?
```

The :SBUS<n>:MANChester:BITorder? query returns the bit order setting.

Return Format

```
<bitorder><NL>

<bitorder> ::= {MSBF | LSBF}
```

See Also

- ":SBUS<n>:MANChester:BASE" on page 830
- ":SBUS<n>:MANChester:IDLE:BITS" on page 836
- ":SBUS<n>:MANChester:LOGic" on page 837
- ":SBUS<n>:MANChester:STARt" on page 840

:SBUS<n>:MANChester:DISPlay

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:DISPlay <format>

<format> ::= {BIT | WORD}
```

The :SBUS<n>:MANChester:DISPlay command specifies the format of the Manchester bus display.

Query Syntax

```
:SBUS<n>:MANChester:DISPlay?
```

The :SBUS<n>:MANChester:DISPlay? query returns the bus display format setting.

Return Format

```
<format><NL>

<format> ::= {BIT | WORD}
```

See Also

- ":SBUS<n>:MANChester:DSIZe" on page 834
- ":SBUS<n>:MANChester:HSIZe" on page 835
- ":SBUS<n>:MANChester:SSIZe" on page 839
- ":SBUS<n>:MANChester:TSIZe" on page 845
- ":SBUS<n>:MANChester:WSIZe" on page 846

:SBUS<n>:MANChester:DSIZe

 (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:DSIZe {AUTO | <#words>}

<#words> ::= from 1-255, in NR1 format
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:DSIZe command specifies the number of words in the data field of your Manchester protocol definition.

AUTO is available as a selection only when the trailer field size is 0 (see :SBUS<n>:MANChester:TSIZe).

Query Syntax

```
:SBUS<n>:MANChester:DSIZe?
```

The :SBUS<n>:MANChester:DSIZe? query returns the number of data field words setting.

Return Format

```
<#words><NL>

<#words> ::= from 0-255, in NR1 format
```

In AUTO mode, the query returns 0.

In BIT format the only legal value is 0 (for AUTO).

See Also

- ":SBUS<n>:MANChester:DISPlay" on page 833
- ":SBUS<n>:MANChester:HSIZe" on page 835
- ":SBUS<n>:MANChester:SSIZe" on page 839
- ":SBUS<n>:MANChester:TSIZe" on page 845
- ":SBUS<n>:MANChester:WSIZe" on page 846

:SBUS<n>:MANChester:HSIZe

 (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:HSIZe <#bits>

<#bits> ::= from 0-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:HSIZe command specifies the number of bits in the header field of your Manchester protocol definition.

Query Syntax

```
:SBUS<n>:MANChester:HSIZe?
```

The :SBUS<n>:MANChester:HSIZe? query returns the number of header field bits setting.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:MANChester:DISPlay" on page 833
- ":SBUS<n>:MANChester:DSIZe" on page 834
- ":SBUS<n>:MANChester:SSIZe" on page 839
- ":SBUS<n>:MANChester:TSIZe" on page 845
- ":SBUS<n>:MANChester:WSIZe" on page 846

:SBUS<n>:MANChester:IDLE:BITS

 (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:IDLE:BITS <#bits>

<#bits> ::= minimum idle time, from 1.50 to 32.00 in 0.25 increments,
   in NR3 format.
```

The :SBUS<n>:MANChester:IDLE:BITS command specifies the minimum idle time or inter-frame gap time in terms of the number of bits.

Query Syntax

```
:SBUS<n>:MANChester:IDLE:BITS?
```

The :SBUS<n>:MANChester:IDLE:BITS? query returns the specified idle time in terms of the number of bits.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:MANChester:BASE" on page 830
- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:LOGic" on page 837
- ":SBUS<n>:MANChester:STARt" on page 840

:SBUS<n>:MANChester:LOGic

N (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:LOGic <logic>

<logic> ::= {FALLing | RISing}
```

The :SBUS<n>:MANChester:LOGic command specifies the polarity of the Manchester signal:

- FALLing – specifies that a falling edge is used to encode a bit value of logic 1 (and a rising edge encodes a bit value of logic 0).
- RISing – specifies that a rising edge is used to encode a bit value of logic 1.

Query Syntax

```
:SBUS<n>:MANChester:LOGic?
```

The :SBUS<n>:MANChester:LOGic? query returns the polarity setting.

Return Format

```
<logic><NL>

<logic> ::= {FALL | RIS}
```

See Also

- ":SBUS<n>:MANChester:BASE" on page 830
- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:IDLE:BITS" on page 836
- ":SBUS<n>:MANChester:STARt" on page 840

:SBUS<n>:MANChester:SOURce

 (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:MANChester:SOURce command selects the oscilloscope channel connected to the Manchester signal line.

Query Syntax

```
:SBUS<n>:MANChester:SOURce?
```

The :SBUS<n>:MANChester:SOURce? query returns the selected oscilloscope channel source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":SBUS<n>:MANChester:BAUDrate" on page 831
- ":SBUS<n>:MANChester:TOLerance" on page 841

:SBUS<n>:MANChester:SSIZe

 (see page 1354)

Command Syntax `:SBUS<n>:MANChester:SSIZe <#bits>`

`<#bits> ::= from 0-255, in NR1 format`

The :SBUS<n>:MANChester:SSIZe command specifies the number of sync bits for the Manchester signal.

Query Syntax `:SBUS<n>:MANChester:SSIZe?`

The :SBUS<n>:MANChester:SSIZe? query returns the number of sync bits setting.

Return Format `<#bits><NL>`

See Also

- ":SBUS<n>:MANChester:DISPlay" on page 833
- ":SBUS<n>:MANChester:DSIZe" on page 834
- ":SBUS<n>:MANChester:HSIZe" on page 835
- ":SBUS<n>:MANChester:TSIZe" on page 845
- ":SBUS<n>:MANChester:WSIZe" on page 846

:SBUS<n>:MANChester:STARt

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:STARt <edge#>

<edge#> ::= from 1-256, in NR1 format
```

The :SBUS<n>:MANChester:STARt command specifies the starting edge of the Manchester signal.

Query Syntax

```
:SBUS<n>:MANChester:STARt?
```

The :SBUS<n>:MANChester:STARt? query returns the starting edge number setting.

Return Format

```
<edge#><NL>
```

See Also

- ":SBUS<n>:MANChester:BASE" on page 830
- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:IDLE:BITS" on page 836
- ":SBUS<n>:MANChester:LOGic" on page 837

:SBUS<n>:MANChester:TOLerance

 (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:TOLerance <percent>

<percent> ::= from 5-30, in NR1 format
```

The :SBUS<n>:MANChester:TOLerance command specifies the tolerance for the Manchester signal in terms of the percentage of the bit period.

Query Syntax

```
:SBUS<n>:MANChester:TOLerance?
```

The :SBUS<n>:MANChester:TOLerance? query returns the tolerance setting.

Return Format

```
<percent><NL>

<percent> ::= from 5-30, in NR1 format
```

See Also

- ":SBUS<n>:MANChester:BAUDrate" on page 831
- ":SBUS<n>:MANChester:SOURce" on page 838

:SBUS<n>:MANChester:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:TRIGger <mode>

<mode> ::= {SOF | VALue | MERRor}
```

The :SBUS<n>:MANChester:TRIGger command specifies the trigger mode:

- SOF (Start Of Frame) – triggers at the start of a Manchester frame, after the starting edge.
- VALue – triggers on the specified bit values.
- MERRor – triggers on a Manchester error.

Query Syntax

```
:SBUS<n>:MANChester:TRIGger?
```

The :SBUS<n>:MANChester:TRIGger? query returns the trigger mode setting.

Return Format

```
<mode><NL>

<mode> ::= {SOF | VAL | MERR}
```

See Also

- ":SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA" on page 843
- ":SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh" on page 844

:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

When the VALue trigger mode is selected (:SBUS<n>:MANChester:TRIGger), the :SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA command specifies the value to trigger on.

Note that the trigger value bit order is always for bits as they arrive (that is, MSB first) regardless of the serial decode bit order setting (in :SBUS<n>:MANChester:BITorder).

The bit width (length) of the value is set with the :SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh command.

Query Syntax

```
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA?
```

The :SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA? query returns the specified trigger value as a string of binary digits.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}
```

See Also

- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:TRIGger" on page 842
- ":SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh" on page 844

:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh <width>

<width> ::= integer from 4 to 128 in NR1 format
```

When the VALue trigger mode is selected (:SBUS<n>:MANChester:TRIGger), the :SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh command specifies the bit width (length) of the value to trigger on.

The actual value to trigger on is set with the :SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA command.

Query Syntax

```
:SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh?
```

The :SBUS<n>:MANChester:TRIGger:PATTern:VALue:WIDTh? query returns the specified trigger value bit width (length).

Return Format

```
<width><NL>
```

See Also

- ":SBUS<n>:MANChester:BITorder" on page 832
- ":SBUS<n>:MANChester:TRIGger" on page 842
- ":SBUS<n>:MANChester:TRIGger:PATTern:VALue:DATA" on page 843

:SBUS<n>:MANChester:TSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:TSIZe <#bits>

<#bits> ::= from 0-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:TSIZe command specifies the number of bits in the trailer field of your Manchester protocol definition.

Query Syntax

```
:SBUS<n>:MANChester:TSIZe?
```

The :SBUS<n>:MANChester:TSIZe? query returns the number of trailer field bits setting.

Return Format

```
<#bits><NL>

<#bits> ::= from 0-32, in NR1 format
```

See Also

- ":SBUS<n>:MANChester:DISPlay" on page 833
- ":SBUS<n>:MANChester:DSIZe" on page 834
- ":SBUS<n>:MANChester:HSIZe" on page 835
- ":SBUS<n>:MANChester:SSIZe" on page 839
- ":SBUS<n>:MANChester:WSIZe" on page 846

:SBUS<n>:MANChester:WSIZe

(see page 1354)

Command Syntax

```
:SBUS<n>:MANChester:WSIZe <#bits>

<#bits> ::= from 2-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:MANChester:DISPlay), the :SBUS<n>:MANChester:WSIZe command the number of bits per word in the data field of your Manchester protocol definition.

Query Syntax

```
:SBUS<n>:MANChester:WSIZe?
```

The :SBUS<n>:MANChester:WSIZe? query returns the number of bits per word setting.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:MANChester:DISPlay" on page 833
- ":SBUS<n>:MANChester:DSIZe" on page 834
- ":SBUS<n>:MANChester:HSIZe" on page 835
- ":SBUS<n>:MANChester:SSIZe" on page 839
- ":SBUS<n>:MANChester:TSIZe" on page 845

:SBUS<n>:NRZ Commands

NOTE These commands are valid when the automotive NRZ serial decode and triggering option has been licensed.

Table 110 :SBUS<n>:NRZ Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:NRZ:BASE <base> (see page 849)	:SBUS<n>:NRZ:BASE? (see page 849)	<base> ::= {HEX | DECimal | ASCii}
:SBUS<n>:NRZ:BAUDrate <baudrate> (see page 850)	:SBUS<n>:NRZ:BAUDrate? (see page 850)	<baudrate> ::= integer from 5000 to 5000000 in 100 b/s increments
:SBUS<n>:NRZ:BITorder <bitorder> (see page 851)	:SBUS<n>:NRZ:BITorder? (see page 851)	<bitorder> ::= {MSBFirst | LSBFirst}
:SBUS<n>:NRZ:DISPlay <format> (see page 852)	:SBUS<n>:NRZ:DISPlay? (see page 852)	<format> ::= {BIT | WORD}
:SBUS<n>:NRZ:DSIZe <#words> (see page 853)	:SBUS<n>:NRZ:DSIZe? (see page 853)	<#words> ::= from 1-255, in NR1 format
:SBUS<n>:NRZ:FSIZe <#bits> (see page 854)	:SBUS<n>:NRZ:FSIZe? (see page 854)	<#bits> ::= from 2-255, in NR1 format
:SBUS<n>:NRZ:HSIZe <#bits> (see page 855)	:SBUS<n>:NRZ:HSIZe? (see page 855)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:NRZ:IDLE:BITS <#bits> (see page 856)	:SBUS<n>:NRZ:IDLE:BITS? (see page 856)	<#bits> ::= minimum idle time in terms of bit width, from 1.50 to 32.0, in NR3 format.
:SBUS<n>:NRZ:IDLE:STATe <state> (see page 857)	:SBUS<n>:NRZ:IDLE:STATe? (see page 857)	<state> ::= {LOW | HIGH}
:SBUS<n>:NRZ:LOGic <logic> (see page 858)	:SBUS<n>:NRZ:LOGic? (see page 858)	<logic> ::= {HIGH | LOW}
:SBUS<n>:NRZ:SOURce <source> (see page 859)	:SBUS<n>:NRZ:SOURce? (see page 859)	<source> ::= {CHANnel<n>}
:SBUS<n>:NRZ:STARt <#bits> (see page 860)	:SBUS<n>:NRZ:STARt? (see page 860)	<#bits> ::= from 0-255, in NR1 format
:SBUS<n>:NRZ:TRIGger <mode> (see page 861)	:SBUS<n>:NRZ:TRIGger? (see page 861)	<mode> ::= {SOF | VALue}

Table 110 :SBUS<n>:NRZ Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA <string> (see page 862)	:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA? (see page 862)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh <width> (see page 863)	:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh? (see page 863)	<width> ::= integer from 4 to 128 in NR1 format
:SBUS<n>:NRZ:TSIZe <#bits> (see page 864)	:SBUS<n>:NRZ:TSIZe? (see page 864)	<#bits> ::= from 0-32, in NR1 format
:SBUS<n>:NRZ:WSIZe <#bits> (see page 865)	:SBUS<n>:NRZ:WSIZe? (see page 865)	<#bits> ::= from 2-32, in NR1 format

:SBUS<n>:NRZ:BASE

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:BASE <base>

<base> ::= {HEX | DECimal | ASCii | BINary}
```

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:BASE command specifies the base for the NRZ bus decode and Lister display.

- HEX – hexadecimal
- DECimal – unsigned decimal
- ASCii

When the display format is BIT, the only legal decode base value is BINary.

Query Syntax

```
:SBUS<n>:NRZ:BASE?
```

The :SBUS<n>:NRZ:BASE? query returns the decode number base setting.

Return Format

```
<base><NL>

<base> ::= {HEX | DEC | ASC | BIN}
```

See Also

- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:IDLE:BITS" on page 856
- ":SBUS<n>:NRZ:IDLE:STATe" on page 857
- ":SBUS<n>:NRZ:LOGic" on page 858

:SBUS<n>:NRZ:BAUDrate

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:BAUDrate <baudrate>

<baudrate> ::= integer from 5000 to 5000000 in 100 b/s increments
```

The :SBUS<n>:NRZ:BAUDrate command specifies the baud rate of the NRZ signal.

Query Syntax

```
:SBUS<n>:NRZ:BAUDrate?
```

The :SBUS<n>:NRZ:BAUDrate? query returns the specified baud rate.

Return Format

```
<baudrate><NL>
```

See Also

- ":SBUS<n>:NRZ:SOURce" on page 859

:SBUS<n>:NRZ:BITorder

(see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:BITorder <bitorder>

<bitorder> ::= {MSBFirst | LSBFirst}
```

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:BITorder command specifies the order of transmission on the NRZ bus:

- MSBFirst – specifies the most significant bit is transmitted first.
- LSBFirst – specifies the least significant bit is transmitted first.

Query Syntax

```
:SBUS<n>:NRZ:BITorder?
```

The :SBUS<n>:NRZ:BITorder? query returns the bit order setting.

Return Format

```
<bitorder><NL>

<bitorder> ::= {MSBF | LSBF}
```

See Also

- ":SBUS<n>:NRZ:BASE" on page 849
- ":SBUS<n>:NRZ:IDLE:BITS" on page 856
- ":SBUS<n>:NRZ:IDLE:STATe" on page 857
- ":SBUS<n>:NRZ:LOGic" on page 858

:SBUS<n>:NRZ:DISPlay

(see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:DISPlay <format>

<format> ::= {BIT | WORD}
```

The :SBUS<n>:NRZ:DISPlay command specifies the format of the NRZ bus display.

Query Syntax

```
:SBUS<n>:NRZ:DISPlay?
```

The :SBUS<n>:NRZ:DISPlay? query returns the bus display format setting.

Return Format

```
<format><NL>

<format> ::= {BIT | WORD}
```

See Also

- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:TSIZe" on page 864
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:DSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:DSIZe <#words>

<#words> ::= from 1-255, in NR1 format
```

The :SBUS<n>:NRZ:DSIZe command ...

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:DSIZe command specifies the number of words in the data field of your NRZ protocol definition.

AUTO is available as a selection only when the trailer field size is 0 (see :SBUS<n>:NRZ:TSIZe).

Query Syntax

```
:SBUS<n>:NRZ:DSIZe?
```

The :SBUS<n>:NRZ:DSIZe? query returns the number of data field words setting.

Return Format

```
<#words><NL>

<#words> ::= from 1-255, in NR1 format
```

In AUTO mode, the query returns 0.

In BIT format the only legal value is 0 (for AUTO).

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:TSIZe" on page 864
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:FSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:FSIZe <#bits>

<#bits> ::= from 2-255, in NR1 format
```

When the NRZ bus display format (:SBUS<n>:NRZ:DISPlay) is BIT, the :SBUS<n>:NRZ:FSIZe command lets you specify the total frame size of the NRZ signal from 2 to 255 bits. This would be equivalent to the sum of the number of bits in the header, data, and trailer fields in WORD format.

Query Syntax

```
:SBUS<n>:NRZ:FSIZe?
```

The :SBUS<n>:NRZ:FSIZe? query returns the specified total frame size.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:TSIZe" on page 864
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:HSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:HSIZe <#bits>

<#bits> ::= from 0-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:HSIZe command specifies the number of bits in the header field of your NRZ protocol definition.

Query Syntax

```
:SBUS<n>:NRZ:HSIZe?
```

The :SBUS<n>:NRZ:HSIZe? query returns the number of header field bits setting.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:TSIZe" on page 864
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:IDLE:BITS

(see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:IDLE:BITS <#bits>

<#bits> ::= minimum idle time, from 1.50 to 32.00 in 0.25 increments,
   in NR3 format.
```

The :SBUS<n>:NRZ:IDLE:BITS command specifies the minimum idle time or inter-frame gap time in terms of the number of bits.

Query Syntax

```
:SBUS<n>:NRZ:IDLE:BITS?
```

The :SBUS<n>:NRZ:IDLE:BITS? query returns the specified idle time in terms of the number of bits.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:NRZ:BASE" on page 849
- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:IDLE:STATe" on page 857
- ":SBUS<n>:NRZ:LOGic" on page 858

:SBUS<n>:NRZ:IDLE:STATe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:IDLE:STATe <state>

<state> ::= {LOW | HIGH}
```

The :SBUS<n>:NRZ:IDLE:STATe command specifies the idle state of the NRZ signal.

Query Syntax

```
:SBUS<n>:NRZ:IDLE:STATe?
```

The :SBUS<n>:NRZ:IDLE:STATe? query returns the idle state setting.

Return Format

```
<state><NL>

<state> ::= {LOW | HIGH}
```

See Also

- ":SBUS<n>:NRZ:BASE" on page 849
- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:IDLE:BITS" on page 856
- ":SBUS<n>:NRZ:LOGic" on page 858

:SBUS<n>:NRZ:LOGic

 (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:LOGic <logic>

<logic> ::= {HIGH | LOW}
```

The :SBUS<n>:NRZ:LOGic command specifies the polarity of the NRZ signal:

- HIGH – specifies that a positive voltage is used to encode a bit value of logic 1 (and a negative voltage encodes a bit value of logic 0).
- LOW – specifies that a negative voltage is used to encode a bit value of logic 1.

Query Syntax

```
:SBUS<n>:NRZ:LOGic?
```

The :SBUS<n>:NRZ:LOGic? query returns the polarity setting.

Return Format

```
<logic><NL>

<logic> ::= {HIGH | LOW}
```

See Also

- ":SBUS<n>:NRZ:BASE" on page 849
- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:IDLE:BITS" on page 856
- ":SBUS<n>:NRZ:IDLE:STATe" on page 857

:SBUS<n>:NRZ:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:NRZ:SOURce command selects the oscilloscope channel connected to the NRZ signal

Query Syntax

```
:SBUS<n>:NRZ:SOURce?
```

The :SBUS<n>:NRZ:SOURce? query returns the selected oscilloscope channel source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":SBUS<n>:NRZ:BAUDrate" on page 850

:SBUS<n>:NRZ:STARt

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:STARt <#bits>

<#bits> ::= from 0-255, in NR1 format
```

The :SBUS<n>:NRZ:STARt command specifies the number of start bits for the NRZ signal.

Query Syntax

```
:SBUS<n>:NRZ:STARt?
```

The :SBUS<n>:NRZ:STARt? query returns the number of start bits setting.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:TSIZe" on page 864
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:TRIGger <mode>

<mode> ::= {SOF | VALue}
```

The :SBUS<n>:NRZ:TRIGger command specifies the trigger mode:

- SOF (Start Of Frame) — triggers at the start of a NRZ frame, before the header field.
- VALue — triggers on the specified bit values (see :SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh and :SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA), up to 128 bits, after the specified number of starting bits.

  Note that the trigger value is always for bits as they arrive (that is, MSB first).

  When the decode bit order (see :SBUS<n>:NRZ:BITorder) is MSBFirst, the decoded value bit order matches the trigger value bit order.

  When the serial decode bit order is LSBFirst, the decoded value bit order is opposite of the trigger value bit order.

Query Syntax

```
:SBUS<n>:NRZ:TRIGger?
```

The :SBUS<n>:NRZ:TRIGger? query returns returns the trigger mode setting.

Return Format

```
<mode><NL>

<mode> ::= {SOF | VAL}
```

See Also

- ":SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA" on page 862
- ":SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh" on page 863

:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA

 (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

When the VALue trigger mode is selected (:SBUS<n>:NRZ:TRIGger), the :SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA command specifies the value to trigger on.

Note that the trigger value bit order is always for bits as they arrive (that is, MSB first) regardless of the serial decode bit order setting (in :SBUS<n>:NRZ:BITorder).

The bit width (length) of the value is set with the :SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh command.

Query Syntax

```
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA?
```

The :SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA? query returns the specified trigger value as a string of binary digits.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}
```

See Also

- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:TRIGger" on page 861
- ":SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh" on page 863

:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh <width>

<width> ::= integer from 4 to 128 in NR1 format
```

When the VALue trigger mode is selected (:SBUS<n>:NRZ:TRIGger), the :SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh command specifies the bit width (length) of the value to trigger on.

The actual value to trigger on is set with the :SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA command.

Query Syntax

```
:SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh?
```

The :SBUS<n>:NRZ:TRIGger:PATTern:VALue:WIDTh? query returns the specified trigger value bit width (length).

Return Format

```
<width><NL>
```

See Also

- ":SBUS<n>:NRZ:BITorder" on page 851
- ":SBUS<n>:NRZ:TRIGger" on page 861
- ":SBUS<n>:NRZ:TRIGger:PATTern:VALue:DATA" on page 862

:SBUS<n>:NRZ:TSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:TSIZe <#bits>

<#bits> ::= from 0-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:TSIZe command specifies the number of bits in the trailer field of your NRZ protocol definition.

Query Syntax

```
:SBUS<n>:NRZ:TSIZe?
```

The :SBUS<n>:NRZ:TSIZe? query returns the number of trailer field bits setting.

Return Format

```
<opt><NL>

<#bits> ::= from 0-32, in NR1 format
```

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:WSIZe" on page 865

:SBUS<n>:NRZ:WSIZe

N (see page 1354)

Command Syntax

```
:SBUS<n>:NRZ:WSIZe <#bits>

<#bits> ::= from 2-32, in NR1 format
```

When the display format is WORD (see :SBUS<n>:NRZ:DISPlay), the :SBUS<n>:NRZ:WSIZe command specifies the number of bits per word in the data field of your NRZ protocol definition.

Query Syntax

```
:SBUS<n>:NRZ:WSIZe?
```

The :SBUS<n>:NRZ:WSIZe? query returns the number of bits per word setting.

Return Format

```
<#bits><NL>
```

See Also

- ":SBUS<n>:NRZ:DISPlay" on page 852
- ":SBUS<n>:NRZ:DSIZe" on page 853
- ":SBUS<n>:NRZ:FSIZe" on page 854
- ":SBUS<n>:NRZ:HSIZe" on page 855
- ":SBUS<n>:NRZ:STARt" on page 860
- ":SBUS<n>:NRZ:TSIZe" on page 864

:SBUS<n>:SENT Commands

NOTE These commands are valid when the automotive SENT serial decode and triggering option has been licensed.

Table 111 :SBUS<n>:SENT Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SENT:CLOCk <period> (see page 868)	:SBUS<n>:SENT:CLOCk? (see page 868)	<period> ::= the nominal clock period (tick), from 500 ns to 300 us, in NR3 format.
:SBUS<n>:SENT:CRC <format> (see page 869)	:SBUS<n>:SENT:CRC? (see page 869)	<format> ::= {LEGacy | RECommended}
:SBUS<n>:SENT:DISPlay <base> (see page 870)	:SBUS<n>:SENT:DISPlay? (see page 870)	<base> ::= {HEX | DECimal | SYMBolic}
:SBUS<n>:SENT:FORMat <decode> (see page 872)	:SBUS<n>:SENT:FORMat? (see page 872)	<decode> ::= {NIBBles | FSIGnal | FSSerial | FESerial | SSERial | ESERial}
:SBUS<n>:SENT:IDLE <state> (see page 874)	:SBUS<n>:SENT:IDLE? (see page 874)	<state> ::= {LOW | HIGH}
:SBUS<n>:SENT:LENGth <#_nibbles> (see page 875)	:SBUS<n>:SENT:LENGth? (see page 875)	<#_nibbles> ::= from 1-6, in NR1 format.
:SBUS<n>:SENT:PPULse {{0 | OFF} | {1 | ON} | SPC} (see page 876)	:SBUS<n>:SENT:PPULse? (see page 876)	{0 | 1 | SPC}
:SBUS<n>:SENT:SIGNal<s>:DISPlay {{0 | OFF} | {1 | ON}} (see page 878)	:SBUS<n>:SENT:SIGNal<s>:DISPlay? (see page 878)	<s> ::= 1-6, in NR1 format. {0 | 1}
:SBUS<n>:SENT:SIGNal<s>:LENGth <length> (see page 879)	:SBUS<n>:SENT:SIGNal<s>:LENGth? (see page 879)	<s> ::= 1-6, in NR1 format. <length> ::= from 1-24, in NR1 format.
:SBUS<n>:SENT:SIGNal<s>:MULTiplier <multiplier> (see page 881)	:SBUS<n>:SENT:SIGNal<s>:MULTiplier? (see page 881)	<s> ::= 1-6, in NR1 format. <multiplier> ::= from 1-24, in NR3 format.
:SBUS<n>:SENT:SIGNal<s>:OFFSet <offset> (see page 883)	:SBUS<n>:SENT:SIGNal<s>:OFFSet? (see page 883)	<s> ::= 1-6, in NR1 format. <offset> ::= from 1-24, in NR3 format.

Table 111 :SBUS<n>:SENT Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SENT:SIGNal<s>:ORDer <order> (see page 885)	:SBUS<n>:SENT:SIGNal<s>:ORDer? (see page 885)	<s> ::= 1-6, in NR1 format. <order> ::= {MSNFirst | LSNFirst}
:SBUS<n>:SENT:SIGNal<s>:STARt <position> (see page 887)	:SBUS<n>:SENT:SIGNal<s>:STARt? (see page 887)	<s> ::= 1-6, in NR1 format. <position> ::= from 0-23, in NR1 format.
:SBUS<n>:SENT:SOURce <source> (see page 889)	:SBUS<n>:SENT:SOURce? (see page 889)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:SENT:TOLerance <percent> (see page 890)	:SBUS<n>:SENT:TOLerance? (see page 890)	<percent> ::= from 3-30, in NR1 format.
:SBUS<n>:SENT:TRIGger <mode> (see page 891)	:SBUS<n>:SENT:TRIGger? (see page 891)	<mode> ::= {SFCMessage | SSCMessage | FCData | SCMid | SCData | FCCerror | SCCerror | CRCerror | TOLerror | PPERror | SSPerror}
:SBUS<n>:SENT:TRIGger:FAST:DATA <string> (see page 893)	:SBUS<n>:SENT:TRIGger:FAST:DATA? (see page 893)	<string> ::= "nnnn..." where n ::= {0 | 1 | X} <string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SENT:TRIGger:SLOW:DATA <data> (see page 894)	:SBUS<n>:SENT:TRIGger:SLOW:DATA? (see page 894)	<data> ::= when ILENgth = SHORt, from -1 (don't care) to 65535, in NR1 format. <data> ::= when ILENgth = LONG, from -1 (don't care) to 4095, in NR1 format.
:SBUS<n>:SENT:TRIGger:SLOW:ID <id> (see page 896)	:SBUS<n>:SENT:TRIGger:SLOW:ID? (see page 896)	<id> ::= when ILENgth = SHORt, from -1 (don't care) to 15, in NR1 format. <id> ::= when ILENgth = LONG, from -1 (don't care) to 255, in NR1 format.
:SBUS<n>:SENT:TRIGger:SLOW:ILENgth <length> (see page 898)	:SBUS<n>:SENT:TRIGger:SLOW:ILENgth? (see page 898)	<length> ::= {SHORt | LONG}
:SBUS<n>:SENT:TRIGger:TOLerance <percent> (see page 899)	:SBUS<n>:SENT:TRIGger:TOLerance? (see page 899)	<percent> ::= from 1-18, in NR1 format.

:SBUS<n>:SENT:CLOCk

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:CLOCk <period>

<period> ::= the nominal clock period (tick), from 500 ns to 300 us,
   in NR3 format.
```

The :SBUS<n>:SENT:CLOCk command specifies the nominal clock period (tick), from 500 ns to 300 µs.

Query Syntax

```
:SBUS<n>:SENT:CLOCk?
```

The :SBUS<n>:SENT:CLOCk? query returns the clock period setting.

Return Format

```
<period><NL>

<period> ::= the nominal clock period (tick), from 500 ns to 300 us,
   in NR3 format.
```

See Also

- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:CRC

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:CRC <format>

<format> ::= {LEGacy | RECommended}
```

The :SBUS<n>:SENT:CRC command specifies the format of the CRC. Either Legacy (2008) or Recommended (2010).

Enhanced Serial Message CRCs are always calculated using the 2010 format, but for the Fast Channel Messages, and for Short Serial Message CRCs, this setting is used.

Query Syntax

```
:SBUS<n>:SENT:CRC?
```

The :SBUS<n>:SENT:CRC? query returns the CRC format setting.

Return Format

```
<format><NL>

<format> ::= {LEG | REC}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:DISPlay

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:DISPlay <base>

<base> ::= {HEX | DECimal | SYMBolic}
```

The :SBUS<n>:SENT:DISPlay command specifies the number base used by the decoder. The chosen base is used for the data nibbles in Raw decode format, the defined Signals in the other formats, and for the data field of the Serial Messages.

This selection is used for both the Lister and the decode line displays.

When SYMBolic is selected, Fast Channel Signals display a calculated physical value based on the specified multiplier and offset:

- PhysicalValue = (Multiplier * SignalValueAsUnsignedInteger) + Offset

When SYMBolic is selected, the CRC and Slow Channel information is displayed in hex.

Query Syntax

```
:SBUS<n>:SENT:DISPlay?
```

The :SBUS<n>:SENT:DISPlay? query returns the SENT decode number base setting.

Return Format

```
<base><NL>

<base> ::= {HEX | DEC | SYMB}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893

:SBUS<n>:SENT:FORMat

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:FORMat <decode>

<decode> ::= {NIBBles | FSIGnal | FSSerial | FESerial | SSERial | ESERial}
```

The :SBUS<n>:SENT:FORMat command specifies the message decode/triggering format:

- NIBBles – displays the raw transmitted nibble values.
- FSIGnal – displays Fast Channel Message Signals.
- FSSerial – displays both Fast and Slow Messages (Short format) simultaneously.
- FESerial – displays both Fast and Slow Messages (Enhanced format) simultaneously.
- SSERial – displays Slow Channel Messages in Short format.
- ESERial – displays Slow Channel Messages in Enhanced format.

This selection affects both decoding and triggering. The decode is affected both in how the system interprets the data, and what will be displayed. The trigger is affected in that the trigger hardware needs to be configured to trigger on serial messages correctly.

You can specify the nibble display order for Fast Channel Message Signals (see :SBUS<n>:SENT:SIGNal<s>:ORDer). Raw transmitted nibble values are displayed in the order received.

Note that for the Slow Channel, the proper format, Short or Enhanced, must be chosen for proper decoding and triggering to occur.

Slow Channel Serial Messages are always displayed as defined by the SENT specification.

Query Syntax

```
:SBUS<n>:SENT:FORMat?
```

The :SBUS<n>:SENT:FORMat? query returns the message decode/triggering format setting.

Return Format

```
<decode><NL>

<decode> ::= {NIBB | FSIG | FSS | FES | SSER | ESER}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:IDLE" on page 874

:SBUS<n>:SENT:IDLE

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:IDLE <state>

<state> ::= {LOW | HIGH}
```

The :SBUS<n>:SENT:IDLE command specifies the idle state of the SENT bus.

Query Syntax

```
:SBUS<n>:SENT:IDLE?
```

The :SBUS<n>:SENT:IDLE? query returns the idle state setting.

Return Format

```
<state><NL>

<state> ::= {LOW | HIGH}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:LENGth

N (see page 1354)

Command Syntax `:SBUS<n>:SENT:LENGth <#_nibbles>`

`<#_nibbles> ::= from 1-6, in NR1 format.`

The :SBUS<n>:SENT:LENGth command specifies the number of nibbles in a SENT message, from 1 to 6.

Query Syntax `:SBUS<n>:SENT:LENGth?`

The :SBUS<n>:SENT:LENGth? query returns the number of nibbles setting.

Return Format `<#_nibbles><NL>`

`<#_nibbles> ::= from 1-6, in NR1 format.`

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:PPULse

(see page 1354)

Command Syntax `:SBUS<n>:SENT:PPULse {{0 | OFF} | {1 | ON} | SPC}`

The :SBUS<n>:SENT:PPULse command specifies whether there is a pause pulse between Fast Channel Messages:

- OFF — There is no pause pulse between Fast Channel Messages.

  Note that a SENT serial bus with no pause pulse is never idle. This means, during normal operation, the fast channel decode line will show a continuous stream of packets, with a new packet opening as soon as the previous one has closed.

- ON — Pause pulses are added between Fast Channel Messages so that frames come at a regular interval.

  If there is a pause pulse (and **Pause Pulse** is on), idle time is shown between messages.

- SPC (Short PWM Code) — In SENT SPC, there are no pause pulses. Instead, the message event is triggered by the master when it wants to receive data. SENT SPC ends the transmission after the CRC so it almost appears as if there is a pause pulse from the end until the next master trigger.

Query Syntax `:SBUS<n>:SENT:PPULse?`

The :SBUS<n>:SENT:PPULse? query returns the pause mode setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1 | SPC}`

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889

:SBUS<n>:SENT:SIGNal<s>:DISPlay

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:DISPlay {{0 | OFF} | {1 | ON}}

<s> ::= 1-6, in NR1 format.
```

The :SBUS<n>:SENT:SIGNal<s>:DISPlay command specifies whether the given signal is on or off.

Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:DISPlay?
```

The :SBUS<n>:SENT:SIGNal<s>:DISPlay? query returns the signal on/off setting.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:SIGNal<s>:LENGth

N (see page 1354)

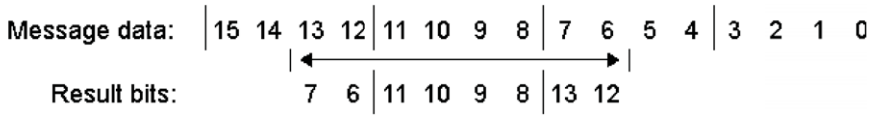
Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:LENGth <length>

<s> ::= 1-6, in NR1 format.

<length> ::= from 1-24, in NR1 format.
```

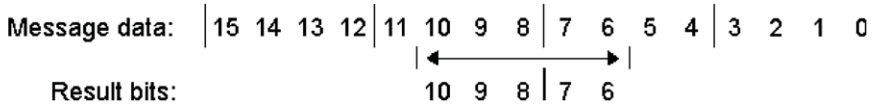
The :SBUS<n>:SENT:SIGNal<s>:LENGth command specifies the bit length of the signal being defined.

Fast Signal definition examples:

Example 1: Start Bit # = 13, # of bits = 8, Nibble Order = LSN First



Example 2: Start Bit # = 10, # of bits = 5, Nibble Order = MSN First



Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:LENGth?
```

The :SBUS<n>:SENT:SIGNal<s>:LENGth? query returns the signal bit length setting.

Return Format

```
<length><NL>

<length> ::= from 1-24, in NR1 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883

:SBUS<n>:SENT:SIGNal<s>:MULTiplier

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:MULTiplier <multiplier>

<s> ::= 1-6, in NR1 format.

<multiplier> ::= from 1-24, in NR3 format.
```

When the display mode setting is SYMBolic (see :SBUS<n>:SENT:DISPlay), the :SBUS<n>:SENT:SIGNal<s>:MULTiplier command specifies the multiplier to be used in calculating a physical value displayed for a Fast Channel Signal.

- PhysicalValue = (Multiplier * SignalValueAsUnsignedInteger) + Offset

Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:MULTiplier?
```

The :SBUS<n>:SENT:SIGNal<s>:MULTiplier? query returns the multiplier value for the Fast Channel Signal.

Return Format

```
<multiplier><NL>

<multiplier> ::= from 1-24, in NR3 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898

- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:SIGNal<s>:OFFSet

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:OFFSet <offset>

<s> ::= 1-6, in NR1 format.

<offset> ::= from 1-24, in NR3 format.
```

When the display mode setting is SYMBolic (see :SBUS<n>:SENT:DISPlay), the :SBUS<n>:SENT:SIGNal<s>:OFFSet command is used in calculating a physical value displayed for the Fast Channel Signal:

- PhysicalValue = (Multiplier * SignalValueAsUnsignedInteger) + Offset

Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:OFFSet?
```

The :SBUS<n>:SENT:SIGNal<s>:OFFSet? query returns the offset value for the Fast Channel Signal.

Return Format

```
<offset><NL>

<offset> ::= from 1-24, in NR3 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898

- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:SIGNal<s>:ORDer

N (see page 1354)

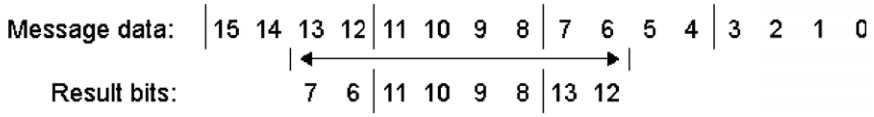
Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:ORDer <order>

<s> ::= 1-6, in NR1 format.

<order> ::= {MSNFirst | LSNFirst}
```

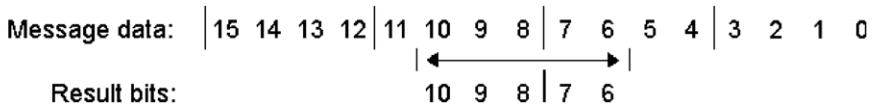
The :SBUS<n>:SENT:SIGNal<s>:ORDer command specifies the nibble order of the signal being defined, either Most Significant Nibble first, or Least Significant Nibble first.

Fast Signal definition examples:

Example 1: Start Bit # = 13, # of bits = 8, Nibble Order = LSN First



Example 2: Start Bit # = 10, # of bits = 5, Nibble Order = MSN First



Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:ORDer?
```

The :SBUS<n>:SENT:SIGNal<s>:ORDer? query returns the nibble order setting.

Return Format

```
<order><NL>

<order> ::= {MSNF | LSNF}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881

:SBUS<n>:SENT:SIGNal<s>:STARt

N (see page 1354)

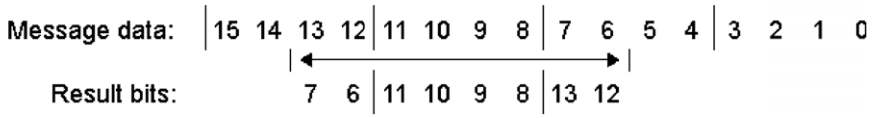
Command Syntax

```
:SBUS<n>:SENT:SIGNal<s>:STARt <position>

<s> ::= 1-6, in NR1 format.

<position> ::= from 0-23, in NR1 format.
```

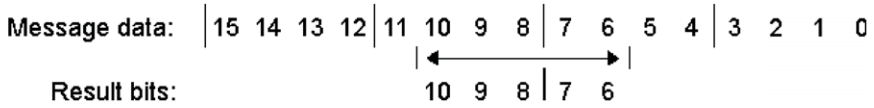
The :SBUS<n>:SENT:SIGNal<s>:STARt command specifies the starting bit of the Fast Signal being defined.

Fast Signal definition examples:

Example 1: Start Bit # = 13, # of bits = 8, Nibble Order = LSN First



Example 2: Start Bit # = 10, # of bits = 5, Nibble Order = MSN First



Query Syntax

```
:SBUS<n>:SENT:SIGNal<s>:STARt?
```

The :SBUS<n>:SENT:SIGNal<s>:STARt? query returns the Fast Signal starting bit setting.

Return Format

```
<position><NL>

<position> ::= from 0-23, in NR1 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881

:SBUS<n>:SENT:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:SENT:SOURce command specifies the input channel for SENT decode and triggering.

Query Syntax

```
:SBUS<n>:SENT:SOURce?
```

The :SBUS<n>:SENT:SOURce? query returns the specified SENT input source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:TOLerance

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TOLerance <percent>

<percent> ::= from 3-30, in NR1 format.
```

The :SBUS<n>:SENT:TOLerance command specifies the tolerance for determining whether the sync pulse is valid. Valid values range from 3% to 30%.

Query Syntax

```
:SBUS<n>:SENT:TOLerance?
```

The :SBUS<n>:SENT:TOLerance? query returns the tolerance setting.

Return Format

```
<percent><NL>

<percent> ::= from 3-30, in NR1 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger <mode>

<mode> ::= {SFCMessage | SSCMessage | FCData | SCMid | SCData
   | TOLerror | FCCerror | SCCerror | CRCerror | PPERror | SSPerror}
```

The :SBUS<n>:SENT:TRIGger command specifies the SENT trigger mode:

- SFCMessage – triggers on the start of any Fast Channel Message (after 56 Synchronization/Calibration ticks).
- SSCMessage – trigger on the start of any Slow Channel Message.
- FCData – triggers on a Fast Channel Message when the Status & Communication nibble and the data nibbles match the values entered using additional softkeys.
- SCMid – triggers when a Slow Channel Message ID matches the value entered using additional softkeys.
- SCData – triggers when a Slow Channel Message ID and Data field both match the values entered using additional softkeys.
- TOLerror – triggers when the sync pulse width varies from the nominal value by greater than the entered percentage.
- FCCerror – triggers on any Fast Channel Message CRC error.
- SCCerror – triggers on any Slow Channel Message CRC error.
- CRCerror – triggers on any CRC error, Fast or Slow.
- PPERror – triggers if a nibble is either too wide or too narrow (for example, data nibble < 12 (11.5) or > 27 (27.5) ticks wide). Sync, S&C, data, or checksum pulse periods are checked.
- SSPerror – triggers on a sync pulse whose width varies from the previous sync pulse's width by greater than 1/64 (1.5625%, as defined in the SENT specification).

Query Syntax

```
:SBUS<n>:SENT:TRIGger?
```

The :SBUS<n>:SENT:TRIGger? query returns the trigger mode setting.

Return Format

```
<mode><NL>

<mode> ::= {SFCM | SSCM | FCD | SCM | SCD | TOL | FCC | SCC | CRC
   | PPER | SSP}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872

:SBUS<n>:SENT:TRIGger:FAST:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger:FAST:DATA <string>

<string> ::= "nnnn..." where n ::= {0 | 1 | X}

<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:SENT:TRIGger:FAST:DATA command specifies the status and data nibbles that will be triggered on when the FCData trigger mode is chosen.

Query Syntax

```
:SBUS<n>:SENT:TRIGger:FAST:DATA?
```

The :SBUS<n>:SENT:TRIGger:FAST:DATA? query returns the fast channel data trigger setting.

Return Format

```
<string><NL>

<string> ::= "nnnn..." where n ::= {0 | 1 | X}

<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:TRIGger:SLOW:DATA

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:DATA <data>

<data> ::= when ILENgth = SHORt, from -1 (don't care) to 65535, in NR1 f
ormat.

<data> ::= when ILENgth = LONG, from -1 (don't care) to 4095, in NR1 for
mat.
```

The :SBUS<n>:SENT:TRIGger:SLOW:DATA command specifies the data to trigger on for the Slow Channel Message ID and Data trigger mode.

Query Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:DATA?
```

The :SBUS<n>:SENT:TRIGger:SLOW:DATA? query returns the data value setting for the slow channel ID and data trigger.

Return Format

```
<data><NL>

<data> ::= when ILENgth = SHORt, from -1 (don't care) to 65535, in NR1 f
ormat.

<data> ::= when ILENgth = LONG, from -1 (don't care) to 4095, in NR1 for
mat.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896

:SBUS<n>:SENT:TRIGger:SLOW:ID

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:ID <id>

<id> ::= when ILENgth = SHORt, from -1 (don't care) to 15, in NR1 format
.

<id> ::= when ILENgth = LONG, from -1 (don't care) to 255, in NR1 format
.
```

The :SBUS<n>:SENT:TRIGger:SLOW:ID command specifies the ID to trigger on for the "Slow Channel Message ID" and "Slow Channel Message ID & Data" trigger modes. The ID can be from -1 (don't care) to 255 (depending on the message length).

Query Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:ID?
```

The :SBUS<n>:SENT:TRIGger:SLOW:ID? query returns the slow channel ID setting.

Return Format

```
<id><NL>

<id> ::= when ILENgth = SHORt, from -1 (don't care) to 15, in NR1 format
.

<id> ::= when ILENgth = LONG, from -1 (don't care) to 255, in NR1 format
.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893

:SBUS<n>:SENT:TRIGger:SLOW:ILENgth

(see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:ILENgth <length>

<length> ::= {SHORt | LONG}
```

The :SBUS<n>:SENT:TRIGger:SLOW:ILENgth command specifies the ID and data lengths for the Slow Message Enhanced messages. Either "SHORt" for the 4-bit ID, 16-bit data format, or "LONG" for the 8-bit ID, 12-bit data format.

Query Syntax

```
:SBUS<n>:SENT:TRIGger:SLOW:ILENgth?
```

The :SBUS<n>:SENT:TRIGger:SLOW:ILENgth? query returns the ID and data length setting.

Return Format

```
<length><NL>

<length> ::= {SHOR | LONG}
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:TOLerance" on page 899

:SBUS<n>:SENT:TRIGger:TOLerance

N (see page 1354)

Command Syntax

```
:SBUS<n>:SENT:TRIGger:TOLerance <percent>

<percent> ::= from 1-28, in NR1 format.
```

The :SBUS<n>:SENT:TRIGger:TOLerance command specifies the tolerance variation that is considered a violation.

The trigger tolerance can be up to the :SBUS<n>:SENT:TOLerance setting minus two percent. For example, after sending ":SBUS1:SENT:TOLerance 20", the :SBUS1:SENT:TRIGger:TOLerance setting can be from 1 to 18.

Query Syntax

```
:SBUS<n>:SENT:TRIGger:TOLerance?
```

The :SBUS<n>:SENT:TRIGger:TOLerance? query returns tolerance variation percent setting.

Return Format

```
<percent><NL>

<percent> ::= from 1-28, in NR1 format.
```

See Also

- ":SBUS<n>:SENT:CLOCk" on page 868
- ":SBUS<n>:SENT:CRC" on page 869
- ":SBUS<n>:SENT:DISPlay" on page 870
- ":SBUS<n>:SENT:FORMat" on page 872
- ":SBUS<n>:SENT:IDLE" on page 874
- ":SBUS<n>:SENT:LENGth" on page 875
- ":SBUS<n>:SENT:PPULse" on page 876
- ":SBUS<n>:SENT:SIGNal<s>:DISPlay" on page 878
- ":SBUS<n>:SENT:SIGNal<s>:LENGth" on page 879
- ":SBUS<n>:SENT:SIGNal<s>:MULTiplier" on page 881
- ":SBUS<n>:SENT:SIGNal<s>:OFFSet" on page 883
- ":SBUS<n>:SENT:SIGNal<s>:ORDer" on page 885
- ":SBUS<n>:SENT:SIGNal<s>:STARt" on page 887
- ":SBUS<n>:SENT:SOURce" on page 889
- ":SBUS<n>:SENT:TOLerance" on page 890
- ":SBUS<n>:SENT:TRIGger" on page 891
- ":SBUS<n>:SENT:TRIGger:FAST:DATA" on page 893
- ":SBUS<n>:SENT:TRIGger:SLOW:DATA" on page 894
- ":SBUS<n>:SENT:TRIGger:SLOW:ID" on page 896
- ":SBUS<n>:SENT:TRIGger:SLOW:ILENgth" on page 898

:SBUS<n>:UART Commands

NOTE These commands are only valid when the UART/RS-232 triggering and serial decode option (COMP license) has been installed.

Table 112 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 902)	:SBUS<n>:UART:BASE? (see page 902)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 903)	:SBUS<n>:UART:BAUDrate? (see page 903)	<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
:SBUS<n>:UART:BITorder <bitorder> (see page 904)	:SBUS<n>:UART:BITorder? (see page 904)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 905)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 906)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 907)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 908)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 909)	:SBUS<n>:UART:FRAMing? (see page 909)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:SBUS<n>:UART:PARity <parity> (see page 910)	:SBUS<n>:UART:PARity? (see page 910)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 911)	:SBUS<n>:UART:POLarity? (see page 911)	<polarity> ::= {HIGH | LOW}

Table 112 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce:RX <source> (see page 912)	:SBUS<n>:UART:SOURce:RX? (see page 912)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:SOURce:TX <source> (see page 913)	:SBUS<n>:UART:SOURce:TX? (see page 913)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 914)	:SBUS<n>:UART:TRIGger:BASE? (see page 914)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 915)	:SBUS<n>:UART:TRIGger:BURSt? (see page 915)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger:DATA <value> (see page 916)	:SBUS<n>:UART:TRIGger:DATA? (see page 916)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SBUS<n>:UART:TRIGger:IDLE <time_value> (see page 917)	:SBUS<n>:UART:TRIGger:IDLE? (see page 917)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger:QUALifier <value> (see page 918)	:SBUS<n>:UART:TRIGger:QUALifier? (see page 918)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:SBUS<n>:UART:TRIGger:TYPE <value> (see page 919)	:SBUS<n>:UART:TRIGger:TYPE? (see page 919)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 920)	:SBUS<n>:UART:WIDTh? (see page 920)	<width> ::= {5 | 6 | 7 | 8 | 9}

:SBUS<n>:UART:BASE

(see page 1354)

Command Syntax

```
:SBUS<n>:UART:BASE <base>

<base> ::= {ASCii | BINary | HEX}
```

The :SBUS<n>:UART:BASE command determines the base to use for the UART decode and Lister display.

Query Syntax

```
:SBUS<n>:UART:BASE?
```

The :SBUS<n>:UART:BASE? query returns the current UART decode and Lister base setting.

Return Format

```
<base><NL>

<base> ::= {ASCii | BINary | HEX}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:BAUDrate

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:BAUDrate <baudrate>

<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
```

The :SBUS<n>:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode. The baud rate can be set in the range from from 100 b/s to 8 Mb/s or to the specific values of 10 Mb/s or 12 Mb/s.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:SBUS<n>:UART:BAUDrate?
```

The :SBUS<n>:UART:BAUDrate? query returns the current UART baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer from 100 to 8000000, 10000000, or 12000000
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:BITorder

(see page 1354)

Command Syntax

```
:SBUS<n>:UART:BITorder <bitorder>

<bitorder> ::= {LSBFirst | MSBFirst}
```

The :SBUS<n>:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

Query Syntax

```
:SBUS<n>:UART:BITorder?
```

The :SBUS<n>:UART:BITorder? query returns the current UART bit order setting.

Return Format

```
<bitorder><NL>

<bitorder> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919
- ":SBUS<n>:UART:SOURce:RX" on page 912
- ":SBUS<n>:UART:SOURce:TX" on page 913

:SBUS<n>:UART:COUNt:ERRor

(see page 1354)

Query Syntax `:SBUS<n>:UART:COUNt:ERRor?`

Returns the UART error frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 906
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:COUNt:RESet

 (see page 1354)

Command Syntax `:SBUS<n>:UART:COUNt:RESet`

Resets the UART frame counters.

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:UART:COUNt:ERRor" on page 905
- ":SBUS<n>:UART:COUNt:RXFRames" on page 907
- ":SBUS<n>:UART:COUNt:TXFRames" on page 908
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:COUNt:RXFRames

(see page 1354)

Query Syntax `:SBUS<n>:UART:COUNt:RXFRames?`

Returns the UART Rx frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 906
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:COUNt:TXFRames

(see page 1354)

Query Syntax `:SBUS<n>:UART:COUNt:TXFRames?`

Returns the UART Tx frame count.

Return Format `<frame_count><NL>`

`<frame_count> ::= integer in NR1 format`

Errors

- "-241, Hardware missing" on page 1295

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 906
- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:FRAMing

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:FRAMing <value>

<value> ::= {OFF | <decimal> | <nondecimal>}

<decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
```

The :SBUS<n>:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

Query Syntax

```
:SBUS<n>:UART:FRAMing?
```

The :SBUS<n>:UART:FRAMing? query returns the current UART decode base setting.

Return Format

```
<value><NL>

<value> ::= {OFF | <decimal>}

<decimal> ::= 8-bit integer in decimal from 0-255
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:UART Commands" on page 900

:SBUS<n>:UART:PARity

(see page 1354)

Command Syntax

```
:SBUS<n>:UART:PARity <parity>

<parity> ::= {EVEN | ODD | NONE}
```

The :SBUS<n>:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:PARity?
```

The :SBUS<n>:UART:PARity? query returns the current UART parity setting.

Return Format

```
<parity><NL>

<parity> ::= {EVEN | ODD | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:POLarity

(see page 1354)

Command Syntax

```
:SBUS<n>:UART:POLarity <polarity>

<polarity> ::= {HIGH | LOW}
```

The :SBUS<n>:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:POLarity?
```

The :SBUS<n>:UART:POLarity? query returns the current UART polarity setting.

Return Format

```
<polarity><NL>

<polarity> ::= {HIGH | LOW}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:SOURce:RX

(see page 1354)

Command Syntax

```
:SBUS<n>:UART:SOURce:RX <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:RX?
```

The :SBUS<n>:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919
- ":SBUS<n>:UART:BITorder" on page 904

:SBUS<n>:UART:SOURce:TX

 (see page 1354)

Command Syntax

```
:SBUS<n>:UART:SOURce:TX <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:TX?
```

The :SBUS<n>:UART:SOURce:TX? query returns the current source for the UART Tx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919
- ":SBUS<n>:UART:BITorder" on page 904

:SBUS<n>:UART:TRIGger:BASE

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:BASE <base>

<base> ::= {ASCii | HEX}
```

The :SBUS<n>:UART:TRIGger:BASE command sets the front panel UART/RS232 trigger setup data selection option:

- ASCii – front panel data selection is from ASCII values.
- HEX – front panel data selection is from hexadecimal values.

The :SBUS<n>:UART:TRIGger:BASE setting does not affect the :SBUS<n>:UART:TRIGger:DATA command which can always set data values using ASCII or hexadecimal values.

NOTE **The :SBUS<n>:UART:TRIGger:BASE command is independent of the :SBUS<n>:UART:BASE command which affects decode and Lister only.**

Query Syntax

```
:SBUS<n>:UART:TRIGger:BASE?
```

The :SBUS<n>:UART:TRIGger:BASE? query returns the current UART base setting.

Return Format

```
<base><NL>

<base> ::= {ASC | HEX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:DATA" on page 916

:SBUS<n>:UART:TRIGger:BURSt

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:BURSt <value>

<value> ::= {OFF | 1 to 4096 in NR1 format}
```

The :SBUS<n>:UART:TRIGger:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:BURSt?
```

The :SBUS<n>:UART:TRIGger:BURSt? query returns the current UART trigger burst value.

Return Format

```
<value><NL>

<value> ::= {OFF | 1 to 4096 in NR1 format}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:IDLE" on page 917
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:TRIGger:DATA

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:DATA <value>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
            <hexadecimal>, <binary>, or <quoted_string> format

<hexadecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<binary> ::= #Bnn...n where n ::= {0 | 1} for binary

<quoted_string> ::= any of the 128 valid 7-bit ASCII characters
                    (or standard abbreviations)
```

The :SBUS<n>:UART:TRIGger:DATA command selects the data byte value (0x00 to 0xFF) for the trigger QUALifier when in UART mode. The data value is used when one of the RD or TD trigger types is selected.

When entering an ASCII character via the quoted string, it must be one of the 128 valid characters (case-sensitive): "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL", "BS", "HT", "LF", "VT", "FF", "CR", "SO","SI", "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN", "ETB", "CAN", "EM", "SUB", "ESC", "FS","GS", "RS", "US", "SP", "!", "\"", "#", "$", "%","&", "\'", "(", ")", "*", "+", ",", "-", ".", "/","0", "1", "2", "3", "4", "5", "6", "7", "8", "9",":", ";", "<", "=", ">", "?", "@", "A", "B", "C","D", "E", "F", "G", "H", "I", "J", "K", "L", "M","N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z","[", "\\", "]", "^", "_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "|", "}", "~", or "DEL".

Query Syntax

```
:SBUS<n>:UART:TRIGger:DATA?
```

The :SBUS<n>:UART:TRIGger:DATA? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= 8-bit integer in decimal from 0-255
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:BASE" on page 914
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:TRIGger:IDLE

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:IDLE <time_value>

<time_value> ::= time from 1 us to 10 s in NR3 format
```

The :SBUS<n>:UART:TRIGger:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:IDLE?
```

The :SBUS<n>:UART:TRIGger:IDLE? query returns the current UART trigger idle period time.

Return Format

```
<time_value><NL>

<time_value> ::= time from 1 us to 10 s in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:BURSt" on page 915
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:TRIGger:QUALifier

 (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
```

The :SBUS<n>:UART:TRIGger:QUALifier command selects the data qualifier when :TYPE is set to RDATa, RD1, RD0, RDX, TDATa, TD1, TD0, or TDX for the trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:TRIGger:QUALifier?
```

The :SBUS<n>:UART:TRIGger:QUALifier? query returns the current UART trigger qualifier.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:UART:TRIGger:TYPE

N (see page 1354)

Command Syntax

```
:SBUS<n>:UART:TRIGger:TYPE <value>

<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror
             | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
```

The :SBUS<n>:UART:TRIGger:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :SBUS<n>:UART:TRIGger:DATA and :SBUS<n>:UART:TRIGger:QUALifier commands are used to specify the data value and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and alert bit values) are only valid when a 9-bit width has been selected.

Query Syntax

```
:SBUS<n>:UART:TRIGger:TYPE?
```

The :SBUS<n>:UART:TRIGger:TYPE? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | PAR | TSTA |
             TSTO | TDAT | TD1 | TD0 | TDX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:DATA" on page 916
- ":SBUS<n>:UART:TRIGger:QUALifier" on page 918
- ":SBUS<n>:UART:WIDTh" on page 920

:SBUS<n>:UART:WIDTh

 (see page 1354)

Command Syntax

```
:SBUS<n>:UART:WIDTh <width>

<width> ::= {5 | 6 | 7 | 8 | 9}
```

The :SBUS<n>:UART:WIDTh command determines the number of bits (5-9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:WIDTh?
```

The :SBUS<n>:UART:WIDTh? query returns the current UART width setting.

Return Format

```
<width><NL>

<width> ::= {5 | 6 | 7 | 8 | 9}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:UART:TRIGger:TYPE" on page 919

:SBUS<n>:USBPd Commands

NOTE These commands are valid when the USB PD (Power Delivery) serial decode and triggering option has been licensed.

Table 113 :SBUS<n>:USBPd Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:USBPd:SOURce <source> (see page 922)	:SBUS<n>:USBPd:SOURce? (see page 922)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:USBPd:TRIGger <mode> (see page 923)	:SBUS<n>:USBPd:TRIGger? (see page 923)	<mode> ::= {PSTart | EOP | SOP | SPRime | SDPRime | SPDebug | SDPDebug | HRST | CRST | CRCerror | PERRor | HEADer}
:SBUS<n>:USBPd:TRIGger:HEADer <type> (see page 924)	:SBUS<n>:USBPd:TRIGger:HEADer? (see page 924)	<type> ::= {CMESsage | DMESsage | EMESsage | VALue}
:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage <type> (see page 926)	:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage? (see page 926)	<type> ::= {GOODcrc | GOTOmin | ACCept | REJect | PING | PSRDy | GSRCap | GSNCap | DRSWap | PRSWap | VCSWap | WAIT | SRST | GSCX | GSTatus | FRSWap | GPSTatus | GCCodes}
:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage <type> (see page 928)	:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage? (see page 928)	<type> ::= {SRCap | REQuest | BIST | SNCap | BSTatus | ALERt | GCINfo | VDEFined}
:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage <type> (see page 929)	:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage? (see page 929)	<type> ::= {SCX | STATus | GBCap | GBSTatus | BCAP | GMINfo | MINFo | SREQuest | SRESponse | FREQuest | FRESponse | PSTatus | CINFo | CCODes}
:SBUS<n>:USBPd:TRIGger:HEADer:VALue <string> (see page 931)	:SBUS<n>:USBPd:TRIGger:HEADer:VALue? (see page 931)	<string> ::= "nn...n" where n ::= {0 | 1 | X} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier <type> (see page 932)	:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier? (see page 932)	<type> ::= {NONE | SOP | SPRime | SDPRime}

:SBUS<n>:USBPd:SOURce

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SBUS<n>:USBPd:SOURce command selects the USB PD waveform source. You can use analog channels.

Query Syntax

```
:SBUS<n>:USBPd:SOURce?
```

The :SBUS<n>:USBPd:SOURce? query returns the selected analog input channel.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger <mode>

<mode> ::= {PSTart | EOP | SOP | SPRime | SDPRime | SPDebug | SDPDebug
   | HRST | CRST | CRCerror | PERRor | HEADer}
```

The :SBUS<n>:USBPd:TRIGger command selects the USB PD trigger mode:

- PSTart – **Preamble Start**, triggers at the start of a preamble, which starts with 0.
- EOP – **EOP**, triggers at the end of packet.
- SOP – **SOP**, triggers on ordered set Sync-1, Sync-1, Sync-1, Sync-2.
- SPRime – **SOP'**, triggers on ordered set Sync-1, Sync-1, Sync-3, Sync-3.
- SDPRime – **SOP"**, triggers on ordered set Sync-1, Sync-3, Sync-1, Sync-3.
- SPDebug – **SOP' Debug**, triggers on ordered set Sync-1, RST-2, RST-2, Sync-3.
- SDPDebug – **SOP" Debug**, triggers on ordered set Sync-1, RST-2, Sync-3, Sync-2.
- HRST – **Hard Reset**, triggers on ordered set RST-1, RST-1, RST-1, RST-2.
- CRST – **Cable Reset**, triggers on ordered set RST-1, Sync-1, RST-1, Sync-3.
- CRCerror – **CRC Error**, triggers when an error is detected on a 32-bit CRC.
- PERRor – **Preamble Error**, triggers when an error is detected on a 64-bit sequence of alternating 0 and 1.
- HEADer – **Header Content**, triggers on a user-defined 16-bit value.

  In this mode, use the :SBUS<n>:USBPd:TRIGger:HEADer command to select the header type.

Query Syntax

```
:SBUS<n>:USBPd:TRIGger?
```

The :SBUS<n>:USBPd:TRIGger? query returns the selected USB PD trigger mode.

Return Format

```
<mode><NL>

<mode> ::= {PST | EOP | SOP | SPR | SDPR | SPD | SDPD | HRST | CRST
   | CRC | PERR | HEAD}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer <type>

<type> ::= {CMESsage | DMESsage | EMESsage | VALue}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command), the :SBUS<n>:USBPd:TRIGger:HEADer command selects the header type:

- CMESsage — **Control Message**, triggers on control message types (0 data object).

  When the CMESsage header type is selected, use the :SBUS<n>:USBPd:TRIGger:HEADer:CMESsage command to select the control message type.

- DMESsage — **Data Message**, triggers on data message types (1 or more data objects).

  When the DMESsage header type is selected, use the :SBUS<n>:USBPd:TRIGger:HEADer:DMESsage command to select the data message type.

- EMESsage — **Extended Message**, triggers on extended message types (bit 15 is set).

  When the EMESsage header type is selected, use the :SBUS<n>:USBPd:TRIGger:HEADer:EMESsage command to select the extended message type.

- VALue — **Value**, triggers on a user-defined header value.

  When the CMESsage header type is selected, use the :SBUS<n>:USBPd:TRIGger:HEADer:VALue command to specify the user-defined header value.

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer?
```

The :SBUS<n>:USBPd:TRIGger:HEADer? query returns the selected header type.

Return Format

```
<type><NL>

<type> ::= {CMES | DMES | EMES | VAL}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931

- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage <type>

<type> ::= {GOODcrc | GOTOmin | ACCept | REJect | PING | PSRDy
   | GSRCap | GSNCap | DRSWap | PRSWap | VCSWap | WAIT | SRST | GSCX
   | GSTatus | FRSWap | GPSTatus | GCCodes}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command) and the **Control Message** header type is selected (with the ":SBUS<n>:USBPd:TRIGger:HEADer CMESsage" command), the :SBUS<n>:USBPd:TRIGger:HEADer:CMESsage command selects the control message type:

- GOODcrc – GoodCRC
- GOTOmin – GotoMin
- ACCept – Accept
- REJect – Reject
- PING – Ping
- PSRDy – PS_RDY
- GSRCap – Get_Source_Cap
- GSNCap – Get_Sink_Cap
- DRSWap – DR_Swap
- PRSWap – PR_Swap
- VCSWap – VCONN_Swap
- WAIT – Wait
- SRST – Soft_Reset
- GSCX – Get_Source_Cap_Extended
- GSTatus – Get_Status
- FRSWap – FR_Swap
- GPSTatus – Get_PPS_Status
- GCCodes – Get_Country_Codes

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:CMESsage?
```

The :SBUS<n>:USBPd:TRIGger:HEADer:CMESsage? query returns the selected control message type.

Return Format

```
<type><NL>

<type> ::= {GOOD | GOTO | ACC | REJ | PING | PSRD | GSRC | GSNC | DRSW
   | PRSW | VCSW | WAIT | SRST | GSCX | GST | FRSW | GPST | GCC}
```

See Also
- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage <type>

<type> ::= {SRCap | REQuest | BIST | SNCap | BSTatus | ALERt | GCINfo
   | VDEFined}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command) and the **Data Message** header type is selected (with the ":SBUS<n>:USBPd:TRIGger:HEADer DMESsage" command), the :SBUS<n>:USBPd:TRIGger:HEADer:DMESsage command selects the data message type:

- SRCap – Source_Capabilities
- REQuest – Request
- BIST– BIST
- SNCap – Sink_Capabilities
- BSTatus – Battery_Status
- ALERt – Alert
- GCINfo – Get_Country_Info
- VDEFined – Vendor_Defined

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:DMESsage?
```

The :SBUS<n>:USBPd:TRIGger:HEADer:DMESsage? query returns the selected data message type.

Return Format

```
<type><NL>

<type> ::= {SRC | REQ | BIST | SNC | BST | ALER | GCIN | VDEF}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage <type>

<type> ::= {SCX | STATus | GBCap | GBSTatus | BCAP | GMINfo | MINFo
   | SREQuest | SRESponse | FREQuest | FRESponse | PSTatus | CINFo
   | CCODes}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command) and the **Extended Message** header type is selected (with the ":SBUS<n>:USBPd:TRIGger:HEADer EMESsage" command), the :SBUS<n>:USBPd:TRIGger:HEADer:EMESsage command selects the extended message type:

- SCX – Source_Capabilities_Extended
- STATus – Status
- GBCap – Get_Battery_Cap
- GBSTatus – Get_Battery_Status
- BCAP – Battery_Capabilities
- GMINfo – Get_Manufacturer_Info
- MINFo – Manufacturer_Info
- SREQuest – Security_Request
- SRESponse – Security_Response
- FREQuest – Firmware_Update_Request
- FRESponse – Firmware_Update_Response
- PSTatus – PPS_Status
- CINFo – Country_Info
- CCODes – Country_Codes

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:EMESsage?
```

The :SBUS<n>:USBPd:TRIGger:HEADer:EMESsage? query returns the selected extended message type

Return Format

```
<type><NL>

<type> ::= {SCX | STATus | GBC | GBST | BCAP | GMIN | MINF | SREQ
   | SRES | FREQ | FRES | PSTatus | CINF | CCOD}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926

- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer:VALue

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:VALue <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command) and the user-defined **Value** type is selected (with the ":SBUS<n>:USBPd:TRIGger:HEADer VALue" command), the :SBUS<n>:USBPd:TRIGger:HEADer:VALue command specifies the user-defined header value.

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:VALue?
```

The :SBUS<n>:USBPd:TRIGger:HEADer:VALue? query returns the specified user-defined header value.

Returned data values are always quoted binary format strings.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1 | X}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:QUALifier" on page 932

:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier

N (see page 1354)

Command Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier <type>

<type> ::= {NONE | SOP | SPRime | SDPRime}
```

When the **Header Content** trigger mode is selected (with the ":SBUS<n>:USBPd:TRIGger HEADer" command), the :SBUS<n>:USBPd:TRIGger:HEADer:QUALifier command selects an additional qualifier for the Header Content trigger:

- NONE — **None**. There is no additional qualifier for the trigger.
- SOP — **SOP**. The trigger occurs on Sync-1, Sync-1, Sync-1, Sync-2 ordered sets only.
- SPRime — **SOP'**. The trigger occurs on Sync-1, Sync-1, Sync-3, Sync-3 ordered sets only.
- SDPRime — **SOP"**. The trigger occurs on Sync-1, Sync-3, Sync-1, Sync-3 ordered sets only.

Query Syntax

```
:SBUS<n>:USBPd:TRIGger:HEADer:QUALifier?
```

The :SBUS<n>:USBPd:TRIGger:HEADer:QUALifier? query returns the selected header content trigger qualifier.

Return Format

```
<type><NL>

<type> ::= {NONE | SOP | SPRime | SDPRime}
```

See Also

- ":SBUS<n>:USBPd:SOURce" on page 922
- ":SBUS<n>:USBPd:TRIGger" on page 923
- ":SBUS<n>:USBPd:TRIGger:HEADer" on page 924
- ":SBUS<n>:USBPd:TRIGger:HEADer:CMESsage" on page 926
- ":SBUS<n>:USBPd:TRIGger:HEADer:DMESsage" on page 928
- ":SBUS<n>:USBPd:TRIGger:HEADer:EMESsage" on page 929
- ":SBUS<n>:USBPd:TRIGger:HEADer:VALue" on page 931

28 :SEARch Commands

Control the event search modes and parameters for each search type. See:

General :SEARch Commands

Table 114 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 935)	<count> ::= an integer count value
:SEARch:EVENt <event_number> (see page 936)	:SEARch:EVENt? (see page 936)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 937)	:SEARch:MODE? (see page 937)	<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2} | PEAK}
:SEARch:STATe <value> (see page 938)	:SEARch:STATe? (see page 938)	<value> ::= {{0 | OFF} | {1 | ON}}

:SEARch:COUNt

 (see page 1354)

Query Syntax `:SEARch:COUNt?`

The :SEARch:COUNt? query returns the number of search events found.

Return Format `<count><NL>`

`<count> ::= an integer count value`

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:EVENt" on page 936
- ":SEARch:STATe" on page 938
- ":SEARch:MODE" on page 937

:SEARch:EVENt

(see page 1354)

Command Syntax

```
:SEARch:EVENt <event_number>

<event_number> ::= the integer number of a found search event
```

The :SEARch:EVENt command navigates to a found search event.

If the :SEARch:STATe is ON, the horizontal position is changed so that the specified event is located at the time reference.

Query Syntax

```
:SEARch:EVENt?
```

The :SEARch:EVENt? query returns the currently selected event number.

Return Format

```
<event_number><NL>

<event_number> ::= the integer number of a found search event
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:COUNt" on page 935
- ":SEARch:STATe" on page 938
- ":SEARch:MODE" on page 937

:SEARch:MODE

N (see page 1354)

Command Syntax

```
:SEARch:MODE <value>

<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2} | PEAK}
```

The :SEARch:MODE command selects the search mode.

The command is only valid when the :SEARch:STATe is ON.

Query Syntax

```
:SEARch:MODE?
```

The :SEARch:MODE? query returns the currently selected mode or OFF if the :SEARch:STATe is OFF.

Return Format

```
<value><NL>

<value> ::= {EDGE | GLIT | RUNT | TRAN | SER{1 | 2} | PEAK | OFF}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:STATe" on page 938
- ":SEARch:COUNt" on page 935
- ":SEARch:EVENt" on page 936

:SEARch:STATe

(see page 1354)

Command Syntax

```
:SEARch:STATe <value>
<value> ::= {{0 | OFF} | {1 | ON}}
```

The :SEARch:STATe command enables or disables the search feature.

Query Syntax

```
:SEARch:STATe?
```

The :SEARch:STATe? query returns returns the current setting.

Return Format

```
<value><NL>
<value> ::= {0 | 1}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:COUNt" on page 935
- ":SEARch:EVENt" on page 936

:SEARch:EDGE Commands

Table 115 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 940)	:SEARch:EDGE:SLOPe? (see page 940)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 941)	:SEARch:EDGE:SOURce? (see page 941)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:EDGE:SLOPe

N (see page 1354)

Command Syntax

```
:SEARch:EDGE:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer}
```

The :SEARch:EDGE:SLOPe command specifies the slope of the edge for the search.

Query Syntax

```
:SEARch:EDGE:SLOPe?
```

The :SEARch:EDGE:SLOPe? query returns the current slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS | EITH}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933

:SEARch:EDGE:SOURce

 (see page 1354)

Command Syntax

```
:SEARch:EDGE:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:EDGE:SOURce command selects the channel on which to search for edges.

Query Syntax

```
:SEARch:EDGE:SOURce?
```

The :SEARch:EDGE:SOURce? query returns the current source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933

:SEARch:GLITch Commands

Table 116 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 943)	:SEARch:GLITch:GREaterthan? (see page 943)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 944)	:SEARch:GLITch:LESSthan? (see page 944)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 945)	:SEARch:GLITch:POLarity? (see page 945)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 946)	:SEARch:GLITch:QUALifier? (see page 946)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 947)	:SEARch:GLITch:RANGe? (see page 947)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:SOURce <source> (see page 948)	:SEARch:GLITch:SOURce? (see page 948)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:GLITch:GREaterthan

N (see page 1354)

Command Syntax

```
:SEARch:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= floating-point number in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :SEARch:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :SEARch:GLITch:SOURce.

Query Syntax

```
:SEARch:GLITch:GREaterthan?
```

The :SEARch:GLITch:GREaterthan? query returns the minimum pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<greater_than_time><NL>

<greater_than_time> ::= floating-point number in NR3 format.
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:GLITch:SOURce" on page 948
- ":SEARch:GLITch:QUALifier" on page 946
- ":SEARch:MODE" on page 937

:SEARch:GLITch:LESSthan

N (see page 1354)

Command Syntax

```
:SEARch:GLITch:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= floating-point number in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :SEARch:GLITch:LESSthan command sets the maximum pulse width duration for the selected :SEARch:GLITch:SOURce.

Query Syntax

```
:SEARch:GLITch:LESSthan?
```

The :SEARch:GLITch:LESSthan? query returns the pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<less_than_time><NL>

<less_than_time> ::= floating-point number in NR3 format.
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:GLITch:SOURce" on page 948
- ":SEARch:GLITch:QUALifier" on page 946
- ":SEARch:MODE" on page 937

:SEARch:GLITch:POLarity

(see page 1354)

Command Syntax

```
:SEARch:GLITch:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :SEARch:GLITch:POLarity command sets the polarity for the glitch (pulse width) search.

Query Syntax

```
:SEARch:GLITch:POLarity?
```

The :SEARch:GLITch:POLarity? query returns the current polarity setting for the glitch (pulse width) search.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:GLITch:SOURce" on page 948

:SEARch:GLITch:QUALifier

(see page 1354)

Command Syntax

```
:SEARch:GLITch:QUALifier <operator>

<operator> ::= {GREaterthan | LESSthan | RANGe}
```

This command sets the mode of operation of the glitch (pulse width) search. The oscilloscope can search for a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax

```
:SEARch:GLITch:QUALifier?
```

The :SEARch:GLITch:QUALifier? query returns the glitch (pulse width) qualifier.

Return Format

```
<operator><NL>

<operator> ::= {GRE | LESS | RANG}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:GLITch:SOURce" on page 948
- ":SEARch:MODE" on page 937

:SEARch:GLITch:RANGe

N (see page 1354)

Command Syntax

```
:SEARch:GLITch:RANGe <less_than_time>[suffix],
                     <greater_than_time>[suffix]

<less_than_time> ::= (15 ns - 10 seconds) in NR3 format

<greater_than_time> ::= (10 ns - 9.99 seconds) in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :SEARch:GLITch:RANGe command sets the pulse width duration for the selected :SEARch:GLITch:SOURce. You can enter the parameters in any order — the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:SEARch:GLITch:RANGe?
```

The :SEARch:GLITch:RANGe? query returns the pulse width duration time for :SEARch:GLITch:SOURce.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:GLITch:SOURce" on page 948
- ":SEARch:GLITch:QUALifier" on page 946
- ":SEARch:MODE" on page 937

:SEARch:GLITch:SOURce

(see page 1354)

Command Syntax

```
:SEARch:GLITch:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:GLITch:SOURce command selects the channel on which to search for glitches (pulse widths).

Query Syntax

```
:SEARch:GLITch:SOURce?
```

The :SEARch:GLITch:SOURce? query returns the current pulse width source.

If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:GLITch:POLarity" on page 945
- ":SEARch:GLITch:QUALifier" on page 946
- ":SEARch:GLITch:RANGe" on page 947

:SEARch:PEAK Commands

Table 117 :SEARch:PEAK Commands Summary

Command	Query	Options and Query Returns
:SEARch:PEAK:EXCursion <delta_level> (see page 950)	:SEARch:PEAK:EXCursion? (see page 950)	<delta_level> ::= required change in level to be recognized as a peak, in NR3 format.
:SEARch:PEAK:NPEaks <number> (see page 951)	:SEARch:PEAK:NPEaks? (see page 951)	<number> ::= max number of peaks to find, 1-11 in NR1 format.
:SEARch:PEAK:SOURce <source> (see page 952)	:SEARch:PEAK:SOURce? (see page 952)	<source> ::= {FUNCtion<m> | MATH<m> | FFT} (must be an FFT waveform) <m> ::= 1 to (# math functions) in NR1 format
:SEARch:PEAK:THReshold <level> (see page 953)	:SEARch:PEAK:THReshold? (see page 953)	<level> ::= necessary level to be considered a peak, in NR3 format.

:SEARch:PEAK:EXCursion

 (see page 1354)

Command Syntax

```
:SEARch:PEAK:EXCursion <delta_level>

<delta_level> ::= required change in level to be recognized as a peak,
   in NR3 format.
```

The :SEARch:PEAK:EXCursion command specifies the change in level that must occur (in other words, hysteresis) to be recognized as a peak.

The threshold level units are specified by the :FFT:VTYPe or :FUNCtion<m>[:FFT]:VTYPe command.

Query Syntax

```
:SEARch:PEAK:EXCursion?
```

The :SEARch:PEAK:EXCursion? query returns the specified excursion delta level value.

Return Format

```
<delta_level><NL>

<delta_level> ::= in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:VTYPe" on page 382
- ":SEARch:PEAK:NPEaks" on page 951
- ":SEARch:PEAK:SOURce" on page 952
- ":SEARch:PEAK:THReshold" on page 953

:SEARch:PEAK:NPEaks

(see page 1354)

Command Syntax

```
:SEARch:PEAK:NPEaks <number>

<number> ::= max number of peaks to find, 1-11 in NR1 format.
```

The :SEARch:PEAK:NPEaks command specifies the maximum number of FFT peaks to find. This number can be from 1 to 11.

Query Syntax

```
:SEARch:PEAK:NPEaks?
```

The :SEARch:PEAK:NPEaks? query returns the specified maximum number of FFT peaks to find.

Return Format

```
<number><NL>

<number> ::= in NR1 format.
```

See Also

- ":SEARch:PEAK:EXCursion" on page 950
- ":SEARch:PEAK:SOURce" on page 952
- ":SEARch:PEAK:THReshold" on page 953

:SEARch:PEAK:SOURce

(see page 1354)

Command Syntax

```
:SEARch:PEAK:SOURce <source>

<source> ::= {FUNCtion<m> | MATH<m> | FFT} (source must be
   an FFT waveform)

<m> ::= 1 to (# math functions) in NR1 format
```

The :SEARch:PEAK:SOURce command selects the FFT math function waveform to search.

Query Syntax

```
:SEARch:PEAK:SOURce?
```

The :SEARch:PEAK:SOURce? query returns the FFT math function waveform that is being searched.

Return Format

```
<source><NL>

<source> ::= {FUNC<m> | FFT} (must be FFT)

<m> ::= 1 to (# math functions) in NR1 format
```

See Also

- ":SEARch:PEAK:EXCursion" on page 950
- ":SEARch:PEAK:NPEaks" on page 951
- ":SEARch:PEAK:THReshold" on page 953

:SEARch:PEAK:THReshold

(see page 1354)

Command Syntax

```
:SEARch:PEAK:THReshold <level>

<level> ::= necessary level to be considered a peak, in NR3 format.
```

The :SEARch:PEAK:THReshold command specifies the threshold level necessary to be considered a peak.

The threshold level units are specified by the :FFT:VTYPe or :FUNCtion<m>[:FFT]:VTYPe command.

Query Syntax

```
:SEARch:PEAK:THReshold?
```

The :SEARch:PEAK:THReshold? query returns the specified threshold level for FFT peak search.

Return Format

```
<level><NL>

<level> ::= in NR3 format.
```

See Also

- ":FUNCtion<m>[:FFT]:VTYPe" on page 382
- ":SEARch:PEAK:EXCursion" on page 950
- ":SEARch:PEAK:NPEaks" on page 951
- ":SEARch:PEAK:SOURce" on page 952

:SEARch:RUNT Commands

Table 118 :SEARch:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARch:RUNT:POLarity <polarity> (see page 955)	:SEARch:RUNT:POLarity? (see page 955)	<polarity> ::= {POSitive | NEGative | EITHer}
:SEARch:RUNT:QUALifier <qualifier> (see page 956)	:SEARch:RUNT:QUALifier? (see page 956)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:SEARch:RUNT:SOURce <source> (see page 957)	:SEARch:RUNT:SOURce? (see page 957)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:RUNT:TIME <time>[suffix] (see page 958)	:SEARch:RUNT:TIME? (see page 958)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:SEARch:RUNT:POLarity

N (see page 1354)

Command Syntax

```
:SEARch:RUNT:POLarity <slope>
<polarity> ::= {POSitive | NEGative | EITHer}
```

The :SEARch:RUNT:POLarity command sets the polarity for the runt search.

Query Syntax

```
:SEARch:RUNT:POLarity?
```

The :SEARch:RUNT:POLarity? query returns the currently set runt polarity.

Return Format

```
<slope><NL>
<polarity> ::= {POS | NEG | EITH}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:RUNT:SOURce" on page 957

:SEARch:RUNT:QUALifier

(see page 1354)

Command Syntax

```
:SEARch:RUNT:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | NONE}
```

The :SEARch:RUNT:QUALifier command specifies whether to search for a runt that is greater than a time value, less than a time value, or any time value.

Query Syntax

```
:SEARch:RUNT:QUALifier?
```

The :SEARch:RUNT:QUALifier? query returns the current runt search qualifier.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS NONE}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937

:SEARch:RUNT:SOURce

(see page 1354)

Command Syntax

```
:SEARch:RUNT:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:RUNT:SOURce command selects the channel on which to search for the runt pulse.

Query Syntax

```
:SEARch:RUNT:SOURce?
```

The :SEARch:RUNT:SOURce? query returns the current runt search source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:RUNT:POLarity" on page 955

:SEARch:RUNT:TIME

 (see page 1354)

Command Syntax

```
:SEARch:RUNT:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

When searching for runt pulses whose widths are greater than or less than a time (see :SEARch:RUNT:QUALifier), the :SEARch:RUNT:TIME command specifies the time value.

Query Syntax

```
:SEARch:RUNT:TIME?
```

The :SEARch:RUNT:TIME? query returns the currently specified runt time value.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:RUNT:QUALifier" on page 956

:SEARch:TRANsition Commands

Table 119 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 960)	:SEARch:TRANsition:QUALifier? (see page 960)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 961)	:SEARch:TRANsition:SLOPe? (see page 961)	<slope> ::= {NEGative | POSitive}
:SEARch:TRANsition:SOURce <source> (see page 962)	:SEARch:TRANsition:SOURce? (see page 962)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 963)	:SEARch:TRANsition:TIME? (see page 963)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:SEARch:TRANsition:QUALifier

N (see page 1354)

Command Syntax

```
:SEARch:TRANsition:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan}
```

The :SEARch:TRANsition:QUALifier command specifies whether to search for edge transitions greater than or less than a time.

Query Syntax

```
:SEARch:TRANsition:QUALifier?
```

The :SEARch:TRANsition:QUALifier? query returns the current transition search qualifier.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:TRANsition:TIME" on page 963

:SEARch:TRANsition:SLOPe

(see page 1354)

Command Syntax

```
:SEARch:TRANsition:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :SEARch:TRANsition:SLOPe command selects whether to search for rising edge (POSitive slope) transitions or falling edge (NEGative slope) transitions.

Query Syntax

```
:SEARch:TRANsition:SLOPe?
```

The :SEARch:TRANsition:SLOPe? query returns the current transition search slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:TRANsition:SOURce" on page 962
- ":SEARch:TRANsition:TIME" on page 963

:SEARch:TRANsition:SOURce

(see page 1354)

Command Syntax

```
:SEARch:TRANsition:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :SEARch:TRANsition:SOURce command selects the channel on which to search for edge transitions.

Query Syntax

```
:SEARch:TRANsition:SOURce?
```

The :SEARch:TRANsition:SOURce? query returns the current transition search source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:MODE" on page 937
- ":SEARch:TRANsition:SLOPe" on page 961

:SEARch:TRANsition:TIME

N (see page 1354)

Command Syntax

```
:SEARch:TRANsition:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :SEARch:TRANsition:TIME command sets the time of the transition to search for. You can search for transitions greater than or less than this time.

Query Syntax

```
:SEARch:TRANsition:TIME?
```

The :SEARch:TRANsition:TIME? query returns the current transition time value.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:TRANsition:QUALifier" on page 960

:SEARch:SERial:A429 Commands

Table 120 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:LABel <value> (see page 965)	:SEARch:SERial:A429:LABel? (see page 965)	<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string> from 0-255 <hex> ::= #Hnn where n ::= {0,..,9 | A,..,F} <octal> ::= #Qnnn where n ::= {0,..,7} <string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
:SEARch:SERial:A429:MODE <condition> (see page 966)	:SEARch:SERial:A429:MODE? (see page 966)	<condition> ::= {LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors}
:SEARch:SERial:A429:PATTern:DATA <string> (see page 967)	:SEARch:SERial:A429:PATTern:DATA? (see page 967)	<string> ::= "nn...n" where n ::= {0 | 1}, length depends on FORMat
:SEARch:SERial:A429:PATTern:SDI <string> (see page 968)	:SEARch:SERial:A429:PATTern:SDI? (see page 968)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
:SEARch:SERial:A429:PATTern:SSM <string> (see page 969)	:SEARch:SERial:A429:PATTern:SSM? (see page 969)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits

:SEARch:SERial:A429:LABel

(see page 1354)

Command Syntax

```
:SEARch:SERial:A429:LABel <value>

<value> ::= 8-bit integer in decimal, <hex>, <octal>, or <string>
            from 0-255

<hex> ::= #Hnn where n ::= {0,..,9 | A,..,F}

<octal> ::= #Qnnn where n ::= {0,..,7}

<string> ::= "0xnn" where n::= {0,..,9 | A,..,F}
```

The :SEARch:SERial:A429:LABel command defines the ARINC 429 label value when labels are used in the selected search mode.

Query Syntax

```
:SEARch:SERial:A429:LABel?
```

The :SEARch:SERial:A429:LABel? query returns the current label value in decimal format.

Return Format

```
<value><NL> in decimal format
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SEARch:SERial:A429:MODE" on page 966

:SEARch:SERial:A429:MODE

(see page 1354)

Command Syntax

```
:SEARch:SERial:A429:MODE <condition>

<condition> ::= {LABel | LBITs | PERRor | WERRor | GERRor | WGERrors
                 | ALLerrors}
```

The :SEARch:SERial:A429:MODE command selects the type of ARINC 429 information to find in the Lister display:

- LABel – finds the specified label value.
- LBITs – finds the label and the other word fields as specified.
- PERRor – finds words with a parity error.
- WERRor – finds an intra-word coding error.
- GERRor – finds an inter-word gap error.
- WGERrors – finds either a Word or Gap Error.
- ALLerrors – finds any of the above errors.

Query Syntax

```
:SEARch:SERial:A429:MODE?
```

The :SEARch:SERial:A429:MODE? query returns the current ARINC 429 search mode condition.

Return Format

```
<condition><NL>

<condition> ::= {LAB | LBIT | PERR | WERR | GERR | WGER | ALL}
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :SBUS<n> Commands" on page 721
- ":SBUS<n>:MODE" on page 725
- ":SEARch:SERial:A429:LABel" on page 965
- ":SEARch:SERial:A429:PATTern:DATA" on page 967
- ":SEARch:SERial:A429:PATTern:SDI" on page 968
- ":SEARch:SERial:A429:PATTern:SSM" on page 969
- ":SBUS<n>:A429:SOURce" on page 736

:SEARch:SERial:A429:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SEARch:SERial:A429:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1}, length depends on FORMat
```

The :SEARch:SERial:A429:PATTern:DATA command defines the ARINC 429 data pattern resource according to the string parameter. This pattern controls the data pattern searched for in each ARINC 429 word.

NOTE If more bits are sent for <string> than specified by the :SBUS<n>:A429:FORMat command, the most significant bits will be truncated.

Query Syntax

```
:SEARch:SERial:A429:PATTern:DATA?
```

The :SEARch:SERial:A429:PATTern:DATA? query returns the current settings of the specified ARINC 429 data pattern resource in the binary string format.

Return Format

```
<string><NL>
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SEARch:SERial:A429:MODE" on page 966
- ":SEARch:SERial:A429:PATTern:SDI" on page 968
- ":SEARch:SERial:A429:PATTern:SSM" on page 969

:SEARch:SERial:A429:PATTern:SDI

(see page 1354)

Command Syntax

```
:SEARch:SERial:A429:PATTern:SDI <string>

<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
```

The :SEARch:SERial:A429:PATTern:SDI command defines the ARINC 429 two-bit SDI pattern resource according to the string parameter. This pattern controls the SDI pattern searched for in each ARINC 429 word.

The specified SDI is only used if the :SBUS<n>:A429:FORMat includes the SDI field.

Query Syntax

```
:SEARch:SERial:A429:PATTern:SDI?
```

The :SEARch:SERial:A429:PATTern:SDI? query returns the current settings of the specified ARINC 429 two-bit SDI pattern resource in the binary string format.

Return Format

```
<string><NL>
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:FORMat" on page 734
- ":SEARch:SERial:A429:MODE" on page 966
- ":SEARch:SERial:A429:PATTern:DATA" on page 967
- ":SEARch:SERial:A429:PATTern:SSM" on page 969

:SEARch:SERial:A429:PATTern:SSM

N (see page 1354)

Command Syntax

```
:SEARch:SERial:A429:PATTern:SSM <string>

<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
```

The :SEARch:SERial:A429:PATTern:SSM command defines the ARINC 429 two-bit SSM pattern resource according to the string parameter. This pattern controls the SSM pattern searched for in each ARINC 429 word.

The specified SSM is only used if the :SBUS<n>:A429:FORMat includes the SSM field.

Query Syntax

```
:SEARch:SERial:A429:PATTern:SSM?
```

The :SEARch:SERial:A429:PATTern:SSM? query returns the current settings of the specified ARINC 429 two-bit SSM pattern resource in the binary string format.

Return Format

```
<string><NL>
```

Errors

- "-241, Hardware missing" on page 1295

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":SBUS<n>:A429:FORMat" on page 734
- ":SEARch:SERial:A429:MODE" on page 966
- ":SEARch:SERial:A429:PATTern:DATA" on page 967
- ":SEARch:SERial:A429:PATTern:SDI" on page 968

:SEARch:SERial:CAN Commands

Table 121 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 971)	:SEARch:SERial:CAN:MODE? (see page 971)	<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload | MESSage | MSIGnal}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 973)	:SEARch:SERial:CAN:PATTern:DATA? (see page 973)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 974)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 974)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 975)	:SEARch:SERial:CAN:PATTern:ID? (see page 975)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 976)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 976)	<value> ::= {STANdard | EXTended}
:SEARch:SERial:CAN:SYMBolic:MESSage <name> (see page 977)	:SEARch:SERial:CAN:SYMBolic:MESSage? (see page 977)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:SIGNal <name> (see page 978)	:SEARch:SERial:CAN:SYMBolic:SIGNal? (see page 978)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:VALue <data> (see page 979)	:SEARch:SERial:CAN:SYMBolic:VALue? (see page 979)	<data> ::= value in NR3 format

:SEARch:SERial:CAN:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:MODE <value>

<value> ::= {IDEither | IDData | DATA | IDRemote | ERRor | ACKerror
    | FORMerror | STUFferror | CRCerror | ALLerrors | OVERload
    | MESSage | MSIGnal}
```

The :SEARch:SERial:CAN:MODE command selects the type of CAN information to find in the Lister display:

Condition	Front-panel name	Description
IDEither	Frame ID	Finds remote or data frames matching the specified ID.
IDData	Data Frame ID	Finds data frames matching the specified ID.
DATA	Data Frame ID and Data	Finds data frames matching the specified ID and data.
IDRemote	Remote Frame ID	Finds remote frames with the specified ID.
ERRor	Error Frame	Finds CAN active error frames.
ACKerror	Acknowledge Error	Finds the acknowledge bit if the polarity is incorrect.
FORMerror	Form Error	Finds reserved bit errors.
STUFferror	Stuff Error	Finds 6 consecutive 1s or 6 consecutive 0s, while in a non-error or non overload frame.
CRCerror	CRC Field Error	Finds when the calculated CRC does not match the transmitted CRC.
ALLerrors	All Errors	Finds any form error or active error.
OVERload	Overload Frame	Finds CAN overload frames.
MESSage	Message	Finds a symbolic message.
MSIGnal	Message and Signal (non-FD)	Finds a symbolic message and a signal value.

Query Syntax

```
:SEARch:SERial:CAN:MODE?
```

The :SEARch:SERial:CAN:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {IDE | IDD | DATA | IDR | ERR | ACK | FORM | STUF | CRC
    | ALL | OVER | MESS | MSIG}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:CAN:PATTern:DATA" on page 973
- ":SEARch:SERial:CAN:PATTern:ID" on page 975

- ":RECall:DBC[:STARt]" on page 684
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 977
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 978
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 979

:SEARch:SERial:CAN:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:DATA <string>

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
             for hexadecimal
```

The :SEARch:SERial:CAN:PATTern:DATA command specifies the data value when searching for Data Frame ID and Data.

The length of the data value is specified using the :SEARch:SERial:CAN:PATTern:DATA:LENGth command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:DATA?
```

The :SEARch:SERial:CAN:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
             for hexadecimal
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:PATTern:DATA:LENGth" on page 974

:SEARch:SERial:CAN:PATTern:DATA:LENGth

(see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SEARch:SERial:CAN:PATTern:DATA:LENGth command specifies the length of the data value when searching for Data Frame ID and Data.

The data value is specified using the :SEARch:SERial:CAN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:DATA:LENGth?
```

The :SEARch:SERial:CAN:PATTern:DATA:LENGth? query returns the current data length setting.

Return Format

```
<length><NL>

<length> ::= integer from 1 to 8 in NR1 format
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:PATTern:DATA" on page 973

:SEARch:SERial:CAN:PATTern:ID

(see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:ID <string>

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
             for hexadecimal
```

The :SEARch:SERial:CAN:PATTern:ID command specifies the ID value when searching for a CAN event.

The value can be a standard ID or an extended ID, depending on the :SEARch:SERial:CAN:PATTern:ID:MODE command's setting.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:ID?
```

The :SEARch:SERial:CAN:PATTern:ID? query returns the current ID value setting.

Return Format

```
<string><NL>

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
             for hexadecimal
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:PATTern:ID:MODE" on page 976

:SEARch:SERial:CAN:PATTern:ID:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:PATTern:ID:MODE <value>

<value> ::= {STANdard | EXTended}
```

The :SEARch:SERial:CAN:PATTern:ID:MODE command specifies whether a standard ID value or an extended ID value is used when searching for a CAN event.

The ID value is specified using the :SEARch:SERial:CAN:PATTern:ID command.

Query Syntax

```
:SEARch:SERial:CAN:PATTern:ID:MODE?
```

The :SEARch:SERial:CAN:PATTern:ID:MODE? query returns the current setting.

Return Format

```
<value><NL>

<value> ::= {STAN | EXT}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:PATTern:ID" on page 975

:SEARch:SERial:CAN:SYMBolic:MESSage

N (see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:MESSage <name>

<name> ::= quoted ASCII string
```

The :SEARch:SERial:CAN:SYMBolic:MESSage command specifies the message to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MESSage or MSIGnal.

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:MESSage?
```

The :SEARch:SERial:CAN:SYMBolic:MESSage? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quotes ASCII string
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 978
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 979

:SEARch:SERial:CAN:SYMBolic:SIGNal

(see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:SIGNal <name>

<name> ::= quoted ASCII string
```

The :SEARch:SERial:CAN:SYMBolic:SIGNal command specifies the signal to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:SIGNal?
```

The :SEARch:SERial:CAN:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 977
- ":SEARch:SERial:CAN:SYMBolic:VALue" on page 979

:SEARch:SERial:CAN:SYMBolic:VALue

N (see page 1354)

Command Syntax

```
:SEARch:SERial:CAN:SYMBolic:VALue <data>

<data> ::= value in NR3 format
```

The :SEARch:SERial:CAN:SYMBolic:VALue command specifies the signal value to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SEARch:SERial:CAN:SYMBolic:VALue?
```

The :SEARch:SERial:CAN:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

<data> ::= value in NR3 format
```

See Also

- ":RECall:DBC[:STARt]" on page 684
- ":SEARch:SERial:CAN:MODE" on page 971
- ":SEARch:SERial:CAN:SYMBolic:MESSage" on page 977
- ":SEARch:SERial:CAN:SYMBolic:SIGNal" on page 978

:SEARch:SERial:IIC Commands

Table 122 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 981)	:SEARch:SERial:IIC:MODE? (see page 981)	<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | READEprom | READ7 | WRITE7 | R7Data2 | W7Data2}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 983)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 983)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 984)	:SEARch:SERial:IIC:PATTern:DATA? (see page 984)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 985)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 985)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:QUALifier <value> (see page 986)	:SEARch:SERial:IIC:QUALifier? (see page 986)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

:SEARch:SERial:IIC:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:IIC:MODE <value>

<value> ::= {RESTart | ADDRess | ANACk | NACKnowledge | READEprom
   | READ7 | WRITE7 | R7Data2 | W7Data2}
```

The :SEARch:SERial:IIC:MODE command selects the type of IIC information to find in the Lister display:

- RESTart – searches for another start condition occurring before a stop condition.
- ADDRess – searches for a packet with the specified address, ignoring the R/W bit.
- ANACk – searches for address with no acknowledge events.
- NACKnowledge – searches for missing acknowledge events.
- READEprom – searches for EEPROM data reads.
- READ7 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data. The value READ is also accepted for READ7.
- WRITe7 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data. The value WRITe is also accepted for WRITe7.
- R7Data2 – searches for 7-bit address frames containing Start:Address7:Read:Ack:Data:Ack:Data2.
- W7Data2 – searches for 7-bit address frames containing Start:Address7:Write:Ack:Data:Ack:Data2.

NOTE **The short form of READ7 (READ7), READEprom (READE), and WRITe7 (WRIT7) do not follow the defined Long Form to Short Form Truncation Rules (see page 1356).**

When searching for events containing addresses, address values are specified using the :SEARch:SERial:IIC:PATTern:ADDRess command.

When searching for events containing data, data values are specified using the :SEARch:SERial:IIC:PATTern:DATA and :SEARch:SERial:IIC:PATTern:DATA2 commands.

Query Syntax

```
:SEARch:SERial:IIC:MODE?
```

The :SEARch:SERial:IIC:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {REST | ADDR | ANAC | NACK | READE | READ7 | WRITE7
   | R7D2 | W7D2}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:IIC:PATTern:ADDRess" on page 983
- ":SEARch:SERial:IIC:PATTern:DATA" on page 984
- ":SEARch:SERial:IIC:PATTern:DATA2" on page 985
- ":SEARch:SERial:IIC:QUALifier" on page 986

:SEARch:SERial:IIC:PATTern:ADDRess

N (see page 1354)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:ADDRess <value>

<value> ::= integer or <string>

<string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
```

The :SEARch:SERial:IIC:PATTern:ADDRess command specifies address values when searching for IIC events.

To set don't care values, use the integer -1.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:ADDRess?
```

The :SEARch:SERial:IIC:PATTern:ADDRess? query returns the current address value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:IIC:MODE" on page 981

:SEARch:SERial:IIC:PATTern:DATA

(see page 1354)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:DATA <value>

<value> ::= integer or <string>

<string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
```

The :SEARch:SERial:IIC:PATTern:DATA command specifies data values when searching for IIC events.

To set don't care values, use the integer -1.

When searching for IIC EEPROM data read events, you specify the data value qualifier using the :SEARch:SERial:IIC:QUALifier command.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:DATA?
```

The :SEARch:SERial:IIC:PATTern:DATA? query returns the current data value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:IIC:MODE" on page 981
- ":SEARch:SERial:IIC:QUALifier" on page 986
- ":SEARch:SERial:IIC:PATTern:DATA2" on page 985

:SEARch:SERial:IIC:PATTern:DATA2

N (see page 1354)

Command Syntax

```
:SEARch:SERial:IIC:PATTern:DATA2 <value>

<value> ::= integer or <string>

<string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
```

The :SEARch:SERial:IIC:PATTern:DATA2 command specifies the second data value when searching for IIC events with two data values.

To set don't care values, use the integer -1.

Query Syntax

```
:SEARch:SERial:IIC:PATTern:DATA2?
```

The :SEARch:SERial:IIC:PATTern:DATA2? query returns the current second data value setting.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:IIC:MODE" on page 981
- ":SEARch:SERial:IIC:PATTern:DATA" on page 984

:SEARch:SERial:IIC:QUALifier

(see page 1354)

Command Syntax

```
:SEARch:SERial:IIC:QUALifier <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :SEARch:SERial:IIC:QUALifier command specifies the data value qualifier used when searching for IIC EEPROM data read events.

Query Syntax

```
:SEARch:SERial:IIC:QUALifier?
```

The :SEARch:SERial:IIC:QUALifier? query returns the current data value qualifier setting.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | LESS | GRE}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:IIC:MODE" on page 981
- ":SEARch:SERial:IIC:PATTern:DATA" on page 984

:SEARch:SERial:LIN Commands

Table 123 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 989)	:SEARch:SERial:LIN:ID? (see page 989)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with AUTO license) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SEARch:SERial:LIN:MODE <value> (see page 990)	:SEARch:SERial:LIN:MODE? (see page 990)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 991)	:SEARch:SERial:LIN:PATTern:DATA? (see page 991)	When :SEARch:SERial:LIN:PATTern:FORMat DECimal, <string> ::= "n" where n ::= 32-bit integer in unsigned decimal, returns "$" if data has any don't cares When :SEARch:SERial:LIN:PATTern:FORMat HEX, <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length> (see page 992)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 992)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 993)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 993)	<base> ::= {HEX | DECimal}
:SEARch:SERial:LIN:SYMBolic:FRAMe <name> (see page 994)	:SEARch:SERial:LIN:SYMBolic:FRAMe? (see page 994)	<name> ::= quoted ASCII string

Table 123 :SEARch:SERial:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:SYMBolic:SIGNal <name> (see page 995)	:SEARch:SERial:LIN:SYMBolic:SIGNal? (see page 995)	<name> ::= quoted ASCII string
:SEARch:SERial:LIN:SYMBolic:VALue <data> (see page 996)	:SEARch:SERial:LIN:SYMBolic:VALue? (see page 996)	<data> ::= value in NR3 format

:SEARch:SERial:LIN:ID

(see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
            from 0-63 or 0x00-0x3f (with AUTO license)

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :SEARch:SERial:LIN:ID command specifies the frame ID value when searching for LIN events.

Query Syntax

```
:SEARch:SERial:LIN:ID?
```

The :SEARch:SERial:LIN:ID? query returns the current frame ID setting.

Return Format

```
<value><NL>

<value> ::= 7-bit integer in decimal (with AUTO license)
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:LIN:MODE" on page 990

:SEARch:SERial:LIN:MODE

 (see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:MODE <value>

<value> ::= {ID | DATA | ERRor | FRAMe | FSIGnal}
```

The :SEARch:SERial:LIN:MODE command selects the type of LIN information to find in the Lister display:

- ID – searches for a frame ID.
- DATA – searches for a frame ID and data.
- ERRor – searches for errors.
- FRAMe – searches for symbolic frames.
- FSIGnal – searched for symbolic frames and a signal values.

Frame IDs are specified using the :SEARch:SERial:LIN:ID command.

Data values are specified using the :SEARch:SERial:LIN:PATTern:DATA command.

Frames, signals, and signal values are specified using the :SEARch:SERial:LIN:SYMBolic:FRAMe, :SEARch:SERial:LIN:SYMBolic:SIGNal, and :SEARch:SERial:LIN:SYMBolic:VALue commands. LIN symbolic data files are loaded (recalled) using the :RECall:LDF[:STARt] command.

Query Syntax

```
:SEARch:SERial:LIN:MODE?
```

The :SEARch:SERial:LIN:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {ID | DATA | ERR | FRAM | FSIG}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:LIN:ID" on page 989
- ":SEARch:SERial:LIN:PATTern:DATA" on page 991
- ":RECall:LDF[:STARt]" on page 686
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 994
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 995
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 996

:SEARch:SERial:LIN:PATTern:DATA

N (see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:DATA <string>

When :SEARch:SERial:LIN:PATTern:FORMat DECimal,
<string> ::= "n" where n ::= 32-bit integer in unsigned decimal

When :SEARch:SERial:LIN:PATTern:FORMat HEX,
<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
```

The :SEARch:SERial:LIN:PATTern:DATA command specifies the data value when searching for LIN events.

The number base of the value entered with this command is specified using the :SEARch:SERial:LIN:PATTern:FORMat command. To set don't care values with the DATA command, the FORMat must be HEX.

The length of the data value entered is specified using the :SEARch:SERial:LIN:PATTern:DATA:LENGth command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:DATA?
```

The :SEARch:SERial:LIN:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

When :SEARch:SERial:LIN:PATTern:FORMat DECimal,
<string> ::= "n" where n ::= 32-bit integer in unsigned decimal or
             "$" if data has any don't cares

When :SEARch:SERial:LIN:PATTern:FORMat HEX,
<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:LIN:MODE" on page 990
- ":SEARch:SERial:LIN:PATTern:FORMat" on page 993
- ":SEARch:SERial:LIN:PATTern:DATA:LENGth" on page 992

:SEARch:SERial:LIN:PATTern:DATA:LENGth

 (see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length>

<length> ::= integer from 1 to 8 in NR1 format
```

The :SEARch:SERial:LIN:PATTern:DATA:LENGth command specifies the the length of the data value when searching for LIN events.

The data value is specified using the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:DATA:LENGth?
```

The :SEARch:SERial:LIN:PATTern:DATA:LENGth? query returns the current data value length setting.

Return Format

```
<length><NL>

<length> ::= integer from 1 to 8 in NR1 format
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:LIN:PATTern:DATA" on page 991

:SEARch:SERial:LIN:PATTern:FORMat

(see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:PATTern:FORMat <base>

<base> ::= {HEX | DECimal}
```

The :SEARch:SERial:LIN:PATTern:FORMat command specifies the number base used with the :SEARch:SERial:LIN:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:LIN:PATTern:FORMat?
```

The :SEARch:SERial:LIN:PATTern:FORMat? query returns the current number base setting.

Return Format

```
<base><NL>

<base> ::= {HEX | DEC}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:LIN:PATTern:DATA" on page 991

:SEARch:SERial:LIN:SYMBolic:FRAMe

(see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:FRAMe <name>

<name> ::= quoted ASCII string
```

The :SEARch:SERial:LIN:SYMBolic:FRAMe command specifies the message to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FRAMe or FSIGnal.

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:FRAMe?
```

The :SEARch:SERial:LIN:SYMBolic:FRAMe? query returns the specified message.

Return Format

```
<name><NL>

<name> ::= quotes ASCII string
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SEARch:SERial:LIN:MODE" on page 990
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 995
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 996

:SEARch:SERial:LIN:SYMBolic:SIGNal

N (see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:SIGNal <name>

<name> ::= quoted ASCII string
```

The :SEARch:SERial:LIN:SYMBolic:SIGNal command specifies the signal to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FSIGnal.

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:SIGNal?
```

The :SEARch:SERial:LIN:SYMBolic:SIGNal? query returns the specified signal.

Return Format

```
<name><NL>

<name> ::= quoted ASCII string
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SEARch:SERial:LIN:MODE" on page 990
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 994
- ":SEARch:SERial:LIN:SYMBolic:VALue" on page 996

:SEARch:SERial:LIN:SYMBolic:VALue

(see page 1354)

Command Syntax

```
:SEARch:SERial:LIN:SYMBolic:VALue <data>

<data> ::= value in NR3 format
```

The :SEARch:SERial:LIN:SYMBolic:VALue command specifies the signal value to search for when LIN symbolic data has been loaded (recalled) into the oscilloscope and the LIN serial search mode is set to FSIGnal.

NOTE Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax

```
:SEARch:SERial:LIN:SYMBolic:VALue?
```

The :SEARch:SERial:LIN:SYMBolic:VALue? query returns the specified signal value.

Return Format

```
<data><NL>

<data> ::= value in NR3 format
```

See Also

- ":RECall:LDF[:STARt]" on page 686
- ":SEARch:SERial:LIN:MODE" on page 990
- ":SEARch:SERial:LIN:SYMBolic:FRAMe" on page 994
- ":SEARch:SERial:LIN:SYMBolic:SIGNal" on page 995

:SEARch:SERial:M1553 Commands

Table 124 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553:MODE <value> (see page 998)	:SEARch:SERial:M1553:MODE? (see page 998)	<value> ::= {DSTArt | CSTArt | RTA | RTA11 | PERRor | SERRor | MERRor}
:SEARch:SERial:M1553:PATTern:DATA <string> (see page 999)	:SEARch:SERial:M1553:PATTern:DATA? (see page 999)	<string> ::= "nn...n" where n ::= {0 | 1}
:SEARch:SERial:M1553:RTA <value> (see page 1000)	:SEARch:SERial:M1553:RTA? (see page 1000)	<value> ::= 5-bit integer in decimal, <hexadecimal>, <binary>, or <string> from 0-31 < hexadecimal > ::= #Hnn where n ::= {0,..,9|A,..,F} <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n::= {0,..,9|A,..,F}

:SEARch:SERial:M1553:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:M1553:MODE <value>

<value> ::= {DSTArt | CSTArt | RTA | RTA11 | PERRor | SERRor | MERRor}
```

The :SEARch:SERial:M1553:MODE command selects the type of MIL-STD-1553 information to find in the Lister display:

- DSTArt – searches for the start of a Data word (at the end of a valid Data Sync pulse).
- CSTArt – searches for the start of a Command/Status word (at the end of a valid C/S Sync pulse).
- RTA – searches for the Remote Terminal Address (RTA) of a Command/Status word.
- RTA11 – searches for the Remote Terminal Address (RTA) and the additional 11 bits of a Command/Status word.
- PERRor – searches for (odd) parity errors for the data in the word.
- SERRor – searches for invalid Sync pulses.
- MERRor – searches for Manchester encoding errors.

In the RTA or RTA11 modes, the Remote Terminal Address is specified using the :SEARch:SERial:M1553:RTA command.

In the RTA11 mode, the additional 11 bits are specified using the :SEARch:SERial:M1553:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:M1553:MODE?
```

The :SEARch:SERial:M1553:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {DSTA | CSTA | RTA | RTA11 | PERR | SERR | MERR}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:M1553:RTA" on page 1000
- ":SEARch:SERial:M1553:PATTern:DATA" on page 999

:SEARch:SERial:M1553:PATTern:DATA

(see page 1354)

Command Syntax

```
:SEARch:SERial:M1553:PATTern:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1}
```

The :SEARch:SERial:M1553:PATTern:DATA command specifies the additional 11 bits when searching for the MIL-STD-1553 Remote Terminal Address + 11 Bits.

Query Syntax

```
:SEARch:SERial:M1553:PATTern:DATA?
```

The :SEARch:SERial:M1553:PATTern:DATA? query returns the current value setting for the additional 11 bits.

Return Format

```
<string><NL>

<string> ::= "nn...n" where n ::= {0 | 1}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:M1553:MODE" on page 998

:SEARch:SERial:M1553:RTA

(see page 1354)

Command Syntax

```
:SEARch:SERial:M1553:RTA <value>

<value> ::= 5-bit integer in decimal, <hexadecimal>, <binary>,
            or <string> from 0-31

<hexadecimal> ::= #Hnn where n ::= {0,..,9|A,..,F}

<binary> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn" where n::= {0,..,9|A,..,F}
```

The :SEARch:SERial:M1553:RTA command specifies the Remote Terminal Address (RTA) value when searching for MIL-STD-1553 events.

Query Syntax

```
:SEARch:SERial:M1553:RTA?
```

The :SEARch:SERial:M1553:RTA? query returns the current Remote Terminal Address value setting.

Return Format

```
<value><NL>

<value> ::= 5-bit integer in decimal from 0-31
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933

:SEARch:SERial:SENT Commands

Table 125 :SEARch:SERial:SENT Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SENT:FAST:DATA <string> (see page 1002)	:SEARch:SERial:SENT:FAST:DATA? (see page 1002)	<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
:SEARch:SERial:SENT:MODE <mode> (see page 1003)	:SEARch:SERial:SENT:MODE? (see page 1003)	<mode> ::= {FCData | SCMid | SCData | CRCerror PPERror}
:SEARch:SERial:SENT:SLOW:DATA <data> (see page 1004)	:SEARch:SERial:SENT:SLOW:DATA? (see page 1004)	<data> ::= from -1 (don't care) to 65535, in NR1 format.
:SEARch:SERial:SENT:SLOW:ID <id> (see page 1005)	:SEARch:SERial:SENT:SLOW:ID? (see page 1005)	<id> ::= from -1 (don't care) to 255, in NR1 format.

:SEARch:SERial:SENT:FAST:DATA

N (see page 1354)

Command Syntax

```
:SEARch:SERial:SENT:FAST:DATA <string>

<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
```

The :SEARch:SERial:SENT:FAST:DATA command specifies the status and data nibbles that will be searched for when the FCData search mode is chosen.

Query Syntax

```
:SEARch:SERial:SENT:FAST:DATA?
```

The :SEARch:SERial:SENT:FAST:DATA? query returns the fast channel data search value setting.

Return Format

```
<string><NL>

<string> ::= "0xn..." where n ::= {0,..,9 | A,..,F | X | $}
```

See Also

- ":SEARch:SERial:SENT:MODE" on page 1003
- ":SEARch:SERial:SENT:SLOW:DATA" on page 1004
- ":SEARch:SERial:SENT:SLOW:ID" on page 1005

:SEARch:SERial:SENT:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:SENT:MODE <mode>

<mode> ::= {FCData | SCMid | SCData | CRCerror | PPERror}
```

When SENT serial decode is turned on and displayed in the Lister, the :SEARch:SERial:SENT:MODE command specifies what to search for in the decoded data:

- FCData – finds Fast Channel data nibbles that match the values entered using additional softkeys.
- SCMid – finds Slow Channel Message IDs that match the value entered using additional softkeys.
- SCData – finds Slow Channel Message IDs and Data that match the values entered using additional softkeys.
- CRCerror – finds any CRC error, Fast or Slow.
- PPERror – finds where a nibble is either too wide or too narrow (for example, data nibble < 12 (11.5) or > 27 (27.5) ticks wide). Sync, S&C, data, or checksum pulse periods are checked.

Query Syntax

```
:SEARch:SERial:SENT:MODE?
```

The :SEARch:SERial:SENT:MODE? query returns the search mode setting.

Return Format

```
<mode><NL>

<mode> ::= {FCD | SCM | SCD | CRC | PPER}
```

See Also

- ":SEARch:SERial:SENT:FAST:DATA" on page 1002
- ":SEARch:SERial:SENT:SLOW:DATA" on page 1004
- ":SEARch:SERial:SENT:SLOW:ID" on page 1005

:SEARch:SERial:SENT:SLOW:DATA

 (see page 1354)

Command Syntax

```
:SEARch:SERial:SENT:SLOW:DATA <data>

<data> ::= from -1 (don't care) to 65535, in NR1 format.
```

The :SEARch:SERial:SENT:SLOW:DATA command specifies the data to search for in the Slow Channel Message ID and Data search mode.

Query Syntax

```
:SEARch:SERial:SENT:SLOW:DATA?
```

The :SEARch:SERial:SENT:SLOW:DATA? query returns the slow channel data search value setting.

Return Format

```
<data><NL>

<data> ::= from -1 (don't care) to 65535, in NR1 format.
```

See Also

- ":SEARch:SERial:SENT:FAST:DATA" on page 1002
- ":SEARch:SERial:SENT:MODE" on page 1003
- ":SEARch:SERial:SENT:SLOW:ID" on page 1005

:SEARch:SERial:SENT:SLOW:ID

 (see page 1354)

Command Syntax

```
:SEARch:SERial:SENT:SLOW:ID <id>

<id> ::= from -1 (don't care) to 255, in NR1 format.
```

The :SEARch:SERial:SENT:SLOW:ID command specifies the ID to search for in the "Slow Channel Message ID" and "Slow Channel Message ID & Data" trigger modes. The ID can be from -1 (don't care) to 255 (depending on the message length).

Query Syntax

```
:SEARch:SERial:SENT:SLOW:ID?
```

The :SEARch:SERial:SENT:SLOW:ID? query returns the slow channel ID search value setting.

Return Format

```
<id><NL>

<id> ::= from -1 (don't care) to 255, in NR1 format.
```

See Also

- ":SEARch:SERial:SENT:FAST:DATA" on page 1002
- ":SEARch:SERial:SENT:MODE" on page 1003
- ":SEARch:SERial:SENT:SLOW:DATA" on page 1004

:SEARch:SERial:UART Commands

Table 126 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 1007)	:SEARch:SERial:UART:DATA? (see page 1007)	<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal, <hexadecimal>, <binary>, or <quoted_string> format <hexadecimal> ::= #Hnn where n ::= {0,..,9| A,..,F} for hexadecimal <binary> ::= #Bnn...n where n ::= {0 | 1} for binary <quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or standard abbreviations)
:SEARch:SERial:UART:MODE <value> (see page 1008)	:SEARch:SERial:UART:MODE? (see page 1008)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | PARityerror | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 1009)	:SEARch:SERial:UART:QUALifier? (see page 1009)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

:SEARch:SERial:UART:DATA

(see page 1354)

Command Syntax

```
:SEARch:SERial:UART:DATA <value>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal,
            <hexadecimal>, <binary>, or <quoted_string> format

<hexadecimal> ::= #Hnn where n ::= {0,..,9| A,..,F} for hexadecimal

<binary> ::= #Bnn...n where n ::= {0 | 1} for binary

<quoted_string> ::= any of the 128 valid 7-bit ASCII characters (or
                    standard abbreviations)
```

The :SEARch:SERial:UART:DATA command specifies a data value when searching for UART/RS232 events.

The data value qualifier is specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax

```
:SEARch:SERial:UART:DATA?
```

The :SEARch:SERial:UART:DATA? query returns the current data value setting.

Return Format

```
<value><NL>

<value> ::= 8-bit integer from 0-255 (0x00-0xff) in decimal format
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:UART:MODE" on page 1008
- ":SEARch:SERial:UART:QUALifier" on page 1009

:SEARch:SERial:UART:MODE

N (see page 1354)

Command Syntax

```
:SEARch:SERial:UART:MODE <value>

<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX
             | PARityerror | AERRor}
```

The :SEARch:SERial:UART:MODE command selects the type of UART/RS232 information to find in the Lister display:

- RDATa – searches for a receive data value when data words are from 5 to 8 bits long.
- RD1 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 1.
- RD0 – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is 0.
- RDX – searches for a receive data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- TDATa – searches for a transmit data value when data words are from 5 to 8 bits long.
- TD1 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 1.
- TD0 – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is 0.
- TDX – searches for a transmit data value when data words are 9 bits long and the 9th (alert) bit is a don't care (X).
- PARityerror – searches for parity errors.
- AERRor – searches for any error.

Data values are specified using the :SEARch:SERial:UART:DATA command.

Data value qualifiers are specified using the :SEARch:SERial:UART:QUALifier command.

Query Syntax

```
:SEARch:SERial:UART:MODE?
```

The :SEARch:SERial:UART:MODE? query returns ...

Return Format

```
<value><NL>

<value> ::= {RDAT | RD1 | RD0 | RDX | TDAT | TD1 | TD0 | TDX | PAR
             | AERR}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:UART:DATA" on page 1007
- ":SEARch:SERial:UART:QUALifier" on page 1009

:SEARch:SERial:UART:QUALifier

(see page 1354)

Command Syntax

```
:SEARch:SERial:UART:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
```

The :SEARch:SERial:UART:QUALifier command specifies the data value qualifier when searching for UART/RS232 events.

Query Syntax

```
:SEARch:SERial:UART:QUALifier?
```

The :SEARch:SERial:UART:QUALifier? query returns the current data value qualifier setting.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}
```

See Also

- Chapter 28, ":SEARch Commands," starting on page 933
- ":SEARch:SERial:UART:DATA" on page 1007

29 :SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 1013.

Table 127 :SYSTem Commands Summary

Command	Query	Options and Query Returns
n/a	:SYSTem:DATE? (see page 1014)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,..,12} <day> ::= {1,..31}
:SYSTem:DSP <string> (see page 1015)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 1016)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1293).
:SYSTem:GUI:SHOW {{0 | OFF} | {1 | ON}} (see page 1017)	:SYSTem:GUI:SHOW? (see page 1017)	<setting> ::= {0 | 1}
:SYSTem:LOCK <value> (see page 1018)	:SYSTem:LOCK? (see page 1018)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:PERSona[:MANufacturer] <manufacturer_string> (see page 1019)	:SYSTem:PERSona[:MANufacturer]? (see page 1019)	<manufacturer_string> ::= quoted ASCII string, up to 63 characters
:SYSTem:PERSona[:MANufacturer]:DEFault (see page 1020)	n/a	Sets manufacturer string to "KEYSIGHT TECHNOLOGIES"
:SYSTem:PRECision {{1 | ON} | {0 | OFF}} (see page 1021)	:SYSTem:PRECision? (see page 1021)	{1 | 0}

Table 127 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:PRECision:LENGth <length> (see page 1022)	:SYSTem:PRECision:LENGth? (see page 1022)	<length> ::= between 100k and 1M points in NR1 format
:SYSTem:PRESet (see page 1023)	n/a	See :SYSTem:PRESet (see page 1023)
:SYSTem:PROTection:LOCK <value> (see page 1026)	:SYSTem:PROTection:LOCK? (see page 1026)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]] (see page 1027)	n/a	<setting> ::= {{0 | OFF} | {1 | ON}} <file_name> ::= quoted ASCII string <write_mode> ::= {CREate | APPend}
:SYSTem:RLOGger:DESTination <dest> (see page 1028)	:SYSTem:RLOGger:DESTination? (see page 1028)	<dest> ::= {FILE | SCReen | BOTH}
:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}} (see page 1029)	:SYSTem:RLOGger:DISPlay? (see page 1029)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:FNAMe <file_name> (see page 1030)	:SYSTem:RLOGger:FNAMe? (see page 1030)	<file_name> ::= quoted ASCII string
:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}} (see page 1031)	:SYSTem:RLOGger:STATe? (see page 1031)	<setting> ::= {0 | 1}
:SYSTem:RLOGger:TRANsparent {{0 | OFF} | {1 | ON}} (see page 1032)	:SYSTem:RLOGger:TRANsparent? (see page 1032)	<setting> ::= 0
:SYSTem:RLOGger:WMODe <write_mode> (see page 1033)	:SYSTem:RLOGger:WMODe? (see page 1033)	<write_mode> ::= {CREate | APPend}
:SYSTem:SETup <setup_data> (see page 1034)	:SYSTem:SETup? (see page 1034)	<setup_data> ::= data in IEEE 488.2 # format.

Table 127 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SYSTem:TIME? (see page 1036)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TOUCh {{1 | ON} | {0 | OFF}} (see page 1037)	:SYSTem:TOUCh? (see page 1037)	{1 | 0}

Introduction to :SYSTem Commands

SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem:DATE

(see page 1354)

Query Syntax `:SYSTem:DATE?`

The SYSTem:DATE? query returns the date.

Return Format `<year>,<month>,<day><NL>`

`<year> ::= 4-digit year in NR1 format`

`<month> ::= {1,..,12}`

`<day> ::= {1,..,31}`

See Also

- "Introduction to :SYSTem Commands" on page 1013
- ":SYSTem:TIME" on page 1036

:SYSTem:DSP

(see page 1354)

Command Syntax

```
:SYSTem:DSP <string>

<string> ::= quoted ASCII string (up to 75 characters)
```

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box on-screen.

Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.)

Press any menu key to manually remove the message from the display.

See Also

- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:ERRor

 (see page 1354)

Query Syntax

```
:SYSTem:ERRor?
```

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

When remote logging is enabled (using the oscilloscope's front panel), additional debug information can be included in the returned error string. If the error is detected by the SCPI command parser, such as a header error or other syntax error, the extra debug information is generated and included. But if the error is detected by the oscilloscope system, such as when an out-of-range value is sent, then no extra debug information is included.

Return Format

```
<error number>,<error string><NL>

<error number> ::= an integer error code in NR1 format

<error string> ::= quoted ASCII string containing the error message
```

Error messages are listed in Chapter 36, “Error Messages,” starting on page 1293.

See Also

- "Introduction to :SYSTem Commands" on page 1013
- "*ESR (Standard Event Status Register)" on page 181
- "*CLS (Clear Status)" on page 178

:SYSTem:GUI:SHOW

(see page 1354)

Command Syntax `:SYSTem:GUI:SHOW {{0 | OFF} | {1 | ON}}`

The :SYSTem:GUI:SHOW command shows or hides the Soft Front Panel (SFP) user interface.

This remote command performs the same operation as the Show Front Panel and Hide Front Panel buttons in the Keysight M924x InfiniiVision SFP Launcher application on the PXIe chassis controller PC.

Query Syntax `:SYSTem:GUI:SHOW?`

The :SYSTem:GUI:SHOW? query returns whether the SFP is shown or hidden.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:LOCK

(see page 1354)

Command Syntax

```
:SYSTem:LOCK <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:LOCK command disables the front panel. LOCK ON is the equivalent of sending a local lockout message over the programming interface.

Query Syntax

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:PERSona[:MANufacturer]

N (see page 1354)

Command Syntax

```
:SYSTem:PERSona[:MANufacturer] <manufacturer_string>

<manufacturer_string> ::= ::= quoted ASCII string, up to 63 characters
```

The :SYSTem:PERSona[:MANufacturer] command lets you change the manufacturer string portion of the identification string returned by the *IDN? query.

The default manufacturer string is "KEYSIGHT TECHNOLOGIES".

If your remote programs depend on a legacy manufacturer string, for example, you could use this command to set the manufacturer string to "AGILENT TECHNOLOGIES".

Query Syntax

```
:SYSTem:PERSona[:MANufacturer]?
```

The :SYSTem:PERSona[:MANufacturer]? query returns the currently set manufacturer string.

Return Format

```
<manufacturer_string><NL>
```

See Also

- "*IDN (Identification Number)" on page 183
- ":SYSTem:PERSona[:MANufacturer]:DEFault" on page 1020
- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:PERSona[:MANufacturer]:DEFault

 (see page 1354)

Command Syntax `:SYSTem:PERSona[:MANufacturer]:DEFault`

The :SYSTem:PERSona[:MANufacturer]:DEFault command sets the manufacturer string to "KEYSIGHT TECHNOLOGIES".

See Also

- "*IDN (Identification Number)" on page 183
- ":SYSTem:PERSona[:MANufacturer]" on page 1019
- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:PRECision

(see page 1354)

Command Syntax

```
:SYSTem:PRECision <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:PRECision command turns the oscilloscope's precision analysis setting on or off.

- OFF (0) – provides the maximum oscilloscope waveform update rate by performing measurements and math functions on the default 65535-point (maximum) *measurement record*.
- ON (1) – at the expense of oscilloscope waveform update rate, this setting allows measurements and math functions to be performed on a longer, *precision analysis record* (see ":WAVeform:POINts:MODE" on page 1172).

  The length of the precision analysis record is specified by the :SYSTem:PRECision:LENGth command.

The precision analysis setting is OFF after a *RST command.

Precision analysis is not available when:

- Realtime sampling mode is off.
- Averaging acquisition modes is selected.
- XY or Roll time modes are selected.

Query Syntax

```
:SYSTem:PRECision?
```

The :SYSTem:PRECision? query returns the current precision analysis setting.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 1013
- ":SYSTem:PRECision:LENGth" on page 1022
- ":WAVeform:POINts:MODE" on page 1172
- "*RST (Reset)" on page 189

:SYSTem:PRECision:LENGth

 (see page 1354)

Command Syntax

```
:SYSTem:PRECision:LENGth <length>

<length> ::= between 100k and 1M points in NR1 format
```

The :SYSTem:PRECision:LENGth command specifies the length of the precision analysis record (when precision measurements and math functions mode is enabled). You can specify between 100k and 1M points.

Query Syntax

```
:SYSTem:PRECision:LENGth?
```

The :SYSTem:PRECision:LENGth? query returns the current precision analysis record length setting.

Return Format

```
<length><NL>

<length> ::= between 100k and 1M points in NR1 format
```

See Also

- "Introduction to :SYSTem Commands" on page 1013
- ":SYSTem:PRECision" on page 1021
- ":WAVeform:POINts:MODE" on page 1172
- "*RST (Reset)" on page 189

:SYSTem:PRESet

(see page 1354)

Command Syntax `:SYSTem:PRESet`

The :SYSTem:PRESet command places the instrument in a known state. This is the same as pressing the **[Default Setup]** key or **[Save/Recall] > Default/Erase > Default Setup** on the front panel.

When you perform a default setup, some user settings (like preferences) remain unchanged. To reset all user settings to their factory defaults, use the *RST command.

Reset conditions are:

Acquire Menu	
Mode	Normal
Averaging	Off
# Averages	8

Analog Channel Menu	
Channel 1	On
Channel 2	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	10:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm (cannot be changed)
Units	Volts
Skew	0

Cursor Menu	
Source	Channel 1

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

Time Base Menu	
Main time/division	100 us
Main time base delay	0.00 s
Delay time/division	500 ns
Delay time base delay	0.00 s
Reference	center
Mode	main
Vernier	Off

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

See Also
- "Introduction to Common (*) Commands" on page 176
- "*RST (Reset)" on page 189

:SYSTem:PROTection:LOCK

(see page 1354)

Command Syntax

```
:SYSTem:PROTection:LOCK <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:PROTection:LOCK command disables the fifty ohm impedance setting for all analog channels.

NOTE

Be careful when turning ON the :SYSTem:PROTection:LOCK because there is no visual indication of this setting in the front-panel user interface (other than a disabled 50 Ω channel input impedance selection), and a user could think the oscilloscope is not working properly.

Query Syntax

```
:SYSTem:PROTection:LOCK?
```

The :SYSTem:PROTection:LOCK? query returns the analog channel protection lock status.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 1013

:SYSTem:RLOGger

(see page 1354)

Command Syntax

```
:SYSTem:RLOGger <setting>[,<file_name>[,<write_mode>]]

<setting> ::= {{0 | OFF} | {1 | ON}}

<file_name> ::= quoted ASCII string

<write_mode> ::= {CREate | APPend}
```

The :SYSTem:RLOGger command enables or disables remote command logging, optionally specifying the log file name and write mode.

See Also

- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:TRANsparent" on page 1032
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:DESTination

(see page 1354)

Command Syntax

```
:SYSTem:RLOGger:DESTination <dest>

<dest> ::= {FILE | SCReen | BOTH}
```

The :SYSTem:RLOGger:DESTination command specifies whether remote commands are logged to a text file (on a connected USB storage device), logged to the screen, or both.

NOTE **If the destination is changed while remote command logging is running, remote command logging is turned off.**

Query Syntax

```
:SYSTem:RLOGger:DESTination?
```

The :SYSTem:RLOGger:DESTination? query returns the remote command logging destination.

Return Format

```
<dest><NL>

<dest> ::= {FILE | SCR | BOTH}
```

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:TRANsparent" on page 1032
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:DISPlay

N (see page 1354)

Command Syntax `:SYSTem:RLOGger:DISPlay {{0 | OFF} | {1 | ON}}`

The :SYSTem:RLOGger:DISPlay command enables or disables the screen display of logged remote commands and their return values (if applicable).

Query Syntax `:SYSTem:RLOGger:DISPlay?`

The :SYSTem:RLOGger:DISPlay? query returns whether the screen display for remote command logging is enabled or disabled.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:TRANsparent" on page 1032
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:FNAMe

(see page 1354)

Command Syntax

```
:SYSTem:RLOGger:FNAMe <file_name>

<file_name> ::= quoted ASCII string
```

The :SYSTem:RLOGger:FNAMe command specifies the remote command log file name.

Because log files are ASCII text files, the ".txt" extension is automatically added to the name specified.

Query Syntax

```
:SYSTem:RLOGger:FNAMe?
```

The :SYSTem:RLOGger:FNAMe? query returns the remote command log file name.

Return Format

```
<file_name><NL>
```

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:TRANsparent" on page 1032
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:STATe

N (see page 1354)

Command Syntax `:SYSTem:RLOGger:STATe {{0 | OFF} | {1 | ON}}`

The :SYSTem:RLOGger:STATe command enables or disables remote command logging.

Query Syntax `:SYSTem:RLOGger:STATe?`

The :SYSTem:RLOGger:STATe? query returns the remote command logging state.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:TRANsparent" on page 1032
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:TRANsparent

(see page 1354)

Command Syntax

```
:SYSTem:RLOGger:TRANsparent {0 | OFF}
```

The :SYSTem:RLOGger:TRANsparent command specifies whether the screen display background for remote command logging is transparent or solid.

Query Syntax

```
:SYSTem:RLOGger:TRANsparent?
```

The :SYSTem:RLOGger:TRANsparent? query returns the setting for transparent screen display background.

Return Format

```
<setting><NL>

<setting> ::= 0
```

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:WMODe" on page 1033

:SYSTem:RLOGger:WMODe

 (see page 1354)

Command Syntax

```
:SYSTem:RLOGger:WMODe <write_mode>

<write_mode> ::= {CREate | APPend}
```

The :SYSTem:RLOGger:WMODe command specifies the remote command logging write mode.

Query Syntax

```
:SYSTem:RLOGger:WMODe?
```

The :SYSTem:RLOGger:WMODe? query returns the remote command logging write mode.

Return Format

```
<write_mode><NL>

<write_mode> ::= {CRE | APP}
```

See Also

- ":SYSTem:RLOGger" on page 1027
- ":SYSTem:RLOGger:DESTination" on page 1028
- ":SYSTem:RLOGger:DISPlay" on page 1029
- ":SYSTem:RLOGger:FNAMe" on page 1030
- ":SYSTem:RLOGger:STATe" on page 1031
- ":SYSTem:RLOGger:TRANsparent" on page 1032

:SYSTem:SETup

(see page 1354)

Command Syntax

```
:SYSTem:SETup <setup_data>

<setup_data> ::= binary block data in IEEE 488.2 # format.
```

The :SYSTem:SETup command sets the oscilloscope as defined by the data in the setup (learn) string sent from the controller. The setup string does not change the interface mode or interface address.

Query Syntax

```
:SYSTem:SETup?
```

The :SYSTem:SETup? query operates the same as the *LRN? query. It outputs the current oscilloscope setup in the form of a learn string to the controller. The setup (learn) string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

```
<setup_data><NL>

<setup_data> ::= binary block data in IEEE 488.2 # format
```

See Also

- "Introduction to :SYSTem Commands" on page 1013
- "*LRN (Learn Device Setup)" on page 184

Example Code

```
' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
' message that contains the current state of the instrument.  Its
' format is a definite-length binary block, for example,
' #800075595<setup string><NL>
' where the setup string is 75595 bytes in length.
myScope.WriteString ":SYSTEM:SETUP?"
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors   ' After reading query results.

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"

' Open file for output.
Close #1   ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult   ' Write data.
Close #1   ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and
' write it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"

' Open file for input.
Open strPath For Binary Access Read As #1
Get #1, , varSetupString   ' Read data.
Close #1   ' Close file.
```

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:SYSTem:TIME

 (see page 1354)

Query Syntax `:SYSTem:TIME? <time>`

The :SYSTem:TIME? query returns the current system time.

Return Format `<time><NL>`

`<time> ::= hours,minutes,seconds in NR1 format`

See Also

- "Introduction to :SYSTem Commands" on page 1013
- ":SYSTem:DATE" on page 1014

:SYSTem:TOUCh

(see page 1354)

Command Syntax

```
:SYSTem:TOUCh <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:TOUCh command disables or enables the touchscreen.

Query Syntax

```
:SYSTem:TOUCh?
```

The :SYSTem:TOUCh? query returns the touchscreen's on/off status.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 1013

30 :TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 1040.

Table 128 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 1041)	:TIMebase:MODE? (see page 1041)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 1042)	:TIMebase:POSition? (see page 1042)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 1043)	:TIMebase:RANGe? (see page 1043)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase:REFClock {{0 | OFF} | {1 | ON}} (see page 1044)	:TIMebase:REFClock? (see page 1044)	{0 | 1}
:TIMebase:REFerence {LEFT | CENTer | RIGHt | CUSTom} (see page 1045)	:TIMebase:REFerence? (see page 1045)	<return_value> ::= {LEFT | CENTer | RIGHt | CUSTom}
:TIMebase:REFerence:LOCation <loc> (see page 1046)	:TIMebase:REFerence:LOCation? (see page 1046)	<loc> ::= 0.0 to 1.0 in NR3 format
:TIMebase:SCALe <scale_value> (see page 1047)	:TIMebase:SCALe? (see page 1047)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 1048)	:TIMebase:VERNier? (see page 1048)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 1049)	:TIMebase:WINDow:POSition? (see page 1049)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format

Table 128 :TIMebase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMebase:WINDow:RANGe <range_value> (see page 1050)	:TIMebase:WINDow:RANGe? (see page 1050)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 1051)	:TIMebase:WINDow:SCALe? (see page 1051)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Introduction to :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (zoomed) time bases.

Reporting the Setup

Use :TIMebase? to query setup information for the TIMebase subsystem.

Return Format

The following is a sample response from the :TIMebase? query. In this case, the query was issued following a *RST command.

```
:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

:TIMebase:MODE

(see page 1354)

Command Syntax

```
:TIMebase:MODE <value>

<value> ::= {MAIN | WINDow | XY | ROLL}
```

The :TIMebase:MODE command sets the current time base. There are four time base modes:

- MAIN – The normal time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- WINDow – In the WINDow (zoomed or delayed) time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.
- XY – In the XY mode, the :TIMebase:RANGe, :TIMebase:POSition, and :TIMebase:REFerence commands are not available. No measurements are available in this mode.
- ROLL – In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMebase:REFerence selection changes to RIGHt.

Query Syntax

```
:TIMebase:MODE?
```

The :TIMebase:MODE query returns the current time base mode.

Return Format

```
<value><NL>

<value> ::= {MAIN | WIND | XY | ROLL}
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- "*RST (Reset)" on page 189
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:POSition" on page 1042
- ":TIMebase:REFerence" on page 1045

Example Code

```
' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
myScope.WriteString ":TIMEBASE:MODE MAIN"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TIMebase:POSition

(see page 1354)

Command Syntax

```
:TIMebase:POSition <pos>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

The :TIMebase:POSition command sets the time interval between the trigger event and the display reference point on the screen. The display reference point is either left, right, or center and is set with the :TIMebase:REFerence command. The maximum position value depends on the time/division settings.

NOTE This command is an alias for the :TIMebase:DELay command.

Query Syntax

```
:TIMebase:POSition?
```

The :TIMebase:POSition? query returns the current time from the trigger to the display reference in seconds.

Return Format

```
<pos><NL>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:REFerence" on page 1045
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:SCALe" on page 1047
- ":TIMebase:WINDow:POSition" on page 1049
- ":TIMebase:DELay" on page 1291

:TIMebase:RANGe

(see page 1354)

Command Syntax

```
:TIMebase:RANGe <range_value>

<range_value> ::= time for 10 div in seconds in NR3 format
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds for the main window. The range is 10 times the current time-per-division setting.

Query Syntax

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe query returns the current full-scale range value for the main window.

Return Format

```
<range_value><NL>

<range_value> ::= time for 10 div in seconds in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:MODE" on page 1041
- ":TIMebase:SCALe" on page 1047
- ":TIMebase:WINDow:RANGe" on page 1050

Example Code

```
   ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
   ' range value is 10 times the time per division.
   myScope.WriteString ":TIM:RANG 2e-3"   ' Set the time range to 0.002
seconds.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TIMebase:REFClock

N (see page 1354)

Command Syntax

```
:TIMebase:REFClock <value>

<value> ::= {{1 | ON} | {0 | OFF}
```

The :TIMebase:REFClock command enables or disables the REF I/O MMCX connector located on the front panel of the oscilloscope.

The REF I/O MMCX connector can be used as an input for the oscilloscope's reference clock (instead of the internal 10 MHz reference), or it can be used to output the internal 10 MHz reference clock when synchronizing multiple instruments (see ":ACQuire:RSIGnal" on page 250).

The :TIMebase:REFClock ON command enables the REF I/O MMCX and sets the reference signal mode to IN. The :TIMebase:REFClock OFF command disables the REF I/O MMCX (the same as setting the reference signal mode to OFF).

Query Syntax

```
:TIMebase:REFClock?
```

The :TIMebase:REFClock? query returns the current state of the 10 MHz reference signal mode. A "1" indicates that the REF I/O input is enabled (on), and a "0" indicates that either the REF I/O MMCX is disabled (off) or that it is set as an output (by the :ACQuire:RSIGnal command).

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- ":ACQuire:RSIGnal" on page 250

:TIMebase:REFerence

 (see page 1354)

Command Syntax

```
:TIMebase:REFerence <reference>

<reference> ::= {LEFT | CENTer | RIGHt | CUSTom}
```

The :TIMebase:REFerence command sets the time reference to:

- LEFT – one division from the left side of the screen.
- CENTer – the center of the screen.
- RIGHt – one division from the right side of the screen.
- CUSTom – lets you use the :TIMebase:REFerence:LOCation command to place the time reference location at a percent of the graticule width (where 0.0 is the left edge and 1.0 is the right edge).

The time reference is the point on the display where the trigger point is referenced.

Query Syntax

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current display reference for the main window.

Return Format

```
<reference><NL>

<reference> ::= {LEFT | CENT | RIGH | CUST}
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:REFerence:LOCation" on page 1046
- ":TIMebase:MODE" on page 1041

Example Code

```
myScope.WriteString ":TIMebase:REFerence CENTer"    ' Set reference to
center.
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TIMebase:REFerence:LOCation

(see page 1354)

Command Syntax

```
:TIMebase:REFerence:LOCation <loc>

<loc> ::= 0.0 to 1.0 in NR3 format
```

When the :TIMebase:REFerence is set to CUSTom, the :TIMebase:REFerence:LOCation command lets you place the time reference location at a percent of the graticule width (where 0.0 is the left edge and 1.0 is the right edge).

Query Syntax

```
:TIMebase:REFerence:LOCation?
```

The :TIMebase:REFerence:LOCation? query returns the time base reference custom location setting.

Return Format

```
<loc><NL>

<loc> ::= 0.0 to 1.0 in NR3 format
```

See Also

- ":TIMebase:REFerence" on page 1045

:TIMebase:SCALe

 (see page 1354)

Command Syntax

```
:TIMebase:SCALe <scale_value>

<scale_value> ::= time/div in seconds in NR3 format
```

The :TIMebase:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax

```
:TIMebase:SCALe?
```

The :TIMebase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format

```
<scale_value><NL>

<scale_value> ::= time/div in seconds in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:WINDow:SCALe" on page 1051
- ":TIMebase:WINDow:RANGe" on page 1050

:TIMebase:VERNier

(see page 1354)

Command Syntax

```
:TIMebase:VERNier <vernier value>

<vernier value> ::= {{1 | ON} | {0 | OFF}
```

The :TIMebase:VERNier command specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

Query Syntax

```
:TIMebase:VERNier?
```

The :TIMebase:VERNier? query returns the current state of the time base control's vernier setting.

Return Format

```
<vernier value><NL>

<vernier value> ::= {0 | 1}
```

See Also

- "Introduction to :TIMebase Commands" on page 1040

:TIMebase:WINDow:POSition

(see page 1354)

Command Syntax

```
:TIMebase:WINDow:POSition <pos value>

<pos value> ::= time from the trigger event to the zoomed (delayed)
                view reference point in NR3 format
```

The :TIMebase:WINDow:POSition command sets the horizontal position in the zoomed (delayed) view of the main sweep. The main sweep range and the main sweep horizontal position determine the range for this command. The value for this command must keep the zoomed view window within the main sweep range.

Query Syntax

```
:TIMebase:WINDow:POSition?
```

The :TIMebase:WINDow:POSition? query returns the current horizontal window position setting in the zoomed view.

Return Format

```
<value><NL>

<value> ::= position value in seconds
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:MODE" on page 1041
- ":TIMebase:POSition" on page 1042
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:SCALe" on page 1047
- ":TIMebase:WINDow:RANGe" on page 1050
- ":TIMebase:WINDow:SCALe" on page 1051

:TIMebase:WINDow:RANGe

(see page 1354)

Command Syntax

```
:TIMebase:WINDow:RANGe <range value>

<range value> ::= range value in seconds in NR3 format
```

The :TIMebase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the zoomed (delayed) window. The range is 10 times the current zoomed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMebase:RANGe value.

Query Syntax

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current window timebase range setting.

Return Format

```
<value><NL>

<value> ::= range value in seconds
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:POSition" on page 1042
- ":TIMebase:SCALe" on page 1047

:TIMebase:WINDow:SCALe

(see page 1354)

Command Syntax

```
:TIMebase:WINDow:SCALe <scale_value>

<scale_value> ::= scale value in seconds in NR3 format
```

The :TIMebase:WINDow:SCALe command sets the zoomed (delayed) window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMebase:SCALe value.

Query Syntax

```
:TIMebase:WINDow:SCALe?
```

The :TIMebase:WINDow:SCALe? query returns the current zoomed window scale setting.

Return Format

```
<scale_value><NL>

<scale_value> ::= current seconds per division for the zoomed window
```

See Also

- "Introduction to :TIMebase Commands" on page 1040
- ":TIMebase:RANGe" on page 1043
- ":TIMebase:POSition" on page 1042
- ":TIMebase:SCALe" on page 1047
- ":TIMebase:WINDow:RANGe" on page 1050

31 :TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

Introduction to :TRIGger Commands

The commands in the TRIGger subsystem define the conditions for an internal trigger. Many of these commands are valid in multiple trigger modes.

The default trigger mode is :EDGE.

The trigger subsystem controls the trigger sweep mode and the trigger specification. The trigger sweep (see ":TRIGger:SWEep" on page 1068) can be AUTO or NORMal.

- **NORMal** mode – displays a waveform only if a trigger signal is present and the trigger conditions are met. Otherwise the oscilloscope does not trigger and the display is not updated. This mode is useful for low-repetitive-rate signals.
- **AUTO** trigger mode – generates an artificial trigger event if the trigger specification is not satisfied within a preset time, acquires unsynchronized data and displays it.

AUTO mode is useful for signals other than low-repetitive-rate signals. You must use this mode to display a DC signal because there are no edges on which to trigger.

The following trigger types are available (see ":TRIGger:MODE" on page 1066).

- **Edge triggering**— identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Nth Edge Burst triggering**— lets you trigger on the Nth edge of a burst that occurs after an idle time.
- **Pulse width triggering**— (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering**— identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels. You can also trigger on a specified time duration of a pattern.
- **TV triggering**— is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than ¼ division of sync amplitude with any analog channel as the trigger source.

Reporting the Setup

Use :TRIGger? to query setup information for the TRIGger subsystem.

Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a *RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC;
:TRIG:ZONE:STAT 0
```

General :TRIGger Commands

Table 129 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 1057)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 1058)	:TRIGger:HFReject? (see page 1058)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 1059)	:TRIGger:HOLDoff? (see page 1059)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:HOLDoff:MAXimum <max_holdoff> (see page 1060)	:TRIGger:HOLDoff:MAXimum? (see page 1060)	<max_holdoff> ::= maximum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:MINimum <min_holdoff> (see page 1061)	:TRIGger:HOLDoff:MINimum? (see page 1061)	<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}} (see page 1062)	:TRIGger:HOLDoff:RANDom? (see page 1062)	<setting> ::= {0 | 1}
:TRIGger:LEVel:ASETup (see page 1063)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 1064)	:TRIGger:LEVel:HIGH? <source> (see page 1064)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see page 1065)	:TRIGger:LEVel:LOW? <source> (see page 1065)	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see page 1066)	:TRIGger:MODE? (see page 1066)	<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2} | NFC}

Table 129 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 1067)	:TRIGger:NREJect? (see page 1067)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 1068)	:TRIGger:SWEep? (see page 1068)	<sweep> ::= {AUTO | NORMal}

:TRIGger:FORCe

 (see page 1354)

Command Syntax `:TRIGger:FORCe`

The :TRIGger:FORCe command causes an acquisition to be captured even though the trigger condition has not been met. This command is equivalent to the front panel **[Force Trigger]** key.

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:HFReject

(see page 1354)

Command Syntax

```
:TRIGger:HFReject <value>

<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:HFReject command turns the high frequency reject filter off and on. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use this filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.

Query Syntax

```
:TRIGger:HFReject?
```

The :TRIGger:HFReject? query returns the current high frequency reject filter mode.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger[:EDGE]:REJect" on page 1084

:TRIGger:HOLDoff

(see page 1354)

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>

<holdoff_time> ::= 40 ns to 10 s in NR3 format
```

The :TRIGger:HOLDoff command defines the holdoff time value in seconds. Holdoff keeps a trigger from occurring until after a certain amount of time has passed since the last trigger. This feature is valuable when a waveform crosses the trigger level multiple times during one period of the waveform. Without holdoff, the oscilloscope could trigger on each of the crossings, producing a confusing waveform. With holdoff set correctly, the oscilloscope always triggers on the same crossing. The correct holdoff setting is typically slightly less than one period.

Query Syntax

```
:TRIGger:HOLDoff?
```

The :TRIGger:HOLDoff? query returns the holdoff time value for the current trigger mode.

Return Format

```
<holdoff_time><NL>

<holdoff_time> ::= the holdoff time value in seconds in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:HOLDoff:MAXimum

(see page 1354)

Command Syntax

```
:TRIGger:HOLDoff:MAXimum <max_holdoff>

<max_holdoff> ::= maximum holdoff time in seconds in NR3 format
```

When the random trigger holdoff mode is enabled (see :TRIGger:HOLDoff:RANDom), the :TRIGger:HOLDoff:MAXimum command specifies the maximum trigger holdoff time.

Query Syntax

```
:TRIGger:HOLDoff:MAXimum?
```

The :TRIGger:HOLDoff:MAXimum? query returns the maximum random trigger holdoff time setting.

Return Format

```
<max_holdoff><NL>
```

See Also

- ":TRIGger:HOLDoff:MINimum" on page 1061
- ":TRIGger:HOLDoff:RANDom" on page 1062

:TRIGger:HOLDoff:MINimum

(see page 1354)

Command Syntax

```
:TRIGger:HOLDoff:MINimum <min_holdoff>

<min_holdoff> ::= minimum holdoff time in seconds in NR3 format
```

When the random trigger holdoff mode is enabled (see :TRIGger:HOLDoff:RANDom), the :TRIGger:HOLDoff:MINimum command specifies the minimum trigger holdoff time.

Query Syntax

```
:TRIGger:HOLDoff:MINimum?
```

The :TRIGger:HOLDoff:MINimum? query returns the minimum random trigger holdoff time setting.

Return Format

```
<min_holdoff><NL>
```

See Also

- ":TRIGger:HOLDoff:MAXimum" on page 1060
- ":TRIGger:HOLDoff:RANDom" on page 1062

:TRIGger:HOLDoff:RANDom

(see page 1354)

Command Syntax

```
:TRIGger:HOLDoff:RANDom {{0 | OFF} | {1 | ON}}
```

The :TRIGger:HOLDoff:RANDom command enables or disables the random trigger holdoff mode. This mode randomizes the holdoff time from one acquisition to the next. The randomized holdoff time values will be between the values specified by the :TRIGger:HOLDoff:MINimum and :TRIGger:HOLDoff:MAXimum commands.

The random trigger holdoff mode ensures that the oscilloscope re-arms after each acquisition in a manner that minimizes or eliminates the likelihood of triggering at the beginning of a DDR burst. Randomizing the holdoff time increases the likelihood that the oscilloscope will trigger on different data phases of a multi-phase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

Query Syntax

```
:TRIGger:HOLDoff:RANDom?
```

The :TRIGger:HOLDoff:RANDom? query returns random trigger holdoff mode setting.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- ":TRIGger:HOLDoff:MAXimum" on page 1060
- ":TRIGger:HOLDoff:MINimum" on page 1061

:TRIGger:LEVel:ASETup

 (see page 1354)

Command Syntax `:TRIGger:LEVel:ASETup`

The :TRIGger:LEVel:ASETup command automatically sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

If AC coupling is used, the trigger levels are set to 0 V.

When High and Low (dual) trigger levels are used (as with Rise/Fall Time and Runt triggers, for example), this command has no effect.

See Also

- ":TRIGger[:EDGE]:LEVel" on page 1083

:TRIGger:LEVel:HIGH

(see page 1354)

Command Syntax

```
:TRIGger:LEVel:HIGH <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:LEVel:HIGH command sets the high trigger voltage level voltage for the specified source.

High and low trigger levels are used with runt triggers and rise/fall time (transition) triggers.

Query Syntax

```
:TRIGger:LEVel:HIGH? <source>
```

The :TRIGger:LEVel:HIGH? query returns the high trigger voltage level for the specified source.

Return Format

```
<level><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:LEVel:LOW" on page 1065
- ":TRIGger:RUNT Commands" on page 1124
- ":TRIGger:TRANsition Commands" on page 1135
- ":TRIGger[:EDGE]:SOURce" on page 1086

:TRIGger:LEVel:LOW

(see page 1354)

Command Syntax

```
:TRIGger:LEVel:LOW <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:LEVel:LOW command sets the low trigger voltage level voltage for the specified source.

High and low trigger levels are used with runt triggers and rise/fall time (transition) triggers.

Query Syntax

```
:TRIGger:LEVel:LOW? <source>
```

The :TRIGger:LEVel:LOW? query returns the low trigger voltage level for the specified source.

Return Format

```
<level><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:LEVel:HIGH" on page 1064
- ":TRIGger:RUNT Commands" on page 1124
- ":TRIGger:TRANsition Commands" on page 1135
- ":TRIGger[:EDGE]:SOURce" on page 1086

:TRIGger:MODE

(see page 1354)

Command Syntax

```
:TRIGger:MODE <mode>

<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT
           | SHOLd | TRANsition | SBUS{1 | 2} | NFC}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode.

If the :TIMebase:MODE is ROLL or XY, the query returns "NONE".

If the PXIe oscilloscope module is set up as a slave module in PXI triggering (see :TRIGger:PXI:MODE), the query returns "PXIS".

Return Format

```
<mode><NL>

<mode> ::= {EDGE | GLIT | PATT | TV | DEL | EBUR | OR | RUNT | SHOL
           | TRAN | SBUS{1 | 2} | NFC | PXIS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:SWEep" on page 1068
- ":TIMebase:MODE" on page 1041

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TRIGger:NREJect

(see page 1354)

Command Syntax

```
:TRIGger:NREJect <value>

<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:NREJect command turns the noise reject filter off and on. When the noise reject filter is on, the trigger circuitry is less sensitive to noise but may require a greater amplitude waveform to trigger the oscilloscope. This command is not valid in TV trigger mode.

Query Syntax

```
:TRIGger:NREJect?
```

The :TRIGger:NREJect? query returns the current noise reject filter mode.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:SWEep

(see page 1354)

Command Syntax

```
:TRIGger:SWEep <sweep>

<sweep> ::= {AUTO | NORMal}
```

The :TRIGger:SWEep command selects the trigger sweep mode.

When AUTO sweep mode is selected, a baseline is displayed in the absence of a signal. If a signal is present but the oscilloscope is not triggered, the unsynchronized signal is displayed instead of a baseline.

When NORMal sweep mode is selected and no trigger is present, the instrument does not sweep, and the data acquired on the previous trigger remains on the screen.

NOTE This feature is called "Mode" on the instrument's front panel.

Query Syntax

```
:TRIGger:SWEep?
```

The :TRIGger:SWEep? query returns the current trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= current trigger sweep mode
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:DELay Commands

Table 130 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SLOPe <slope> (see page 1070)	:TRIGger:DELay:ARM:SLOPe? (see page 1070)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:ARM:SOURce <source> (see page 1071)	:TRIGger:DELay:ARM:SOURce? (see page 1071)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:DELay:TDELay:TIME <time_value> (see page 1072)	:TRIGger:DELay:TDELay:TIME? (see page 1072)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGger:COUNt <count> (see page 1073)	:TRIGger:DELay:TRIGger:COUNt? (see page 1073)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGger:SLOPe <slope> (see page 1074)	:TRIGger:DELay:TRIGger:SLOPe? (see page 1074)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:TRIGger:SOURce <source> (see page 1075)	:TRIGger:DELay:TRIGger:SOURce? (see page 1075)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

The :TRIGger:DELay:ARM:SOURce and :TRIGger:DELay:TRIGger:SOURce commands are used to specify the source channel for the arming edge and the trigger edge in the Edge Then Edge trigger.

If an analog channel is selected as a source, the :TRIGger:EDGE:LEVel command is used to set the trigger level.

:TRIGger:DELay:ARM:SLOPe

N (see page 1354)

Command Syntax

```
:TRIGger:DELay:ARM:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:DELay:ARM:SLOPe command specifies rising (POSitive) or falling (NEGative) for the arming edge in the Edge Then Edge trigger.

Query Syntax

```
:TRIGger:DELay:ARM:SLOPe?
```

The :TRIGger:DELay:ARM:SLOPe? query returns the current arming edge slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:ARM:SOURce" on page 1071
- ":TRIGger:DELay:TDELay:TIME" on page 1072

:TRIGger:DELay:ARM:SOURce

(see page 1354)

Command Syntax

```
:TRIGger:DELay:ARM:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:DELay:ARM:SOURce command selects the input used for the arming edge in the Edge Then Edge trigger.

Query Syntax

```
:TRIGger:DELay:ARM:SOURce?
```

The :TRIGger:DELay:ARM:SOURce? query returns the current arming edge source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:ARM:SLOPe" on page 1070
- ":TRIGger:DELay:TDELay:TIME" on page 1072
- ":TRIGger:MODE" on page 1066

:TRIGger:DELay:TDELay:TIME

(see page 1354)

Command Syntax

```
:TRIGger:DELay:TDELay:TIME <time_value>

<time_value> ::= time in seconds in NR3 format
```

The :TRIGger:DELay:TDELay:TIME command sets the delay time between the arming edge and the trigger edge in the Edge Then Edge trigger. The time is in seconds and must be from 4 ns to 10 s.

Query Syntax

```
:TRIGger:DELay:TDELay:TIME?
```

The :TRIGger:DELay:TDELay:TIME? query returns current delay time setting.

Return Format

```
<time value><NL>

<time_value> ::= time in seconds in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:TRIGger:SLOPe" on page 1074
- ":TRIGger:DELay:TRIGger:COUNt" on page 1073

:TRIGger:DELay:TRIGger:COUNt

N (see page 1354)

Command Syntax

```
:TRIGger:DELay:TRIGger:COUNt <count>

<count> ::= integer in NR1 format
```

The :TRIGger:DELay:TRIGger:COUNt command sets the Nth edge of the trigger source to trigger on.

Query Syntax

```
:TRIGger:DELay:TRIGger:COUNt?
```

The :TRIGger:DELay:TRIGger:COUNt? query returns the current Nth trigger edge setting.

Return Format

```
<count><NL>

<count> ::= integer in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:TRIGger:SLOPe" on page 1074
- ":TRIGger:DELay:TRIGger:SOURce" on page 1075
- ":TRIGger:DELay:TDELay:TIME" on page 1072

:TRIGger:DELay:TRIGger:SLOPe

(see page 1354)

Command Syntax

```
:TRIGger:DELay:TRIGger:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:DELay:TRIGger:SLOPe command specifies rising (POSitive) or falling (NEGative) for the trigger edge in the Edge Then Edge trigger.

Query Syntax

```
:TRIGger:DELay:TRIGger:SLOPe?
```

The :TRIGger:DELay:TRIGger:SLOPe? query returns the current trigger edge slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:TRIGger:SOURce" on page 1075
- ":TRIGger:DELay:TDELay:TIME" on page 1072
- ":TRIGger:DELay:TRIGger:COUNt" on page 1073

:TRIGger:DELay:TRIGger:SOURce

N (see page 1354)

Command Syntax

```
:TRIGger:DELay:TRIGger:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:DELay:TRIGger:SOURce command selects the input used for the trigger edge in the Edge Then Edge trigger.

Query Syntax

```
:TRIGger:DELay:TRIGger:SOURce?
```

The :TRIGger:DELay:TRIGger:SOURce? query returns the current trigger edge source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:DELay:TRIGger:SLOPe" on page 1074
- ":TRIGger:DELay:TDELay:TIME" on page 1072
- ":TRIGger:DELay:TRIGger:COUNt" on page 1073
- ":TRIGger:MODE" on page 1066

:TRIGger:EBURst Commands

Table 131 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNt <count> (see page 1077)	:TRIGger:EBURst:COUNt? (see page 1077)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 1078)	:TRIGger:EBURst:IDLE? (see page 1078)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 1079)	:TRIGger:EBURst:SLOPe? (see page 1079)	<slope> ::= {NEGative | POSitive}
:TRIGger:EBURst:SOURce <source> (see page 1080)	:TRIGger:EBURst:SOURce? (see page 1080)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

The :TRIGger:EBURst:SOURce command is used to specify the source channel for the Nth Edge Burst trigger. If an analog channel is selected as the source, the :TRIGger:EDGE:LEVel command is used to set the Nth Edge Burst trigger level.

:TRIGger:EBURst:COUNt

N (see page 1354)

Command Syntax

```
:TRIGger:EBURst:COUNt <count>

<count> ::= integer in NR1 format
```

The :TRIGger:EBURst:COUNt command sets the Nth edge at burst counter resource. The edge counter is used in the trigger stage to determine which edge in a burst will generate a trigger.

Query Syntax

```
:TRIGger:EBURst:COUNt?
```

The :TRIGger:EBURst:COUNt? query returns the current Nth edge of burst edge counter setting.

Return Format

```
<count><NL>

<count> ::= integer in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:EBURst:SLOPe" on page 1079
- ":TRIGger:EBURst:IDLE" on page 1078

:TRIGger:EBURst:IDLE

(see page 1354)

Command Syntax

```
:TRIGger:EBURst:IDLE <time_value>

<time_value> ::= time in seconds in NR3 format
```

The :TRIGger:EBURst:IDLE command sets the Nth edge in a burst idle resource in seconds from 10 ns to 10 s. The timer is used to set the minimum time before the next burst.

Query Syntax

```
:TRIGger:EBURst:IDLE?
```

The :TRIGger:EBURst:IDLE? query returns current Nth edge in a burst idle setting.

Return Format

```
<time value><NL>

<time_value> ::= time in seconds in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:EBURst:SLOPe" on page 1079
- ":TRIGger:EBURst:COUNt" on page 1077

:TRIGger:EBURst:SLOPe

(see page 1354)

Command Syntax

```
:TRIGger:EBURst:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:EBURst:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.

Query Syntax

```
:TRIGger:EBURst:SLOPe?
```

The :TRIGger:EBURst:SLOPe? query returns the current Nth edge in a burst slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:EBURst:IDLE" on page 1078
- ":TRIGger:EBURst:COUNt" on page 1077

:TRIGger:EBURst:SOURce

(see page 1354)

Command Syntax

```
:TRIGger:EBURst:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:EBURst:SOURce command selects the input that produces the Nth edge burst trigger.

Query Syntax

```
:TRIGger:EBURst:SOURce?
```

The :TRIGger:EBURst:SOURce? query returns the current Nth edge burst trigger source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066

:TRIGger[:EDGE] Commands

Table 132 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 1082)	:TRIGger[:EDGE]:COUPling? (see page 1082)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 1083)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 1083)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 1084)	:TRIGger[:EDGE]:REJect? (see page 1084)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 1085)	:TRIGger[:EDGE]:SLOPe? (see page 1085)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 1086)	:TRIGger[:EDGE]:SOURce? (see page 1086)	<source> ::= {CHANnel<n> | EXTernal | WGEN | WGEN1 | WMOD} <n> ::= 1 to (# analog channels) in NR1 format Note: WGEN and WGEN1 are equivalent.

:TRIGger[:EDGE]:COUPling

 (see page 1354)

Command Syntax

```
:TRIGger[:EDGE]:COUPling <coupling>

<coupling> ::= {AC | DC | LFReject}
```

The :TRIGger[:EDGE]:COUPling command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LFReject.

- AC coupling places a high-pass filter (10 Hz for analog channels, and 3.5 Hz for all External trigger inputs) in the trigger path, removing dc offset voltage from the trigger waveform. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset.
- LFReject coupling places a 50 KHz high-pass filter in the trigger path.
- DC coupling allows dc and ac signals into the trigger path.

NOTE **The :TRIGger[:EDGE]:COUPling and the :TRIGger[:EDGE]:REJect selections are coupled. Changing the setting of the :TRIGger[:EDGE]:REJect can change the COUPling setting.**

Query Syntax

```
:TRIGger[:EDGE]:COUPling?
```

The :TRIGger[:EDGE]:COUPling? query returns the current coupling selection.

Return Format

```
<coupling><NL>

<coupling> ::= {AC | DC | LFR}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger[:EDGE]:REJect" on page 1084

:TRIGger[:EDGE]:LEVel

(see page 1354)

Command Syntax

```
:TRIGger[:EDGE]:LEVel <level>

<level> ::= <level>[,<source>]

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= ±(external range setting) in NR3 format
            for external triggers

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

```
:TRIGger[:EDGE]:LEVel? [<source>]
```

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

```
<level><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger[:EDGE]:SOURce" on page 1086
- ":EXTernal:RANGe" on page 342

:TRIGger[:EDGE]:REJect

(see page 1354)

Command Syntax

```
:TRIGger[:EDGE]:REJect <reject>

<reject> ::= {OFF | LFReject | HFReject}
```

The :TRIGger[:EDGE]:REJect command turns the low-frequency or high-frequency reject filter on or off. You can turn on one of these filters at a time.

- The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use the high frequency reject filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.
- The low frequency reject filter adds a 50 kHz high-pass filter in series with the trigger waveform to remove any unwanted low frequency components from a trigger waveform, such as power line frequencies, that can interfere with proper triggering.

NOTE **The :TRIGger[:EDGE]:REJect and the :TRIGger[:EDGE]:COUPling selections are coupled. Changing the setting of the :TRIGger[:EDGE]:COUPling can change the COUPling setting.**

Query Syntax

```
:TRIGger[:EDGE]:REJect?
```

The :TRIGger[:EDGE]:REJect? query returns the current status of the reject filter.

Return Format

```
<reject><NL>

<reject> ::= {OFF | LFR | HFR}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:HFReject" on page 1058
- ":TRIGger[:EDGE]:COUPling" on page 1082

:TRIGger[:EDGE]:SLOPe

(see page 1354)

Command Syntax

```
:TRIGger[:EDGE]:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer | ALTernate}
```

The :TRIGger[:EDGE]:SLOPe command specifies the slope of the edge for the trigger. The SLOPe command is not valid in TV trigger mode. Instead, use :TRIGger:TV:POLarity to set the polarity in TV trigger mode.

Query Syntax

```
:TRIGger[:EDGE]:SLOPe?
```

The :TRIGger[:EDGE]:SLOPe? query returns the current trigger slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS | EITH | ALT}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:TV:POLarity" on page 1143

Example Code

```
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TRIGger[:EDGE]:SOURce

(see page 1354)

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal | WGEN | WGEN1 | WMOD}

<n> ::= 1 to (# analog channels) in NR1 format

Note: WAVE and WGEN1 are equivalent.
```

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTernal – triggers on the rear panel EXT TRIG IN signal.
- WGEN, WGEN1 – triggers at the 50% level of the rising edge of the waveform generator output signal. This option is not available when the DC, NOISe, or CARDiac waveforms are selected.
- WMOD – when waveform generator FSK or FM modulation is used, triggers at the 50% level of the rising edge of the modulating signal.

Query Syntax

```
:TRIGger[:EDGE]:SOURce?
```

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | EXT | WGEN | WGEN1 | WMOD | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066

Example Code

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces th
e
' edge trigger.  Any channel can be selected.
myScope.WriteString ":TRIGger:EDGE:SOURce CHANnel1"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TRIGger:GLITch Commands

Table 133 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 1088)	:TRIGger:GLITch:GREaterthan? (see page 1088)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 1089)	:TRIGger:GLITch:LESSthan? (see page 1089)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 1090)	:TRIGger:GLITch:LEVel? (see page 1090)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 1091)	:TRIGger:GLITch:POLarity? (see page 1091)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 1092)	:TRIGger:GLITch:QUALifier? (see page 1092)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1093)	:TRIGger:GLITch:RANGe? (see page 1093)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 1094)	:TRIGger:GLITch:SOURce? (see page 1094)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:TRIGger:GLITch:GREaterthan

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= floating-point number in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:GREaterthan?
```

The :TRIGger:GLITch:GREaterthan? query returns the minimum pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<greater_than_time><NL>

<greater_than_time> ::= floating-point number in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:GLITch:SOURce" on page 1094
- ":TRIGger:GLITch:QUALifier" on page 1092
- ":TRIGger:MODE" on page 1066

:TRIGger:GLITch:LESSthan

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= floating-point number in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:LESSthan command sets the maximum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:LESSthan?
```

The :TRIGger:GLITch:LESSthan? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<less_than_time><NL>

<less_than_time> ::= floating-point number in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:GLITch:SOURce" on page 1094
- ":TRIGger:GLITch:QUALifier" on page 1092
- ":TRIGger:MODE" on page 1066

:TRIGger:GLITch:LEVel

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:LEVel <level>[,<source>]

<level> ::= .75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= ±(external range setting) in NR3 format
            for external triggers

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:LEVel?
```

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

Return Format

```
<level><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:GLITch:SOURce" on page 1094
- ":EXTernal:RANGe" on page 342

:TRIGger:GLITch:POLarity

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:GLITch:POLarity command sets the polarity for the glitch pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:POLarity?
```

The :TRIGger:GLITch:POLarity? query returns the glitch pulse width trigger polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:GLITch:SOURce" on page 1094

:TRIGger:GLITch:QUALifier

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:QUALifier <operator>

<operator> ::= {GREaterthan | LESSthan | RANGe}
```

This command sets the mode of operation of the glitch pulse width trigger. The oscilloscope can trigger on a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax

```
:TRIGger:GLITch:QUALifier?
```

The :TRIGger:GLITch:QUALifier? query returns the glitch pulse width qualifier.

Return Format

```
<operator><NL>

<operator> ::= {GRE | LESS | RANG}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:GLITch:SOURce" on page 1094
- ":TRIGger:MODE" on page 1066

:TRIGger:GLITch:RANGe

(see page 1354)

Command Syntax

```
:TRIGger:GLITch:RANGe <less_than_time>[suffix],
                      <greater_than_time>[suffix]

<less_than_time> ::= (15 ns - 10 seconds) in NR3 format

<greater_than_time> ::= (10 ns - 9.99 seconds) in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:RANGe command sets the pulse width duration for the selected :TRIGger:GLITch:SOURce. You can enter the parameters in any order — the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:TRIGger:GLITch:RANGe?
```

The :TRIGger:GLITch:RANGe? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:GLITch:SOURce" on page 1094
- ":TRIGger:GLITch:QUALifier" on page 1092
- ":TRIGger:MODE" on page 1066

:TRIGger:GLITch:SOURce

N (see page 1354)

Command Syntax

```
:TRIGger:GLITch:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:SOURce?
```

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE".

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:GLITch:LEVel" on page 1090
- ":TRIGger:GLITch:POLarity" on page 1091
- ":TRIGger:GLITch:QUALifier" on page 1092
- ":TRIGger:GLITch:RANGe" on page 1093

Example Code

- "Example Code" on page 1086

:TRIGger:NFC Commands

NFC (Near Field Communication) triggering is used to capture waveforms used in NFC testing. The NFC trigger mode is license-enabled.

Table 134 :TRIGger:NFC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:NFC:AEVent <arm_event> (see page 1096)	:TRIGger:NFC:AEVent? (see page 1096)	<arm_event> ::= {NONE | ASReq | AALLreq | AEITher | BSReq | BALLreq | BEITher | FSReq}
n/a	:TRIGger:NFC:ATTime? (see page 1097)	<time> ::= seconds in NR3 format
:TRIGger:NFC:RPOLarity {{0 | OFF} | {1 | ON}} (see page 1098)	:TRIGger:NFC:RPOLarity? (see page 1098)	{0 | 1}
:TRIGger:NFC:SOURce <source> (see page 1099)	:TRIGger:NFC:SOURce? (see page 1099)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:NFC:STANdard <standard> (see page 1100)	:TRIGger:NFC:STANdard? (see page 1100)	<standard> ::= {{A | A106} | {B | B106} | F212 | F424}
:TRIGger:NFC:TEVent <trigger_event> (see page 1101)	:TRIGger:NFC:TEVent? (see page 1101)	<trigger_event> ::= {ATRigger | ASReq | AALLreq | AEITher | ASDDreq | BSReq | BALLreq | BEITher | BATTrib | FSReq | FAReq | FPReamble}
n/a	:TRIGger:NFC:TIMeout? (see page 1103)	{0 | 1}
:TRIGger:NFC:TIMeout:ENABle {{0 | OFF} | {1 | ON}} (see page 1104)	:TRIGger:NFC:TIMeout:ENABle? (see page 1104)	{0 | 1}
:TRIGger:NFC:TIMeout:TIME <time> (see page 1105)	:TRIGger:NFC:TIMeout:TIME? (see page 1105)	<time> ::= seconds in NR3 format

:TRIGger:NFC:AEVent

N (see page 1354)

Command Syntax

```
:TRIGger:NFC:AEVent <arm_event>

<arm_event> ::= {NONE | ASReq | AALLreq | AEITher | BSReq | BALLreq
   | BEITher | FSReq}
```

When the ATRigger (Arm & Trigger) trigger event is selected (by :TRIGger:NFC:TEVent), the :TRIGger:NFC:AEVent command specifies the arm event. Valid arm event settings depend on the signaling technology selected (by :TRIGger:NFC:STANdard):

Signaling Technology	Arm Event	Description
NFC-A	ASReq	SENS_REQ
	AALLreq	ALL_REQ
	AEITher	Either the SENS_REQ or ALL_REQ events will arm.
NFC-B	BSReq	SENSB_REQ
	BALLreq	ALLB_REQ
	BEITher	Either the SENSB_REQ or ALLB_REQ events will arm.
NFC-F	FSReq	SENSF_REQ

When a trigger event other than ATRigger (Arm & Trigger) is selected, the arm event is NONE.

Query Syntax

```
:TRIGger:NFC:AEVent?
```

The :TRIGger:NFC:AEVent? query returns the specified arm event.

Return Format

```
<arm_event><NL>

<arm_event> ::= {NONE | ASR | AALL | AEIT | BSR | BALL | BEIT | FSR}
```

See Also

- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:ATTime

(see page 1354)

Query Syntax `:TRIGger:NFC:ATTime?`

The :TRIGger:NFC:ATTime? query returns the time between the arm and the trigger when the ATRigger (Arm & Trigger) trigger event is selected.

Return Format `<time><NL>`

`<time> ::= seconds in NR3 format`

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:RPOLarity

N (see page 1354)

Command Syntax `:TRIGger:NFC:RPOLarity {{0 | OFF} | {1 | ON}}`

The :TRIGger:NFC:RPOLarity ON command enables the oscilloscope to trigger on signals with "reverse" polarity.

When OFF, the oscilloscope will trigger on signals with "obverse" polarity.

Query Syntax `:TRIGger:NFC:RPOLarity?`

The :TRIGger:NFC:RPOLarity? query returns the "reverse" polarity setting.

Return Format `<setting><NL>`

`<setting> ::= {0 | 1}`

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:SOURce

N (see page 1354)

Command Syntax

```
:TRIGger:NFC:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:NFC:SOURce command selects the input waveform source for the NFC trigger. You can choose an analog input channel.

Query Syntax

```
:TRIGger:NFC:SOURce?
```

The :TRIGger:NFC:SOURce? query returns the input waveform source setting.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:STANdard

N (see page 1354)

Command Syntax

```
:TRIGger:NFC:STANdard <standard>

<standard> ::= {{A | A106} | {B | B106} | F212 | F424}
```

The :TRIGger:NFC:STANdard command selects the signaling technology used by the input signal:

- A or A106 – NFC-A standard, 106 kbits/s.
- B or B106 – NFC-A standard, 106 kbits/s.
- F212 – NFC-F standard, 212 kbits/s.
- F424 – NFC-F standard, 424 kbits/s.

Query Syntax

```
:TRIGger:NFC:STANdard?
```

The :TRIGger:NFC:STANdard? query returns the signaling technology setting.

Return Format

```
<standard><NL>

<standard> ::= {{A | A106} | {B | B106} | F212 | F424}
```

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:TEVent

N (see page 1354)

Command Syntax

```
:TRIGger:NFC:TEVent <trigger_event>

<trigger_event> ::= {ATRigger | ASReq | AALLreq | AEITher | ASDDreq
   | BSReq | BALLreq | BEITher | BATTrib | FSReq | FAReq | FPReamble}
```

The :TRIGger:NFC:TEVent command specifies the trigger event. Valid trigger event settings depend on the signaling technology selected (by :TRIGger:NFC:STANdard):

Signaling Technology	Trigger Event	Description
NFC-A	ATRigger	The arm event is specified by :TRIGger:NFC:AEVent, and the trigger event is SDD_REQ.
	ASReq	SENS_REQ
	AALLreq	ALL_REQ
	AEITher	Either the SENS_REQ or ALL_REQ events will cause a trigger.
	ASDDreq	SDD_REQ
NFC-B	ATRigger	The arm event is specified by :TRIGger:NFC:AEVent, and the trigger event is ATTRIB.
	BSReq	SENSB_REQ
	BALLreq	ALLB_REQ
	BEITher	Either the SENSB_REQ or ALLB_REQ events will cause a trigger.
	BATTrib	ATTRIB
NFC-F	ATRigger	The arm event is specified by :TRIGger:NFC:AEVent, and the trigger event is ATR_REQ.
	FSReq	SENSF_REQ
	FAReq	ATR_REQ
	FPReamble	The preamble sequence that begins a data frame will cause a trigger.

Query Syntax

```
:TRIGger:NFC:TEVent?
```

The :TRIGger:NFC:TEVent? query returns the specified trigger event.

Return Format

```
<trigger_event><NL>

<trigger_event> ::= {ATR | ASR | AALL | AEIT | ASDD | BSR | BALL
   | BEIT | BATT | FSR | FAR | FPR}
```

See Also
- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:TIMeout

(see page 1354)

Query Syntax `:TRIGger:NFC:TIMeout?`

The :TRIGger:NFC:TIMeout? query returns whether the timeout occurred.

With the ATRigger (Arm & Trigger) trigger event, if the second event does not occur within the timeout period, the oscilloscope triggers when the timeout period expires.

A return value of 1 says the desired trigger event did not occur within the specified time after the arm event.

A return value of 0 says the desired trigger event occurred before the timeout.

Return Format `<timeout_occurred><NL>`

`{0 | 1}`

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:TIMeout:ENABle

(see page 1354)

Command Syntax

```
:TRIGger:NFC:TIMeout:ENABle {1 | ON}
```

The :TRIGger:NFC:TIMeout:ENABle command enables the timeout period. Currently, ON is the only valid setting.

With the ATRigger (Arm & Trigger) trigger event, if the second event does not occur within the timeout period, the oscilloscope triggers when the timeout period expires.

Query Syntax

```
:TRIGger:NFC:TIMeout:ENABle?
```

The :TRIGger:NFC:TIMeout:ENABle? query returns the timeout period enable setting.

Return Format

```
<setting><NL>

{1}
```

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:TIME" on page 1105

:TRIGger:NFC:TIMeout:TIME

 (see page 1354)

Command Syntax

```
:TRIGger:NFC:TIMeout:TIME <time>

<time> ::= seconds in NR3 format
```

The :TRIGger:NFC:TIMeout:TIME command specifies the timeout period. With the ATRigger (Arm & Trigger) trigger event, if the second event does not occur within the timeout period, the oscilloscope triggers when the timeout period expires.

Query Syntax

```
:TRIGger:NFC:TIMeout:TIME?
```

The :TRIGger:NFC:TIMeout:TIME? query returns the specified timeout period.

Return Format

```
<time><NL>

<time> ::= seconds in NR3 format
```

See Also

- ":TRIGger:NFC:AEVent" on page 1096
- ":TRIGger:NFC:ATTime" on page 1097
- ":TRIGger:NFC:RPOLarity" on page 1098
- ":TRIGger:NFC:SOURce" on page 1099
- ":TRIGger:NFC:STANdard" on page 1100
- ":TRIGger:NFC:TEVent" on page 1101
- ":TRIGger:NFC:TIMeout" on page 1103
- ":TRIGger:NFC:TIMeout:ENABle" on page 1104

:TRIGger:OR Commands

Table 135 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1107)	:TRIGger:OR? (see page 1107)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog channel as shown on the soft front panel display.

:TRIGger:OR

(see page 1354)

Command Syntax

```
:TRIGger:OR <string>

<string> ::= "nn...n" where n ::= {R | F | E | X}

R = rising edge, F = falling edge, E = either edge, X = don't care.
```

The :TRIGger:OR command specifies the edges to include in the OR'ed edge trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
2 analog channels only	Bits 0 and 1 - analog channels 2 and 1.

Query Syntax

```
:TRIGger:OR?
```

The :TRIGger:OR? query returns the current OR'ed edge trigger string.

Return Format

```
<string><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066

:TRIGger:PATTern Commands

Table 136 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 1109)	:TRIGger:PATTern? (see page 1109)	<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when <base> = ASCii <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when <base> = HEX <edge_source> ::= {CHANnel<n> | NONE} <n> ::= 1 to (# analog channels) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 1111)	:TRIGger:PATTern:FORMat? (see page 1111)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 1112)	:TRIGger:PATTern:GREaterthan? (see page 1112)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 1113)	:TRIGger:PATTern:LESSthan? (see page 1113)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:QUALifier <qualifier> (see page 1114)	:TRIGger:PATTern:QUALifier? (see page 1114)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1115)	:TRIGger:PATTern:RANGe? (see page 1115)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:PATTern

(see page 1354)

Command Syntax

```
:TRIGger:PATTern <pattern>

<pattern> ::= <string>[,<edge_source>,<edge>]

<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when
             <base> = ASCii

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $} when
             <base> = HEX

<edge_source> ::= {CHANnel<n> | NONE}

<n> ::= 1 to (# of analog channels) in NR1 format

<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTern command specifies the channel values to be used in the pattern trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
2 analog channels only	Bits 0 and 1 - analog channels 2 and 1.

The format of the <string> parameter depends on the :TRIGger:PATTern:FORMat command setting:

- When the format is ASCii, the string looks just like the string you see on the oscilloscope's front panel, made up of 0, 1, X (don't care), R (rising edge), and F (falling edge) characters.
- When the format is HEX, the string begins with "0x" and contains hex digit characters or X (don't care for all four bits in the nibble).

  With the hex format string, you can use the <edge_source> and <edge> parameters to specify an edge on one of the channels.

NOTE

The optional <edge_source> and <edge> parameters should be sent together or not at all. The edge can be specified in the ASCII <string> parameter. If the edge source and edge parameters are used, they take precedence.

You can only specify an edge on one channel. When an edge is specified, the :TRIGger:PATTern:QUALifier does not apply.

Query Syntax

```
:TRIGger:PATTern?
```

The :TRIGger:PATTern? query returns the pattern string, edge source, and edge.

Return Format `<string>,<edge_source>,<edge><NL>`

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern:FORMat" on page 1111
- ":TRIGger:PATTern:QUALifier" on page 1114
- ":TRIGger:MODE" on page 1066

:TRIGger:PATTern:FORMat

(see page 1354)

Command Syntax

```
:TRIGger:PATTern:FORMat <base>

<base> ::= {ASCii | HEX}
```

The :TRIGger:PATTern:FORMat command sets the entry (and query) number base used by the :TRIGger:PATTern command. The default <base> is ASCii.

Query Syntax

```
:TRIGger:PATTern:FORMat?
```

The :TRIGger:PATTern:FORMat? query returns the currently set number base for pattern trigger patterns.

Return Format

```
<base><NL>

<base> ::= {ASC | HEX}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern" on page 1109

:TRIGger:PATTern:GREaterthan

(see page 1354)

Command Syntax

```
:TRIGger:PATTern:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= minimum trigger duration in seconds
                        in NR3 format

<suffix> ::= {s | ms | us | ns | ps }
```

The :TRIGger:PATTern:GREaterthan command sets the minimum duration for the defined pattern when :TRIGger:PATTern:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIGger:PATTern:QUALifier is set to TIMeout.

Query Syntax

```
:TRIGger:PATTern:GREaterthan?
```

The :TRIGger:PATTern:GREaterthan? query returns the minimum duration time for the defined pattern.

Return Format

```
<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern" on page 1109
- ":TRIGger:PATTern:QUALifier" on page 1114
- ":TRIGger:MODE" on page 1066

:TRIGger:PATTern:LESSthan

(see page 1354)

Command Syntax

```
:TRIGger:PATTern:LESSthan <less_than_time>[<suffix>]

<less_than_time> ::= maximum trigger duration in seconds
                     in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:PATTern:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:PATTern:QUALifier is set to LESSthan.

Query Syntax

```
:TRIGger:PATTern:LESSthan?
```

The :TRIGger:PATTern:LESSthan? query returns the duration time for the defined pattern.

Return Format

```
<less_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern" on page 1109
- ":TRIGger:PATTern:QUALifier" on page 1114
- ":TRIGger:MODE" on page 1066

:TRIGger:PATTern:QUALifier

(see page 1354)

Command Syntax

```
:TRIGger:PATTern:QUALifier <qualifier>

<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange
                 | TIMeout}
```

The :TRIGger:PATTern:QUALifier command qualifies when the trigger occurs:

- ENTered – when the pattern is entered.
- LESSthan – when the pattern is present for less than a time value.
- GREaterthan – when the pattern is present for greater than a time value. The trigger occurs when the pattern exits (not when the GREaterthan time value is exceeded).
- TIMeout – when the pattern is present for greater than a time value. In this case, the trigger occurs when the GREaterthan time value is exceeded (not when the pattern exits).
- INRange – when the pattern is present for a time within a range of values.
- OUTRange – when the pattern is present for a time outside of range of values.

Pattern durations are evaluated using a timer. The timer starts on the last edge that makes the pattern (logical AND) true. Except when the TIMeout qualifier is selected, the trigger occurs on the first edge that makes the pattern false, provided the time qualifier criteria has been met.

Set the GREaterthan qualifier value with the :TRIGger:PATTern:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:PATTern:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:PATTern:RANGe command.

Set the TIMeout qualifier value with the :TRIGger:PATTern:GREaterthan command.

Query Syntax

```
:TRIGger:PATTern:QUALifier?
```

The :TRIGger:PATTern:QUALifier? query returns the trigger duration qualifier.

Return Format

```
<qualifier><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern:GREaterthan" on page 1112
- ":TRIGger:PATTern:LESSthan" on page 1113
- ":TRIGger:PATTern:RANGe" on page 1115

:TRIGger:PATTern:RANGe

(see page 1354)

Command Syntax

```
:TRIGger:PATTern:RANGe <less_than_time>[<suffix>],
                       <greater_than_time>[<suffix>]

<greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format

<less_than_time> ::= 15 ns to 10 seconds in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:PATTern:RANGe command sets the duration for the defined pattern when the :TRIGger:PATTern:QUALifier command is set to INRange or OUTRange. You can enter the parameters in any order – the smaller value becomes the <greater_than_time> and the larger value becomes the <less_than_time>.

Query Syntax

```
:TRIGger:PATTern:RANGe?
```

The :TRIGger:PATTern:RANGe? query returns the duration time for the defined pattern.

Return Format

```
<less_than_time>,<greater_than_time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:PATTern" on page 1109
- ":TRIGger:PATTern:QUALifier" on page 1114
- ":TRIGger:MODE" on page 1066

:TRIGger:PXI Commands

The PXI trigger commands set up arming and triggering between multiple M924xA oscilloscope modules (in a chassis) to synchronize acquisitions.

Table 137 :TRIGger:PXI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PXI:MALine<n>:ENABle {{0 | OFF} | {1 | ON}} (see page 1118)	:TRIGger:PXI:MALine<n>:ENABle? (see page 1118)	<setting> ::= {0 | 1} <n> ::= 0 to (# chassis lines - 1) in NR1 format
:TRIGger:PXI:MODE <mode> (see page 1119)	:TRIGger:PXI:MODE? (see page 1119)	<mode> ::= {OFF | MASTer | SLAVe}
:TRIGger:PXI:MSLot <slot_number> (see page 1120)	:TRIGger:PXI:MSLot? (see page 1120)	<slot_number> ::= 2 to (# chassis slots) in NR1 format
:TRIGger:PXI:SALine <trigger_line> (see page 1121)	:TRIGger:PXI:SALine? (see page 1121)	<arm_line> ::= {PTRig<n>} <n> ::= 0 to (# chassis lines - 1) in NR1 format
:TRIGger:PXI:SYNC {{0 | OFF} | {1 | ON}} (see page 1122)	:TRIGger:PXI:SYNC? (see page 1122)	<setting> ::= {0 | 1}
:TRIGger:PXI:TLINe <trigger_line> (see page 1123)	:TRIGger:PXI:TLINe? (see page 1123)	<trigger_line> ::= {PTRig<n>} <n> ::= 0 to (# chassis lines - 1) in NR1 format

Among all the modules to be synchronized, one module must be the master and all other modules must be slaves. The :TRIGger:PXI:MODE command makes the master or slave selection for the module or turns off PXI triggering for the module (making it a standalone module).

The :TRIGger commands on the master module are used to define the trigger condition. The master module outputs its trigger signal to one of the chassis PXI_TRIG lines (0-7). All slave modules receive the trigger signal over the same PXI_TRIG line, and their :TRIGger:MODE queries show that they are PXI slaves.

There is a separate "armed" signal from each slave module so that the master can wait until each slave is ready before triggering. The remaining PXI_TRIG lines can be used for the arm lines from the slave modules. With arming control, there can be up to seven slave modules. (With software arming in remote programs, there can be more than eight modules in a multiple-module system.)

It is important to identify the master slot number on each slave (:TRIGger:PXI:MSLot) so that the proper set of calibration factors can be used when aligning waveforms. Calibration factors are created and saved using the KtScopeCal.exe application on the chassis controller.

It is also important that each module in the system phase-lock its timebase to the PXIe 100 MHz reference (:ACQuire:RSIGnal PXIE).

After trigger and arm lines are properly configured, calibration factors are created, the master slot is identified in all slaves, and the PXIe 100 MHz reference signal is selected, you are ready to use the multiple-oscilloscope system. Follow this sequence to keep acquisitions correlated:

1. When running, stop acquisitions on the master first.
2. Configure the trigger on the master.
3. Start running (or single) acquisitions on the slaves (they will wait for triggers from the master).
4. Start running (or single) acquisitions on the master.

As with standalone M924xA oscilloscope modules, use single acquisitions for the deepest acquisitions possible.

:TRIGger:PXI:MALine<n>:ENABle

(see page 1354)

Command Syntax

```
:TRIGger:PXI:MALine<n>:ENABle <setting>

<setting> ::= {{0 | OFF} | {1 | ON}}

<n> ::= 0 to (# chassis lines - 1) in NR1 format
```

When this PXIe oscilloscope is the master, the :TRIGger:PXI:MALine<n>:ENABle command enables the chassis PXI_TRIG lines (0-7) on which to wait for "armed" signals before triggering.

Each slave module must use a different PXI_TRIG line for its arm signal.

The master module will wait until all slave modules are armed before triggering.

If the modules are in different chassis trigger bus segments (which are isolated by default), you must use the chassis Soft Front Panel (SFP) application to connect the selected PXI_TRIG lines between trigger bus segments.

Query Syntax

```
:TRIGger:PXI:MALine<n>:ENABle?
```

The :TRIGger:PXI:MALine<n>:ENABle? query returns whether the specified chassis arm line is enabled.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- ":TRIGger:PXI:MODE" on page 1119
- ":TRIGger:PXI:MSLot" on page 1120
- ":TRIGger:PXI:SALine" on page 1121
- ":TRIGger:PXI:TLINe" on page 1123
- ":ACQuire:RSIGnal" on page 250
- ":TRIGger:PXI:SYNC" on page 1122

:TRIGger:PXI:MODE

(see page 1354)

Command Syntax

```
:TRIGger:PXI:MODE <mode>

<mode> ::= {OFF | MASTer | SLAVe}
```

The :TRIGger:PXI:MODE command specifies the PXIe oscilloscope multi-module triggering mode:

- OFF – This module does not participate in multi-module triggering and is a standalone module.
- MASTer – This module is the master module.

  The :TRIGger commands on the master module define the trigger condition. After waiting for "armed" signals from each slave module, the master module outputs the trigger signal on the selected trigger line.
- SLAVe – This module is a slave module.

  Slave modules send an "armed" signal on their selected arm like and are triggered by the selected PXIe trigger line.

  It is important to identify the master slot number on each slave (:TRIGger:PXI:MSLot) so that the proper set of calibration factors can be used when aligning waveforms. Calibration factors are created and saved using the KtScopeCal.exe application on the chassis controller.

Among all the oscilloscope modules to be synchronized, one module must be the master and all other modules must be slaves.

Query Syntax

```
:TRIGger:PXI:MODE?
```

The :TRIGger:PXI:MODE? query returns the PXIe oscilloscope's multi-module triggering mode.

Return Format

```
<mode><NL>

<mode> ::= {OFF | MAST | SLAV}
```

See Also

- ":TRIGger:PXI:MALine<n>:ENABle" on page 1118
- ":TRIGger:PXI:MSLot" on page 1120
- ":TRIGger:PXI:SALine" on page 1121
- ":TRIGger:PXI:TLINe" on page 1123
- ":ACQuire:RSIGnal" on page 250
- ":TRIGger:PXI:SYNC" on page 1122

:TRIGger:PXI:MSLot

(see page 1354)

Command Syntax

```
:TRIGger:PXI:MSLot <slot_number>

<slot_number> ::= 2 to (# chassis slots) in NR1 format
```

When this PXIe oscilloscope is a slave, the :TRIGger:PXI:MSLot command specifies the chassis slot number of the master module.

It is important to identify the master slot number on each slave so that the proper set of calibration factors can be used when aligning waveforms.

Calibration factors are created and saved using the KtScopeCal.exe application on the chassis controller.

Query Syntax

```
:TRIGger:PXI:MSLot?
```

The :TRIGger:PXI:MSLot? query returns the master slot number setting.

Return Format

```
<slot_number><NL>

<slot_number> ::= 2 to (# chassis slots) in NR1 format
```

See Also

- ":TRIGger:PXI:MALine<n>:ENABle" on page 1118
- ":TRIGger:PXI:MODE" on page 1119
- ":TRIGger:PXI:SALine" on page 1121
- ":TRIGger:PXI:TLINe" on page 1123
- ":ACQuire:RSIGnal" on page 250
- ":TRIGger:PXI:SYNC" on page 1122

:TRIGger:PXI:SALine

N (see page 1354)

Command Syntax

```
:TRIGger:PXI:SALine <trigger_line>

<arm_line> ::= {PTRig<n>}

<n> ::= 0 to (# chassis lines - 1) in NR1 format
```

When this PXIe oscilloscope is a slave, the :TRIGger:PXI:SALine command specifies which chassis PXI_TRIG line (0-7) to use for its "armed" signal.

Each slave module must use a different PXI_TRIG line for its arm signal.

If the modules are in different chassis trigger bus segments (which are isolated by default), you must use the chassis Soft Front Panel (SFP) application to connect the selected PXI_TRIG line between trigger bus segments.

Query Syntax

```
:TRIGger:PXI:SALine?
```

The :TRIGger:PXI:SALine? query returns the specified chassis line used for this slave module's "armed" signal.

Return Format

```
<arm_line><NL>

<arm_line> ::= {PTR<n>}
```

See Also

- ":TRIGger:PXI:MALine<n>:ENABle" on page 1118
- ":TRIGger:PXI:MODE" on page 1119
- ":TRIGger:PXI:MSLot" on page 1120
- ":TRIGger:PXI:TLINe" on page 1123
- ":ACQuire:RSIGnal" on page 250
- ":TRIGger:PXI:SYNC" on page 1122

:TRIGger:PXI:SYNC

(see page 1354)

Command Syntax

```
:TRIGger:PXI:SYNC {{0 | OFF} | {1 | ON}}
```

The :TRIGger:PXI:SYNC command enables or disables the sync mode (when setting up multiple M924xA oscilloscope module triggers).

Sync mode lessens the jitter between channels in different modules so that it is more like the jitter between channels within a single module. Sync mode uses the trigger system in the master module for all modules.

When sync mode is OFF, this is called asynchronous mode. Asynchronous mode uses the trigger systems in all modules where the trigger output from the master module is fed to the trigger input of the slave modules.

When you turn ON sync mode, you must use the KtScopeCal.exe application on the chassis controller to align the reference clock signals and perform sync mode trigger delay calibration. Refer to the *Startup Guide*'s chapter on "Coordinating Multiple PXIe Oscilloscope Modules" for more information on the sync mode setup procedure.

Query Syntax

```
:TRIGger:PXI:SYNC?
```

The :TRIGger:PXI:SYNC? query returns the sync mode setting.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- ":TRIGger:PXI:MALine<n>:ENABle" on page 1118
- ":TRIGger:PXI:MODE" on page 1119
- ":TRIGger:PXI:MSLot" on page 1120
- ":TRIGger:PXI:SALine" on page 1121
- ":TRIGger:PXI:TLINe" on page 1123
- ":ACQuire:RSIGnal" on page 250

:TRIGger:PXI:TLINe

N (see page 1354)

Command Syntax

```
:TRIGger:PXI:TLINe <trigger_line>

<trigger_line> ::= {PTRig<n>}

<n> ::= 0 to (# chassis lines - 1) in NR1 format
```

The :TRIGger:PXI:TLINe command specifies the chassis PXI_TRIG line (0-7) to be used for the multi-module trigger signal.

The master module and all slave modules must use the same trigger line.

If the modules are in different chassis trigger bus segments (which are isolated by default), you must use the chassis Soft Front Panel (SFP) application to connect the selected PXI_TRIG line between trigger bus segments.

Query Syntax

```
:TRIGger:PXI:TLINe?
```

The :TRIGger:PXI:TLINe? query returns the specified trigger line.

Return Format

```
<trigger_line><NL>

<trigger_line> ::= {PTR<n>}
```

See Also

- ":TRIGger:PXI:MALine<n>:ENABle" on page 1118
- ":TRIGger:PXI:MODE" on page 1119
- ":TRIGger:PXI:MSLot" on page 1120
- ":TRIGger:PXI:SALine" on page 1121
- ":ACQuire:RSIGnal" on page 250
- ":TRIGger:PXI:SYNC" on page 1122

:TRIGger:RUNT Commands

Table 138 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 1125)	:TRIGger:RUNT:POLarity? (see page 1125)	<polarity> ::= {POSitive | NEGative | EITHer}
:TRIGger:RUNT:QUALifier <qualifier> (see page 1126)	:TRIGger:RUNT:QUALifier? (see page 1126)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 1127)	:TRIGger:RUNT:SOURce? (see page 1127)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1128)	:TRIGger:RUNT:TIME? (see page 1128)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:RUNT:POLarity

(see page 1354)

Command Syntax

```
:TRIGger:RUNT:POLarity <polarity>

<polarity> ::= {POSitive | NEGative | EITHer}
```

The :TRIGger:RUNT:POLarity command sets the polarity for the runt trigger:

- POSitive – positive runt pulses.
- NEGative – negative runt pulses.
- EITHer – either positive or negative runt pulses.

Query Syntax

```
:TRIGger:RUNT:POLarity?
```

The :TRIGger:RUNT:POLarity? query returns the runt trigger polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG | EITH}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:LEVel:HIGH" on page 1064
- ":TRIGger:LEVel:LOW" on page 1065
- ":TRIGger:RUNT:SOURce" on page 1127

:TRIGger:RUNT:QUALifier

(see page 1354)

Command Syntax

```
:TRIGger:RUNT:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | NONE}
```

The :TRIGger:RUNT:QUALifier command selects the qualifier used for specifying runt pulse widths:

- GREaterthan — triggers on runt pulses whose width is greater than the :TRIGger:RUNT:TIME.
- LESSthan — triggers on runt pulses whose width is less than the :TRIGger:RUNT:TIME.
- NONE — triggers on runt pulses of any width.

Query Syntax

```
:TRIGger:RUNT:QUALifier?
```

The :TRIGger:RUNT:QUALifier? query returns the runt trigger qualifier setting.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:RUNT:TIME" on page 1128

:TRIGger:RUNT:SOURce

 (see page 1354)

Command Syntax

```
:TRIGger:RUNT:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:RUNT:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:RUNT:SOURce?
```

The :TRIGger:RUNT:SOURce? query returns the current runt trigger source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:RUNT:POLarity" on page 1125

:TRIGger:RUNT:TIME

(see page 1354)

Command Syntax

```
:TRIGger:RUNT:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

When triggering on runt pulses whose width is greater than or less than a certain value (see :TRIGger:RUNT:QUALifier), the :TRIGger:RUNT:TIME command specifies the time used with the qualifier.

Query Syntax

```
:TRIGger:RUNT:TIME?
```

The :TRIGger:RUNT:TIME? query returns the current runt pulse qualifier time setting.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:RUNT:QUALifier" on page 1126

:TRIGger:SHOLd Commands

Table 139 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 1130)	:TRIGger:SHOLd:SLOPe? (see page 1130)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce :CLOCk <source> (see page 1131)	:TRIGger:SHOLd:SOURce :CLOCk? (see page 1131)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:SHOLd:SOURce :DATA <source> (see page 1132)	:TRIGger:SHOLd:SOURce :DATA? (see page 1132)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 1133)	:TRIGger:SHOLd:TIME:HOLD? (see page 1133)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 1134)	:TRIGger:SHOLd:TIME:SETup? (see page 1134)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:SHOLd:SLOPe

(see page 1354)

Command Syntax

```
:TRIGger:SHOLd:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:SHOLd:SLOPe command specifies whether the rising edge or the falling edge of the clock signal is used.

Query Syntax

```
:TRIGger:SHOLd:SLOPe?
```

The :TRIGger:SHOLd:SLOPe? query returns the current rising or falling edge setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:SHOLd:SOURce:CLOCk" on page 1131
- ":TRIGger:SHOLd:SOURce:DATA" on page 1132

:TRIGger:SHOLd:SOURce:CLOCk

N (see page 1354)

Command Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:SHOLd:SOURce:CLOCk command selects the input channel probing the clock signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk?
```

The :TRIGger:SHOLd:SOURce:CLOCk? query returns the currently set clock signal source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:SHOLd:SLOPe" on page 1130

:TRIGger:SHOLd:SOURce:DATA

 (see page 1354)

Command Syntax

```
:TRIGger:SHOLd:SOURce:DATA <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:SHOLd:SOURce:DATA command selects the input channel probing the data signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:DATA?
```

The :TRIGger:SHOLd:SOURce:DATA? query returns the currently set data signal source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:SHOLd:SLOPe" on page 1130

:TRIGger:SHOLd:TIME:HOLD

(see page 1354)

Command Syntax

```
:TRIGger:SHOLd:TIME:HOLD <time>[suffix]
<time> ::= floating-point number in NR3 format
[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:SHOLd:TIME:HOLD command sets the hold time.

Query Syntax

```
:TRIGger:SHOLd:TIME:HOLD?
```

The :TRIGger:SHOLd:TIME:HOLD? query returns the currently specified hold time.

Return Format

```
<time><NL>
<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:SHOLd:TIME:SETup

(see page 1354)

Command Syntax

```
:TRIGger:SHOLd:TIME:SETup <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:SHOLd:TIME:SETup command sets the setup time.

Query Syntax

```
:TRIGger:SHOLd:TIME:SETup?
```

The :TRIGger:SHOLd:TIME:SETup? query returns the currently specified setup time.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053

:TRIGger:TRANsition Commands

The :TRIGger:TRANsition commands set the rise/fall time trigger options.

Table 140 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 1136)	:TRIGger:TRANsition:QUALifier? (see page 1136)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 1137)	:TRIGger:TRANsition:SLOPe? (see page 1137)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 1138)	:TRIGger:TRANsition:SOURce? (see page 1138)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 1139)	:TRIGger:TRANsition:TIME? (see page 1139)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:TRANsition:QUALifier

(see page 1354)

Command Syntax

```
:TRIGger:TRANsition:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan}
```

The :TRIGger:TRANsition:QUALifier command specifies whether you are looking for rise/fall times greater than or less than a certain time value. The time value is set using the :TRIGger:TRANsition:TIME command.

Query Syntax

```
:TRIGger:TRANsition:QUALifier?
```

The :TRIGger:TRANsition:QUALifier? query returns the current rise/fall time trigger qualifier setting.

Return Format

```
<qualifier><NL>

<qualifier> ::= {GRE | LESS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:TRANsition:TIME" on page 1139
- ":TRIGger:MODE" on page 1066

:TRIGger:TRANsition:SLOPe

 (see page 1354)

Command Syntax

```
:TRIGger:TRANsition:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:TRANsition:SLOPe command specifies a POSitive rising edge or a NEGative falling edge.

Query Syntax

```
:TRIGger:TRANsition:SLOPe?
```

The :TRIGger:TRANsition:SLOPe? query returns the current rise/fall time trigger slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:TRANsition:SOURce" on page 1138

:TRIGger:TRANsition:SOURce

(see page 1354)

Command Syntax

```
:TRIGger:TRANsition:SOURce <source>

<source> ::= CHANnel<n>

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:TRANsition:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:TRANsition:SOURce?
```

The :TRIGger:TRANsition:SOURce? query returns the current transition trigger source.

Return Format

```
<source><NL>

<source> ::= CHAN<n>
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:TRANsition:SLOPe" on page 1137

:TRIGger:TRANsition:TIME

(see page 1354)

Command Syntax

```
:TRIGger:TRANsition:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:TRANsition:TIME command sets the time value for rise/fall time triggers. You also use the :TRIGger:TRANsition:QUALifier command to specify whether you are triggering on times greater than or less than this time value.

Query Syntax

```
:TRIGger:TRANsition:TIME?
```

The :TRIGger:TRANsition:TIME? query returns the current rise/fall time trigger time value.

Return Format

```
<time><NL>

<time> ::= floating-point number in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:TRANsition:QUALifier" on page 1136

:TRIGger:TV Commands

Table 141 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 1141)	:TRIGger:TV:LINE? (see page 1141)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 1142)	:TRIGger:TV:MODE? (see page 1142)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | LFIeld1 | LFIeld2 | LALTernate}
:TRIGger:TV:POLarity <polarity> (see page 1143)	:TRIGger:TV:POLarity? (see page 1143)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 1144)	:TRIGger:TV:SOURce? (see page 1144)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see page 1145)	:TRIGger:TV:STANdard? (see page 1145)	<standard> ::= {NTSC | PAL | PALM | SECam} <standard> ::= {GENeric | {P480L60HZ | P480} | {P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | P1080L50HZ | P1080L60HZ | {I1080L50HZ | I1080} | I1080L60HZ} with extended video triggering license
:TRIGger:TV:UDTV:ENUMber <count> (see page 1146)	:TRIGger:TV:UDTV:ENUMber? (see page 1146)	<count> ::= edge number in NR1 format
:TRIGger:TV:UDTV:HSYNc {{0 | OFF} | {1 | ON}} (see page 1147)	:TRIGger:TV:UDTV:HSYNc? (see page 1147)	{0 | 1}
:TRIGger:TV:UDTV:HTIMe <time> (see page 1148)	:TRIGger:TV:UDTV:HTIMe? (see page 1148)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTHan <min_time> (see page 1149)	:TRIGger:TV:UDTV:PGTHan? (see page 1149)	<min_time> ::= seconds in NR3 format

:TRIGger:TV:LINE

N (see page 1354)

Command Syntax

```
:TRIGger:TV:LINE <line_number>

<line_number> ::= integer in NR1 format
```

The :TRIGger:TV:LINE command allows triggering on a specific line of video. The line number limits vary with the standard and mode, as shown in the following table.

Table 142 TV Trigger Line Number Limits

TV Standard	Mode				
	LINE	LFIeld1	LFIeld2	LALTernate	VERTical
NTSC		1 to 263	1 to 262	1 to 262	
PAL		1 to 313	314 to 625	1 to 312	
PAL-M		1 to 263	264 to 525	1 to 262	
SECAM		1 to 313	314 to 625	1 to 312	
GENERIC		1 to 1024	1 to 1024		1 to 1024
P480L60HZ	1 to 525				
P720L60HZ	1 to 750				
P1080L24HZ	1 to 1125				
P1080L25HZ	1 to 1125				
P1080L50HZ	1 to 1125				
P1080L60HZ	1 to 1125				
I1080L50HZ	1 to 1125				
I1080L60HZ	1 to 1125				

Query Syntax

```
:TRIGger:TV:LINE?
```

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

Return Format

```
<line_number><NL>

<line_number>::= integer in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:TV:MODE" on page 1142

:TRIGger:TV:MODE

(see page 1354)

Command Syntax

```
:TRIGger:TV:MODE <mode>

<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | LFIeld1
           | LFIeld2 | LALTernate}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIelds	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt

Query Syntax

```
:TRIGger:TV:MODE?
```

The :TRIGger:TV:MODE? query returns the TV trigger mode.

Return Format

```
<value><NL>

<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | LFI1 | LFI2 | LALT}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:MODE" on page 1066

:TRIGger:TV:POLarity

(see page 1354)

Command Syntax

```
:TRIGger:TV:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:TV:POLarity command sets the polarity for the TV trigger.

Query Syntax

```
:TRIGger:TV:POLarity?
```

The :TRIGger:TV:POLarity? query returns the TV trigger polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {POS | NEG}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:TV:SOURce" on page 1144

:TRIGger:TV:SOURce

N (see page 1354)

Command Syntax

```
:TRIGger:TV:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:TV:SOURce?
```

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":TRIGger:TV:POLarity" on page 1143

Example Code

- "Example Code" on page 1086

:TRIGger:TV:STANdard

(see page 1354)

Command Syntax

```
:TRIGger:TV:STANdard <standard>

<standard> ::= {GENeric | NTSC | PALM | PAL | SECam
               | {P480L60HZ | P480} | {P720L60HZ | P720}
               | {P1080L24HZ | P1080} | P1080L25HZ
               | P1080L50HZ | P1080L60HZ
               | {I1080L50HZ | I1080} | I1080L60HZ}
```

The :TRIGger:TV:STANdard command selects the video standard:

- NTSC
- PAL
- PAL-M
- SECAM

With an extended Video triggering license, the oscilloscope additionally supports these standards:

- Generic – GENeric mode is non-interlaced.
- EDTV 480p/60
- HDTV 720p/60
- HDTV 1080p/24
- HDTV 1080p/25
- HDTV 1080i/50
- HDTV 1080i/60

Query Syntax

```
:TRIGger:TV:STANdard?
```

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

Return Format

```
<standard><NL>

<standard> ::= {GEN | NTSC | PALM | PAL | SEC | P480L60HZ | P760L60HZ
               | P1080L24HZ | P1080L25HZ | P1080L50HZ | P1080L60HZ
               | I1080L50HZ | I1080L60HZ}
```

:TRIGger:TV:UDTV:ENUMber

(see page 1354)

Command Syntax

```
:TRIGger:TV:UDTV:ENUMber <count>

<count> ::= edge number in NR1 format
```

The :TRIGger:TV:UDTV:ENUMber command specifies the Generic video trigger's Nth edge to trigger on after synchronizing with the vertical sync.

Query Syntax

```
:TRIGger:TV:UDTV:ENUMber?
```

The :TRIGger:TV:UDTV:ENUMber query returns the edge count setting.

Return Format

```
<count><NL>

<count> ::= edge number in NR1 format
```

See Also

- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:TV:UDTV:PGTHan" on page 1149
- ":TRIGger:TV:UDTV:HSYNc" on page 1147

:TRIGger:TV:UDTV:HSYNc

(see page 1354)

Command Syntax `:TRIGger:TV:UDTV:HSYNc {{0 | OFF} | {1 | ON}}`

The :TRIGger:TV:UDTV:HSYNc command enables or disables the horizontal sync control in the Generic video trigger.

For interleaved video, enabling the HSYNc control and setting the HTIMe adjustment to the sync time of the probed video signal allows the ENUMber function to count only lines and not double count during equalization. Additionally, the Field Holdoff can be adjusted so that the oscilloscope triggers once per frame.

Similarly, for progressive video with a tri-level sync, enabling the HSYNc control and setting the HTIMe adjustment to the sync time of the probed video signal allows the ENUMber function to count only lines and not double count during vertical sync.

Query Syntax `:TRIGger:TV:UDTV:HSYNc?`

The :TRIGger:TV:UDTV:HSYNc query returns the horizontal sync control setting.

Return Format `{0 | 1}`

See Also

- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:TV:UDTV:HTIMe" on page 1148
- ":TRIGger:TV:UDTV:ENUMber" on page 1146
- ":TRIGger:TV:UDTV:PGTHan" on page 1149

:TRIGger:TV:UDTV:HTIMe

(see page 1354)

Command Syntax

```
:TRIGger:TV:UDTV:HTIMe <time>

<time> ::= seconds in NR3 format
```

When the Generic video trigger's horizontal sync control is enabled, the :TRIGger:TV:UDTV:HTIMe command sets the minimum time the horizontal sync pulse must be present to be considered valid.

Query Syntax

```
:TRIGger:TV:UDTV:HTIMe?
```

The :TRIGger:TV:UDTV:HTIMe query returns the horizontal sync time setting.

Return Format

```
<time><NL>

<time> ::= seconds in NR3 format
```

See Also

- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:TV:UDTV:HSYNc" on page 1147

:TRIGger:TV:UDTV:PGTHan

N (see page 1354)

Command Syntax

```
:TRIGger:TV:UDTV:PGTHan <min_time>

<min_time> ::= seconds in NR3 format
```

The :TRIGger:TV:UDTV:PGTHan command specifies the "greater than the sync pulse width" time in the Generic video trigger. This setting allows oscilloscope synchronization to the vertical sync.

Query Syntax

```
:TRIGger:TV:UDTV:PGTHan?
```

The :TRIGger:TV:UDTV:PGTHan query returns the "greater than the sync pulse width" time setting.

Return Format

```
<min_time><NL>

<min_time> ::= seconds in NR3 format
```

See Also

- ":TRIGger:TV:STANdard" on page 1145
- ":TRIGger:TV:UDTV:ENUMber" on page 1146
- ":TRIGger:TV:UDTV:HSYNc" on page 1147

:TRIGger:ZONE Commands

Table 143 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE:SOURce <source> (see page 1151)	:TRIGger:ZONE:SOURce? (see page 1151)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:ZONE:STATe {{0 | OFF} | {1 | ON}} (see page 1152)	:TRIGger:ZONE:STATe? (see page 1152)	{0 | 1}
:TRIGger:ZONE<n>:MODE <mode> (see page 1153)	:TRIGger:ZONE<n>:MODE? (see page 1153)	<mode> ::= {INTersect | NOTintersect} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:PLACement <width>, <height>, <x_center>, <y_center> (see page 1154)	:TRIGger:ZONE<n>:PLACement? (see page 1154)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <n> ::= 1-2 in NR1 format
n/a	:TRIGger:ZONE<n>:VALidity? (see page 1155)	<value> ::= {VALid | INValid | OSCReen} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:STATe {{0 | OFF} | {1 | ON}} (see page 1156)	:TRIGger:ZONE<n>:STATe? (see page 1156)	{0 | 1} <n> ::= 1-2 in NR1 format

:TRIGger:ZONE:SOURce

N (see page 1354)

Command Syntax

```
:TRIGger:ZONE:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :TRIGger:ZONE:SOURce command sets the analog source channel shared by all zones.

Query Syntax

```
:TRIGger:ZONE:SOURce?
```

The :TRIGger:ZONE:SOURce? query returns the analog source channel specified for zone qualified triggers.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- ":TRIGger:ZONE:STATe" on page 1152

:TRIGger:ZONE:STATe

(see page 1354)

Command Syntax

```
:TRIGger:ZONE:STATe <on_off>

<on_off> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:ZONE:STATe command enables or disables the zone qualified trigger feature.

When the zone qualified trigger is on, the zone(s) are actively being used to qualify the trigger.

Note that the :TRIGger:ZONE<n>:STATe setting must also be ON for a zone to be active.

Note that :TRIGger:ZONE:STATe mimics the behavior of the **[Zone]** key on the front panel, and :TRIGger:ZONE<n>:STATe mimics the behavior of the **Zone 1 On** and **Zone 2 On** softkeys. At least one zone's state must be on for the Zone Trigger feature (:TRIGger:ZONE:STATe) to be on. When the states of both individual zones are turned off, Zone Trigger is automatically turned off. In this case, when Zone Trigger is turned back on Zone 1 is forced to on. Otherwise, if at least one zone was on when Zone Trigger was turned off, the same configuration of individual zone on/off states will be restored when Zone Trigger is turned back on.

Query Syntax

```
:TRIGger:ZONE:STATe?
```

The :TRIGger:ZONE:STATe? query returns whether the zone qualified trigger feature is enabled or disabled.

Return Format

```
<on_off><NL>

<on_off> ::= {0 | 1}
```

See Also

- ":TRIGger:ZONE<n>:STATe" on page 1156
- ":TRIGger:ZONE:SOURce" on page 1151

:TRIGger:ZONE<n>:MODE

(see page 1354)

Command Syntax

```
:TRIGger:ZONE<n>:MODE <mode>

<mode> ::= {INTersect | NOTintersect}

<n> ::= 1-2 in NR1 format
```

The :TRIGger:ZONE<n>:MODE command sets the zone qualifying condition for Zone 1 or Zone 2 as either "Must Intersect" or "Must Not Intersect".

Query Syntax

```
:TRIGger:ZONE<n>:MODE?
```

The :TRIGger:ZONE<n>:MODE? query returns the zone qualifying condition for Zone 1 or Zone 2.

Return Format

```
<mode><NL>

<mode> ::= {INT | NOT}
```

See Also

- ":TRIGger:ZONE<n>:STATe" on page 1156
- ":TRIGger:ZONE<n>:PLACement" on page 1154
- ":TRIGger:ZONE<n>:VALidity" on page 1155

:TRIGger:ZONE<n>:PLACement

N (see page 1354)

Command Syntax

```
:TRIGger:ZONE<n>:PLACement <width>, <height>, <x_center>, <y_center>

<width> ::= width of zone in seconds

<height> ::= height of zone in volts

<x_center> ::= center of zone in seconds

<y_center> ::= center of zone in volts

<n> ::= 1-2 in NR1 format
```

The :TRIGger:ZONE<n>:PLACement command sets the size and location of Zone 1 or Zone 2.

No error is returned if the zone is placed off-screen, or if the zones overlap such that Zone 2 becomes invalid. The :TRIGger:ZONE<n>:VALidity? query is used to retrieve this information.

Query Syntax

```
:TRIGger:ZONE<n>:PLACement?
```

The :TRIGger:ZONE<n>:PLACement? query returns the size and location of Zone 1 or Zone 2.

Return Format

```
<opt><NL>

<opt> ::= <width>, <height>, <x_center>, <y_center>
```

See Also

- ":TRIGger:ZONE<n>:STATe" on page 1156
- ":TRIGger:ZONE<n>:MODE" on page 1153
- ":TRIGger:ZONE<n>:VALidity" on page 1155

:TRIGger:ZONE<n>:VALidity

(see page 1354)

Query Syntax

```
:TRIGger:ZONE<n>:VALidity?

<n> ::= 1-2 in NR1 format
```

The :TRIGger:ZONE<n>:VALidity? query returns the validity of Zone 1 or Zone 2.

- INValid is returned (for Zone 2 only) when Zone 1 and Zone 2 overlap and have opposing qualifying conditions (modes). Zone 1 can never be invalid.
- OSCReen (off-screen) is returned when the associated zone is off-screen, and thus not being used to qualify the trigger.
- A zone is valid when it is neither invalid nor off-screen.

The validity of a zone is not affected by the zone's state. For example, a zone can be valid and off. You cannot directly set the validity of a zone.

Return Format

```
<validity><NL>

<validity> ::= {VALid | INValid | OSCReen}
```

See Also

- ":TRIGger:ZONE<n>:STATe" on page 1156
- ":TRIGger:ZONE<n>:MODE" on page 1153
- ":TRIGger:ZONE<n>:PLACement" on page 1154

:TRIGger:ZONE<n>:STATe

N (see page 1354)

Command Syntax

```
:TRIGger:ZONE<n>:STATe <on_off>

<n> ::= {1 | 2}

<on_off> ::= {{0 | OFF} | {1 | ON}}

<n> ::= 1-2 in NR1 format
```

The :TRIGger:ZONE<n>:STATe command sets the state for Zone 1 or Zone 2.

- When a zone's state is on, and the Zone Trigger feature is on (see ":TRIGger:ZONE:STATe" on page 1152), that zone is actively being used to qualify the trigger if it is not invalid or off-screen (see ":TRIGger:ZONE<n>:VALidity" on page 1155).
- When the Zone Trigger feature is off, no zones are being used to qualify the trigger, regardless of their individual states.

Note that :TRIGger:ZONE:STATe mimics the behavior of the **[Zone]** key on the front panel, and :TRIGger:ZONE<n>:STATe mimics the behavior of the **Zone 1 On** and **Zone 2 On** softkeys. At least one zone's state must be on for the Zone Trigger feature (:TRIGger:ZONE:STATe) to be on. When the states of both individual zones are turned off, Zone Trigger is automatically turned off. In this case, when Zone Trigger is turned back on Zone 1 is forced to on. Otherwise, if at least one zone was on when Zone Trigger was turned off, the same configuration of individual zone on/off states will be restored when Zone Trigger is turned back on.

Query Syntax

```
:TRIGger:ZONE<n>:STATe?
```

The :TRIGger:ZONE<n>:STATe? query returns the state of Zone 1 or Zone 2.

Return Format

```
<on_off><NL>

<on_off> ::= {0 | 1}
```

See Also

- ":TRIGger:ZONE:STATe" on page 1152
- ":TRIGger:ZONE<n>:VALidity" on page 1155

32 :WAVeform Commands

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 1160.

Table 144 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 1165)	:WAVeform:BYTeorder? (see page 1165)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 1166)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 1167)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMat <value> (see page 1169)	:WAVeform:FORMat? (see page 1169)	<value> ::= {WORD | BYTE | ASCII}

Table 144 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINts <# points> (see page 1170)	:WAVeform:POINts? (see page 1170)	<# points> ::= {100 | 250 | 500 | 1000 | <points_mode>} if waveform points mode is NORMal <# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMal | MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode> (see page 1172)	:WAVeform:POINts:MODE? (see page 1173)	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble? (see page 1174)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 3 for AVERage type • 4 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
n/a	:WAVeform:SEGMented:COUNt? (see page 1177)	<count> ::= an integer from 2 to 1000 in NR1 format (with SGM license)
n/a	:WAVeform:SEGMented:TTAG? (see page 1178)	<time_tag> ::= in NR3 format (with SGM license)

Table 144 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SOURce <source> (see page 1179)	:WAVeform:SOURce? (see page 1179)	<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r> | SBUS{1 | 2}} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 1183)	:WAVeform:SOURce:SUBSource? (see page 1183)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
n/a	:WAVeform:TYPE? (see page 1184)	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 1185)	:WAVeform:UNSigned? (see page 1185)	{0 | 1}
:WAVeform:VIEW <view> (see page 1186)	:WAVeform:VIEW? (see page 1186)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 1187)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 1188)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 1189)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 1190)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 1191)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 1192)	<return_value> ::= y-reference value in the current preamble in NR1 format

Introduction to :WAVeform Commands

The WAVeform subsystem is used to transfer data to a controller from the oscilloscope waveform memories. The queries in this subsystem will only operate when the channel selected by :WAVeform:SOURce is on.

Waveform Data and Preamble

The waveform record is actually contained in two portions: the preamble and waveform data. The waveform record must be read from the oscilloscope by the controller using two separate commands, :WAVeform:DATA (see page 1167) and :WAVeform:PREamble (see page 1174). The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

Data Acquisition Types

There are four types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command (see page 257): NORMal, AVERage, PEAK, and HRESolution. When the data is acquired using the :DIGitize command (see page 211) or :RUN command (see page 233), the data is placed in the channel buffer of the specified source.

Once you have acquired data with the :DIGitize command, the instrument is stopped. If the instrument is restarted (via the programming interface or the front panel), or if any instrument setting is changed, the data acquired with the :DIGitize command may be overwritten.You should first acquire the data with the :DIGitize command, then immediately read the data with the :WAVeform:DATA? query (see page 1167) before changing any instrument setup.

A waveform record consists of either all of the acquired points or a subset of the acquired points. The number of points acquired may be queried using :ACQuire:POINts? (see page 249).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINts command (see page 1170). If :WAVeform:POINts MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVeform:POINts may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVeform:POINts must be an even divisor of 1,000 or be set to MAXimum. :WAVeform:POINts determines the increment between time buckets that will be transferred. If POINts = MAXimum, the data cannot be decimated. For example:

- `:WAVeform:POINts 1000` — returns time buckets 0, 1, 2, 3, 4 ,.., 999.

- `:WAVeform:POINts 500` — returns time buckets 0, 2, 4, 6, 8 ,.., 998.
- `:WAVeform:POINts 250` — returns time buckets 0, 4, 8, 12, 16 ,.., 996.
- `:WAVeform:POINts 100` — returns time buckets 0, 10, 20, 30, 40 ,.., 990.

Analog Channel Data

NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVeform:POINts? query (see page 1170). Only the magnitude values of each data point are transmitted. The first voltage value corresponds to the first time bucket on the left side of the screen and the last value corresponds to the next-to-last time bucket on the right side of the screen. Time buckets without data return 0. The time values for each data point correspond to the position of the data point in the data array. These time values are not transmitted.

AVERage Data

AVERage data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQuire:COUNt query (see page 246). Time buckets that have fewer than n hits return the average of the data they do have. If a time bucket does not have any data in it, it returns 0.

This data is transmitted over the interface linearly, starting with time bucket 0 and proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see page 1170). The first value corresponds to a point at the left side of the screen and the last value corresponds to one point away from the right side of the screen. The maximum number of points that can be returned in average mode is 1000 unless ACQuire:COUNt has been set to 1.

PEAK Data

Peak detect display mode is used to detect glitches for time base settings of 500 us/div and slower. In this mode, the oscilloscope can sample more data than it can store and display. So, when peak detect is turned on, the oscilloscope scans through the extra data, picks up the minimum and maximum for each time bucket, then stores the data in an array. Each time bucket contains two data sample.

The array is transmitted over the interface bus linearly, starting with time bucket 0 proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see page 1170). In each time bucket, two values are transmitted, first the minimum, followed by the maximum. The first pair of values corresponds to the time bucket at the leftmost side of the screen. The last pair of values corresponds to the time bucket at the far right side of the screen. In :ACQuire:TYPE PEAK mode (see page 257), the value returned by the :WAVeform:XINCrement query (see page 1187) should be doubled to find the time difference between the min-max pairs.

HRESolution Data

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

Data Conversion

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

voltage = [(data value - yreference) * yincrement] + yorigin

If the :WAVeform:FORMat data format is ASCii (see page 1169), the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

time = [(data point number - xreference) * xincrement] + xorigin

This would result in the following calculation for time bucket 3:

time = [(3 - 0) * 2 ns] + 16 ns = 22 ns

In :ACQuire:TYPE PEAK mode (see page 257), because data is acquired in max-min pairs, modify the previous time formula to the following:

time=[(data pair number - xreference) * xincrement * 2] + xorigin

Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCii (see ":WAVeform:FORMat" on page 1169). BYTE, WORD and ASCii formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE 488.2.

When you use the block data format, the ASCII character string "#8<DD...D>" is sent prior to sending the actual data. The 8 indicates how many Ds follow. The Ds are ASCII numbers that indicate how many data bytes follow.

For example, if 1000 points will be transferred, and the WORD format was specified, the block header "#800001000" would be sent. The 8 indicates that eight length bytes follow, and 00001000 indicates that 1000 binary data bytes follow.

Use the :WAVeform:UNSigned command (see page 1185) to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

Data Format for Transfer - ASCii format

The ASCii format (see ":WAVeform:FORMat" on page 1169) provides access to the waveform data as real Y-axis values without using Y origin, Y reference, and Y increment to convert the binary data. Values are transferred as ASCii digits in floating point format separated by commas. In ASCii format, holes are represented by the value 9.9e+37. The setting of :WAVeform:BYTeorder (see page 1165) and :WAVeform:UNSigned (see page 1185) have no effect when the format is ASCii.

Data Format for Transfer - WORD format

WORD format (see ":WAVeform:FORMat" on page 1169) provides 16-bit access to the waveform data. In the WORD format, the number of data bytes is twice the number of data points. The number of data points is the value returned by the :WAVeform:POINts? query (see page 1170). If the data intrinsically has less than 16 bits of resolution, the data is left-shifted to provide 16 bits of resolution and the least significant bits are set to 0. Currently, the greatest intrinsic resolution of any data is 12 bits, so at least the lowest 4 bits of data will be 0. If there is a hole in the data, the hole is represented by a 16 bit value equal to 0.

Use :WAVeform:BYTeorder (see page 1165) to determine if the least significant byte or most significant byte is to be transferred first. The :BYTeorder command can be used to alter the transmit sequence to match the storage sequence of an integer in the programming language being used.

Data Format for Transfer - BYTE format

The BYTE format (see ":WAVeform:FORMat" on page 1169) allows 8-bit access to the waveform data. If the data intrinsically has more than 8 bits of resolution (averaged data), the data is right-shifted (truncated) to fit into 8 bits. If there is a hole in the data, the hole is represented by a value of 0. The BYTE-formatted data transfers over the programming interface faster than ASCii or WORD-formatted data, because in ASCii format, as many as 13 bytes per point are transferred, in BYTE format one byte per point is transferred, and in WORD format two bytes per point are transferred.

The :WAVeform:BYTeorder command (see page 1165) has no effect when the data format is BYTE.

Reporting the Setup

The following is a sample response from the :WAVeform? query. In this case, the query was issued following a *RST command.

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS
NONE
```

:WAVeform:BYTeorder

(see page 1354)

Command Syntax

```
:WAVeform:BYTeorder <value>

<value> ::= {LSBFirst | MSBFirst}
```

The :WAVeform:BYTeorder command sets the output sequence of the WORD data.

- MSBFirst – sets the most significant byte to be transmitted first.
- LSBFirst – sets the least significant byte to be transmitted first.

This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected.

The default setting is MSBFirst.

Query Syntax

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder query returns the current output sequence.

Return Format

```
<value><NL>

<value> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:DATA" on page 1167
- ":WAVeform:FORMat" on page 1169
- ":WAVeform:PREamble" on page 1174

Example Code

- "Example Code" on page 1180
- "Example Code" on page 1175

:WAVeform:COUNt

(see page 1354)

Query Syntax `:WAVeform:COUNt?`

The :WAVeform:COUNT? query returns the count used to acquire the current waveform. This may differ from current values if the unit has been stopped and its configuration modified. For all acquisition types except average, this value is 1.

Return Format `<count_argument><NL>`

`<count_argument> ::= an integer from 1 to 65536 in NR1 format`

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":ACQuire:COUNt" on page 246
- ":ACQuire:TYPE" on page 257

:WAVeform:DATA

(see page 1354)

Query Syntax `:WAVeform:DATA?`

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSigned, :WAVeform:BYTeorder, :WAVeform:FORMat, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINts command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 – Hole. Holes are locations where data has not yet been acquired.

  Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.
- 0x01 or 0x0001 – Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFFFF – Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

Return Format `<binary block data><NL>`

See Also

- For a more detailed description of the data returned for different acquisition types, see: "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:UNSigned" on page 1185
- ":WAVeform:BYTeorder" on page 1165
- ":WAVeform:FORMat" on page 1169
- ":WAVeform:POINts" on page 1170
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:SOURce" on page 1179
- ":WAVeform:TYPE" on page 1184

Example Code

```
' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
```

```
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1)   ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)   ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) _
        * sngYIncrement + sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) _
        * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:WAVeform:FORMat

 (see page 1354)

Command Syntax

```
:WAVeform:FORMat <value>

<value> ::= {WORD | BYTE | ASCii}
```

The :WAVeform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

  ASCII formatted data is transferred ASCii text.
- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCii is the only waveform format allowed.

Query Syntax

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

Return Format

```
<value><NL>

<value> ::= {WORD | BYTE | ASC}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:BYTeorder" on page 1165
- ":WAVeform:SOURce" on page 1179
- ":WAVeform:DATA" on page 1167
- ":WAVeform:PREamble" on page 1174

Example Code

- "Example Code" on page 1180

:WAVeform:POINts

(see page 1354)

Command Syntax

```
:WAVeform:POINts <# points>

<# points> ::= {100 | 250 | 500 | 1000 | <points mode>}
               if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
               | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
               | 4000000 | 8000000 | <points mode>}
               if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMal | MAXimum | RAW}
```

NOTE The <points_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog sources, the records that can be transferred depend on the waveform points mode. The maximum number of points returned for math (function) waveforms is determined by the NORMal waveform points mode. See the :WAVeform:POINts:MODE command (see page 1172) for more information.

Only data visible on the display will be returned.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), this command is ignored, and all available serial decode bus data is returned.

Query Syntax

```
:WAVeform:POINts?
```

The :WAVeform:POINts query returns the number of waveform points to be transferred when using the :WAVeform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVeform:POINts:MODE command (see page 1172) for more information).

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), this query returns the number of messages that were decoded.

Return Format

```
<# points><NL>

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
               if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
               | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
               | 4000000 | 8000000 | <maximum # points>}
               if waveform points mode is MAXimum or RAW
```

NOTE If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.

See Also
- "Introduction to :WAVeform Commands" on page 1160
- ":ACQuire:POINts" on page 249
- ":WAVeform:DATA" on page 1167
- ":WAVeform:SOURce" on page 1179
- ":WAVeform:VIEW" on page 1186
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:POINts:MODE" on page 1172

Example Code

```
' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:WAVeform:POINts:MODE

(see page 1354)

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog sources, there are three different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be retrieved from the analog sources.
- The second is referred to as the *measurement record* and is a 65,535-point (maximum) representation of the raw acquisition record. The measurement record can be retrieved, when :SYSTem:PRECision is OFF, from any source.
- The third is referred to as the *precision analysis record* and is a user-selectable 100k-point to 1M-point (maximum) representation of the raw acquisition record. The precision analysis record can be retrieved when :SYSTem:PRECision is ON, from analog sources.

If the <points_mode> is NORMal and :SYSTem:PRECision is OFF, the measurement record is retrieved.

If the <points_mode> is NORMal and :SYSTem:PRECision is ON, the precision analysis record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

NOTE

If the :WAVeform:SOURce is not an analog or digital source, the only valid parameters for WAVeform:POINts:MODE is NORMal or MAXimum.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see page 237) or the :DIGitize command (see page 211) in the root subsystem) in order to return more than the *measurement record* or *precision analysis record*.
- :TIMebase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution.

- MAXimum or RAW will allow up to 4,000,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVeform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

Query Syntax

```
:WAVeform:POINts:MODE?
```

The :WAVeform:POINts:MODE? query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

Return Format

```
<points_mode><NL>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:DATA" on page 1167
- ":ACQuire:POINts" on page 249
- ":WAVeform:VIEW" on page 1186
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:POINts" on page 1170
- ":TIMebase:MODE" on page 1041
- ":ACQuire:TYPE" on page 257
- ":ACQuire:COUNt" on page 246
- ":SYSTem:PRECision" on page 1021
- ":SYSTem:PRECision:LENGth" on page 1022

:WAVeform:PREamble

(see page 1354)

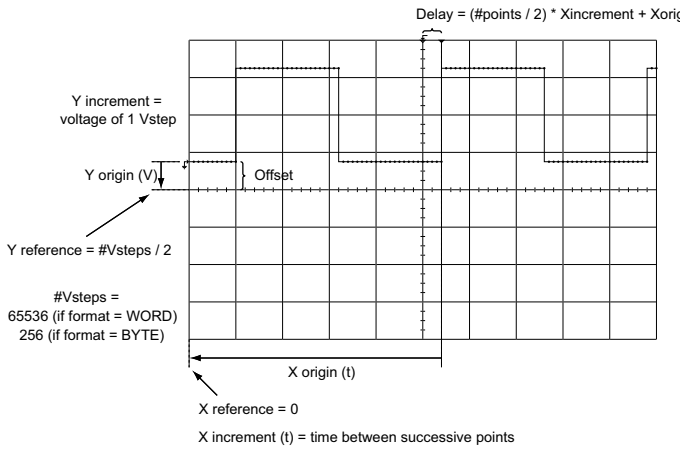
Query Syntax

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble query requests the preamble information for the selected waveform source. The preamble data contains information concerning the vertical and horizontal scaling of the data of the corresponding channel.

Return Format

```
<preamble_block><NL>

<preamble_block> ::= <format 16-bit NR1>,
                     <type 16-bit NR1>,
                     <points 32-bit NR1>,
                     <count 32-bit NR1>,
                     <xincrement 64-bit floating point NR3>,
                     <xorigin 64-bit floating point NR3>,
                     <xreference 32-bit NR1>,
                     <yincrement 32-bit floating point NR3>,
                     <yorigin 32-bit floating point NR3>,
                     <yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 4 for ASCii format;
             an integer in NR1 format (format set by :WAVeform:FORMat).

<type> ::= 3 for HRESolution type, 2 for AVERage type, 0 for NORMal
           type, 1 for PEAK detect type; an integer in NR1 format
           (type set by :ACQuire:TYPE).

<count> ::= Average count or 1 if PEAK or NORMal; an integer in NR1
            format (count set by :ACQuire:COUNt).
```



See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":ACQuire:COUNt" on page 246
- ":ACQuire:POINts" on page 249

- ":ACQuire:TYPE" on page 257
- ":DIGitize" on page 211
- ":WAVeform:COUNt" on page 1166
- ":WAVeform:DATA" on page 1167
- ":WAVeform:FORMat" on page 1169
- ":WAVeform:POINts" on page 1170
- ":WAVeform:TYPE" on page 1184
- ":WAVeform:XINCrement" on page 1187
- ":WAVeform:XORigin" on page 1188
- ":WAVeform:XREFerence" on page 1189
- ":WAVeform:YINCrement" on page 1190
- ":WAVeform:YORigin" on page 1191
- ":WAVeform:YREFerence" on page 1192

Example Code

```
' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'    TYPE          : int16 - 0 = NORM, 1 = PEAK, 2 = AVER, 3 = HRES
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with
'                            x-origin.
'    YINCREMENT    : float32 - voltage diff between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin
'                            occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"    ' Query for the preamble.
Preamble() = myScope.ReadList    ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
```

```
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:WAVeform:SEGMented:COUNt

(see page 1354)

Query Syntax `:WAVeform:SEGMented:COUNt?`

NOTE This command is available when the segmented memory option (SGM license) is installed.

The :WAVeform:SEGMented:COUNt query returns the number of memory segments in the acquired data. You can use the :WAVeform:SEGMented:COUNt? query while segments are being acquired (although :DIGitize blocks subsequent queries until the full segmented acquisition is complete).

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGMented:COUNt command, and data is acquired using the :DIGitize, :SINGle, or :RUN commands.

Return Format

```
<count> ::= an integer from 2 to 1000 in NR1 format (count set by
            :ACQuire:SEGMented:COUNt).
```

See Also

- ":ACQuire:MODE" on page 248
- ":ACQuire:SEGMented:COUNt" on page 252
- ":DIGitize" on page 211
- ":SINGle" on page 235
- ":RUN" on page 233
- "Introduction to :WAVeform Commands" on page 1160

Example Code

- "Example Code" on page 253

:WAVeform:SEGMented:TTAG

 (see page 1354)

Query Syntax `:WAVeform:SEGMented:TTAG?`

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

The :WAVeform:SEGMented:TTAG? query returns the time tag of the currently selected segmented memory index. The index is selected using the :ACQuire:SEGMented:INDex command.

Return Format `<time_tag> ::= in NR3 format`

See Also
- ":ACQuire:SEGMented:INDex" on page 253
- "Introduction to :WAVeform Commands" on page 1160

Example Code
- "Example Code" on page 253

:WAVeform:SOURce

(see page 1354)

Command Syntax

```
:WAVeform:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | FFT | WMEMory<r>
              | SBUS{1 | 2}}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :WAVeform:SOURce command selects the analog channel, function, reference waveform, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply, integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCii is the only waveform format allowed, and the :WAVeform:DATA? query returns a string with timestamps and associated bus decode information.

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE

MATH<m> is an alias for FUNCtion<m>. The :WAVeform:SOURce? query returns FUNC<m> if the source is FUNCtion<m> or MATH<m>.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC<m> | WMEM<r> | SBUS{1 | 2}}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":DIGitize" on page 211
- ":WAVeform:FORMat" on page 1169
- ":WAVeform:BYTeorder" on page 1165
- ":WAVeform:DATA" on page 1167
- ":WAVeform:PREamble" on page 1174

Example Code

```
' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString ":WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'myScope.WriteString ":WAVEFORM:FORMAT BYTE"
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with
'                            x-origin.
'    YINCREMENT    : float32 - voltage diff between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin
'                            occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String
```

```
myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'            FormatNumber(dblXIncrement * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'            FormatNumber(dblXOrigin * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'            CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'            FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'            FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'            CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
            FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
            FormatNumber((lngVSteps / 2 - lngYReference) * _
            sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
            FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
            FormatNumber(((lngPoints / 2 - lngXReference) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
```

```
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1)   ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)   ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) _
        * sngYIncrement + sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) _
        * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:WAVeform:SOURce:SUBSource

(see page 1354)

Command Syntax

```
:WAVeform:SOURce:SUBSource <subsource>

<subsource> ::= {{SUB0 | RX | MOSI | FAST}
                 | {SUB1 | TX | MISO | SLOW}}
```

If the :WAVeform:SOURce is SBUS<n> (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

When using UART serial decode, this option lets you get "TX" data. (TX is an alias for SUB1.) The default, SUB0, specifies "RX" data. (RX is an alias for SUB0.)

When using SPI serial decode, this option lets you get "MISO" data. (MISO is an alias for SUB1.) The default, SUB0, specifies "MOSI" data. (MOSI is an alias for SUB0.)

When using SENT serial decode, this option lets you get "SLOW" data. (SLOW is an alias for SUB1.) The default, SUB0, specifies "FAST" data. (FAST is an alias for SUB0.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS<n>:MODE is not UART, SPI, or SENT, the only valid subsource is SUB0.

Query Syntax

```
:WAVeform:SOURce:SUBSource?
```

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

Return Format

```
<subsource><NL>

<subsource> ::= {SUB0 | SUB1}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:SOURce" on page 1179

:WAVeform:TYPE

(see page 1354)

Query Syntax `:WAVeform:TYPE?`

The :WAVeform:TYPE? query returns the acquisition mode associated with the currently selected waveform. The acquisition mode is set by the :ACQuire:TYPE command.

Return Format `<mode><NL>`

`<mode> ::= {NORM | PEAK | AVER | HRES}`

NOTE If the :WAVeform:SOURce is POD1, POD2, or SBUS1, SBUS2, the type is always NORM.

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":ACQuire:TYPE" on page 257
- ":WAVeform:DATA" on page 1167
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:SOURce" on page 1179

:WAVeform:UNSigned

(see page 1354)

Command Syntax

```
:WAVeform:UNSigned <unsigned>

<unsigned> ::= {{0 | OFF} | {1 | ON}}
```

The :WAVeform:UNSigned command turns unsigned mode on or off for the currently selected waveform. Use the WAVeform:UNSigned command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

If :WAVeform:SOURce is set to POD1, POD2, BUS1, or BUS2, WAVeform:UNSigned must be set to ON.

Query Syntax

```
:WAVeform:UNSigned?
```

The :WAVeform:UNSigned? query returns the status of unsigned mode for the currently selected waveform.

Return Format

```
<unsigned><NL>

<unsigned> ::= {0 | 1}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:SOURce" on page 1179

:WAVeform:VIEW

 (see page 1354)

Command Syntax

```
:WAVeform:VIEW <view>

<view> ::= {MAIN}
```

The :WAVeform:VIEW command sets the view setting associated with the currently selected waveform. Currently, the only legal value for the view setting is MAIN.

Query Syntax

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the view setting associated with the currently selected waveform.

Return Format

```
<view><NL>

<view> ::= {MAIN}
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:POINts" on page 1170

:WAVeform:XINCrement

 (see page 1354)

Query Syntax

```
:WAVeform:XINCrement?
```

The :WAVeform:XINCrement? query returns the x-increment value for the currently specified source. This value is the time difference between consecutive data points in seconds.

Return Format

```
<value><NL>

<value> ::= x-increment in the current preamble in 64-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174

Example Code

- "Example Code" on page 1175

:WAVeform:XORigin

 (see page 1354)

Query Syntax

```
:WAVeform:XORigin?
```

The :WAVeform:XORigin? query returns the x-origin value for the currently specified source. XORigin is the X-axis value of the data point specified by the :WAVeform:XREFerence value. In this product, that is always the X-axis value of the first data point (XREFerence = 0).

Return Format

```
<value><NL>

<value> ::= x-origin value in the current preamble in 64-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:XREFerence" on page 1189

Example Code

- "Example Code" on page 1175

:WAVeform:XREFerence

 (see page 1354)

Query Syntax

```
:WAVeform:XREFerence?
```

The :WAVeform:XREFerence? query returns the x-reference value for the currently specified source. This value specifies the index of the data point associated with the x-origin data value. In this product, the x-reference point is the first point displayed and XREFerence is always 0.

Return Format

```
<value><NL>
```

```
<value> ::= x-reference value = 0 in 32-bit NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:XORigin" on page 1188

Example Code

- "Example Code" on page 1175

:WAVeform:YINCrement

 (see page 1354)

Query Syntax

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values.

Return Format

```
<value><NL>

<value> ::= y-increment value in the current preamble in 32-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174

Example Code

- "Example Code" on page 1175

:WAVeform:YORigin

 (see page 1354)

Query Syntax

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin value for the currently specified source. This value is the Y-axis value of the data value specified by the :WAVeform:YREFerence value. For this product, this is the Y-axis value of the center of the screen.

Return Format

```
<value><NL>

<value> ::= y-origin in the current preamble in 32-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:YREFerence" on page 1192

Example Code

- "Example Code" on page 1175

:WAVeform:YREFerence

 (see page 1354)

Query Syntax

```
:WAVeform:YREFerence?
```

The :WAVeform:YREFerence? query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCii.

Return Format

```
<value><NL>

<value> ::= y-reference value in the current preamble in 32-bit
            NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 1160
- ":WAVeform:PREamble" on page 1174
- ":WAVeform:YORigin" on page 1191

Example Code

- "Example Code" on page 1175

33 :WGEN<w> Commands

When the built-in waveform generator is licensed (WAVEGEN license), you can use it to output sine, square, ramp, pulse, DC, noise, sine cardinal, exponential rise, exponential fall, cardiac, and gaussian pulse waveforms. The :WGEN<w> commands are used to select the waveform function and parameters. See "Introduction to :WGEN<w> Commands" on page 1197.

Table 145 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BYTeorder <order> (see page 1198)	:WGEN<w>:ARBitrary:BYTeorder? (see page 1198)	<order> ::= {MSBFirst | LSBFirst} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 1199)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 to (# WaveGen outputs) in NR1 format
n/a	:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts? (see page 1202)	<points> ::= number of points in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:DATA:CLEar (see page 1203)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 1204)	n/a	<binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format <value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 1205)	:WGEN<w>:ARBitrary:INTerpolate? (see page 1205)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:ARBitrary:STORe <source> (see page 1206)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | FFT | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <m> ::= 1 to (# math functions) in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FREQuency <frequency> (see page 1207)	:WGEN<w>:FREQuency? (see page 1207)	<frequency> ::= frequency in Hz in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion <signal> (see page 1208)	:WGEN<w>:FUNCtion? (see page 1211)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:PULSe:WIDTh <width> (see page 1212)	:WGEN<w>:FUNCtion:PULSe:WIDTh? (see page 1212)	<width> ::= pulse width in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:FUNCtion:RAMP:SYMMetry <percent> (see page 1213)	:WGEN<w>:FUNCtion:RAMP:SYMMetry? (see page 1213)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent> (see page 1214)	:WGEN<w>:FUNCtion:SQUare:DCYCle? (see page 1214)	<percent> ::= duty cycle percentage from 20% to 80% in NR1 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:MODulation:AM:DEPTh <percent> (see page 1215)	:WGEN<w>:MODulation:AM:DEPTh? (see page 1215)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:AM:FREQuency <frequency> (see page 1216)	:WGEN<w>:MODulation:AM:FREQuency? (see page 1216)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:DEViation <frequency> (see page 1217)	:WGEN<w>:MODulation:FM:DEViation? (see page 1217)	<frequency> ::= frequency deviation in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FM:FREQuency <frequency> (see page 1218)	:WGEN<w>:MODulation:FM:FREQuency? (see page 1218)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FSKey:FREQuency <percent> (see page 1219)	:WGEN<w>:MODulation:FSKey:FREQuency? (see page 1219)	<frequency> ::= hop frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FSKey:RATE <rate> (see page 1220)	:WGEN<w>:MODulation:FSKey:RATE? (see page 1220)	<rate> ::= FSK modulation rate in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion <shape> (see page 1221)	:WGEN<w>:MODulation:FUNCtion? (see page 1221)	<shape> ::= {SINusoid | SQUare| RAMP} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry <percent> (see page 1222)	:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry? (see page 1222)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:MODulation:NOISe <percent> (see page 1223)	:WGEN<w>:MODulation:NOISe? (see page 1223)	<percent> ::= 0 to 100 <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:STATe {{0 | OFF} | {1 | ON}} (see page 1224)	:WGEN<w>:MODulation:STATe? (see page 1224)	{0 | 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:TYPE <type> (see page 1225)	:WGEN<w>:MODulation:TYPE? (see page 1225)	<type> ::= {AM | FM | FSK} <w> ::= 1 in NR1 format
:WGEN<w>:OUTPut {{0 | OFF} | {1 | ON}} (see page 1227)	:WGEN<w>:OUTPut? (see page 1227)	{0 | 1} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1228)	:WGEN<w>:OUTPut:LOAD? (see page 1228)	<impedance> ::= {ONEMeg | FIFTy} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:MODE <mode> (see page 1229)	:WGEN<w>:OUTPut:MODE? (see page 1229)	<mode> ::= {NORMal | SINGle}
:WGEN<w>:OUTPut:POLarity <polarity> (see page 1230)	:WGEN<w>:OUTPut:POLarity? (see page 1230)	<polarity> ::= {NORMal | INVerted} <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:OUTPut:SINGle (see page 1231)	n/a	n/a
:WGEN<w>:PERiod <period> (see page 1232)	:WGEN<w>:PERiod? (see page 1232)	<period> ::= period in seconds in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:RST (see page 1233)	n/a	<w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage <amplitude> (see page 1234)	:WGEN<w>:VOLTage? (see page 1234)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1235)	:WGEN<w>:VOLTage:HIGH? (see page 1235)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:VOLTage:LOW <low> (see page 1236)	:WGEN<w>:VOLTage:LOW? (see page 1236)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1237)	:WGEN<w>:VOLTage:OFFSet? (see page 1237)	<offset> ::= offset in volts in NR3 format <w> ::= 1 to (# WaveGen outputs) in NR1 format

Introduction to :WGEN<w> Commands

The :WGEN<w> subsystem provides commands to select the waveform generator function and parameters.

In the :WGEN<w> commands, the <w> can be 1 or 2, and :WGEN is equivalent to :WGEN1

Reporting the Setup

Use :WGEN<w>? to query setup information for the WGEN<w> subsystem.

Return Format

The following is a sample response from the :WGEN? query. In this case, the query was issued following the *RST command.

```
:WGEN1:FUNC SIN;OUTP 0;FREQ +1.0000E+03;VOLT +500.0E-03;VOLT:OFFS
+0.0E+00;:WGEN1:OUTP:LOAD ONEM
```

:WGEN<w>:ARBitrary:BYTeorder

 (see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:BYTeorder <order>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<order> ::= {MSBFirst | LSBFirst}
```

The :WGEN<w>:ARBitrary:BYTeorder command selects the byte order for binary transfers.

Query Syntax

```
:WGEN<w>:ARBitrary:BYTeorder?
```

The :WGEN<w>:ARBitrary:BYTeorder query returns the current byte order selection.

Return Format

```
<order><NL>

<order> ::= {MSBFirst | LSBFirst}
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204

:WGEN<w>:ARBitrary:DATA

N (see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...}

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<binary> ::= floating point values between -1.0 to +1.0
             in IEEE 488.2 binary block format

<value> ::= floating point values between -1.0 to +1.0
            in comma-separated format
```

The :WGEN<w>:ARBitrary:DATA command downloads an arbitrary waveform in floating-point values format.

See Also

- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

Example Code

```
' Waveform generator arbitrary data commands example.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" ( _
  dest As Any, _
  source As Any, _
  ByVal bytes As Long)

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::a-mx4154a-60014.cos.is.keysight.com::inst0::
INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Turn on arbitrary waveform generator function.
  myScope.WriteString ":WGEN1:OUTPut ON"
  myScope.WriteString ":WGEN1:FUNCtion ARBitrary"
  myScope.WriteString ":WGEN1:FUNCtion?"
  strQueryResult = myScope.ReadString
  Debug.Print "WaveGen1 function: " + strQueryResult
```

```
  DefaultArbitraryWaveform

  ' Download comma-separated floating-point values.
  myScope.WriteString ":WGEN1:ARBitrary:DATA 0.0, 0.5, 1.0, 0.5, 0.0, -0
.5, -1.0, -0.5"
  Debug.Print "WaveGen1 CSV floating-point values downloaded."
  Sleep 5000

  DefaultArbitraryWaveform

  ' Download comma-separated 16-bit integer (DAC) values.
  myScope.WriteString ":WGEN1:ARBitrary:DATA:DAC 0, 255, 511, 255, 0, -2
56, -512, -256"
  Debug.Print "WaveGen1 CSV 16-bit integer (DAC) values downloaded."
  Sleep 5000

  ' Set the byte order for binary data.
  myScope.WriteString ":WGEN1:ARBitrary:BYTeorder LSBFirst"
  myScope.WriteString ":WGEN1:ARBitrary:BYTeorder?"
  strQueryResult = myScope.ReadString
  Debug.Print "WaveGen1 byte order for binary data: " + strQueryResult

  DefaultArbitraryWaveform

  ' Download binary floating-point values.
  Dim mySingleArray(8) As Single
  mySingleArray(0) = 0!
  mySingleArray(1) = 0.5!
  mySingleArray(2) = 1!
  mySingleArray(3) = 0.5!
  mySingleArray(4) = 0!
  mySingleArray(5) = -0.5!
  mySingleArray(6) = -1!
  mySingleArray(7) = -0.5!

  Dim myByteArray(32) As Byte
  CopyMemory myByteArray(0), mySingleArray(0), 32 * LenB(myByteArray(0))

  myScope.WriteIEEEBlock ":WGEN1:ARBitrary:DATA", myByteArray, True
  Debug.Print "WaveGen1 binary floating-point values downloaded."
  Sleep 5000

  DefaultArbitraryWaveform

  ' Download binary 16-bit integer (DAC) values.
  Dim myIntegerArray(8) As Integer
  myIntegerArray(0) = 0
  myIntegerArray(1) = 255
  myIntegerArray(2) = 511
  myIntegerArray(3) = 255
  myIntegerArray(4) = 0
  myIntegerArray(5) = -256
  myIntegerArray(6) = -512
  myIntegerArray(7) = -256

  Dim myByteArray2(16) As Byte
```

```
  CopyMemory myByteArray2(0), myIntegerArray(0), 16 * LenB(myByteArray2(
0))

  myScope.WriteIEEEBlock ":WGEN1:ARBitrary:DATA:DAC", myByteArray2, True
  Debug.Print "WaveGen1 binary 16-bit integer (DAC) values downloaded."
  Sleep 5000

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Initialize WaveGen1 to a known state.
' --------------------------------------------------------------------
Private Sub DefaultArbitraryWaveform()

  On Error GoTo VisaComError

  ' Load default arbitrary waveform.
  myScope.WriteString ":WGEN1:ARBitrary:DATA:CLEar"
  Debug.Print "WaveGen1 default arbitrary waveform loaded."
  Sleep 5000

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub
```

:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts

N (see page 1354)

Query Syntax

```
:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts?

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:DATA:ATTRibute:POINts query returns the number of points used by the current arbitrary waveform.

Return Format

```
<points> ::= number of points in NR1 format
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

:WGEN<w>:ARBitrary:DATA:CLEar

 (see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA:CLEar

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:DATA:CLEar command clears the arbitrary waveform memory and loads it with the default waveform.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

Example Code

- "Example Code" on page 1199

:WGEN<w>:ARBitrary:DATA:DAC

(see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...}

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<binary> ::= decimal 16-bit integer values between -512 to +511
             in IEEE 488.2 binary block format

<value> ::= decimal integer values between -512 to +511
            in comma-separated NR1 format
```

The :WGEN<w>:ARBitrary:DATA:DAC command downloads an arbitrary waveform using 16-bit integer (DAC) values.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

Example Code

- "Example Code" on page 1199

:WGEN<w>:ARBitrary:INTerpolate

 (see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}}

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:ARBitrary:INTerpolate command enables or disables the Interpolation control.

Interpolation specifies how lines are drawn between arbitrary waveform points:

- When ON, lines are drawn between points in the arbitrary waveform. Voltage levels change linearly between one point and the next.
- When OFF, all line segments in the arbitrary waveform are horizontal. The voltage level of one point remains until the next point.

Query Syntax

```
:WGEN<w>:ARBitrary:INTerpolate?
```

The :WGEN<w>:ARBitrary:INTerpolate query returns the current interpolation setting.

Return Format

```
{0 | 1}
```

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

:WGEN<w>:ARBitrary:STORe

N (see page 1354)

Command Syntax

```
:WGEN<w>:ARBitrary:STORe <source>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | MATH<m>}

<n> ::= 1 to (# analog channels) in NR1 format

<r> ::= 1 to (# ref waveforms) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format
```

The :WGEN<w>:ARBitrary:STORe command stores the source's waveform into the arbitrary waveform memory.

See Also

- ":WGEN<w>:ARBitrary:DATA" on page 1199
- ":WGEN<w>:ARBitrary:DATA:DAC" on page 1204
- ":SAVE:ARBitrary[:STARt]" on page 695
- ":RECall:ARBitrary[:STARt]" on page 683

:WGEN<w>:FREQuency

(see page 1354)

Command Syntax

```
:WGEN<w>:FREQuency <frequency>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<frequency> ::= frequency in Hz in NR3 format
```

For all waveforms except Noise and DC, the :WGEN<w>:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN<w>:PERiod command.

Query Syntax

```
:WGEN<w>:FREQuency?
```

The :WGEN<w>:FREQuency? query returns the currently set waveform generator frequency.

Return Format

```
<frequency><NL>

<frequency> ::= frequency in Hz in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:PERiod" on page 1232

:WGEN<w>:FUNCtion

N (see page 1354)

Command Syntax

```
:WGEN<w>:FUNCtion <signal>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<signal> ::= {SINusoid | SQUare | RAMP | PULSe | DC | NOISe | SINC
             | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary}
```

The :WGEN<w>:FUNCtion command selects the type of waveform:

Waveform Type	Characteristics	Frequency Range	Max. Amplitude[2] (High-Z)[1]	Offset[2] (High-Z)[1]
SINusoid	Use these commands to set the sine signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236	100 mHz to 20 MHz	20 mVpp to 10 Vpp	±4.00 V
SQUare	Use these commands to set the square wave signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236 ▪ ":WGEN<w>:FUNCtion:SQUare:DCYCle" on page 1214 The duty cycle can be adjusted from 20% to 80%.	100 mHz to 10 MHz	20 mVpp to 10 Vpp	±5.00 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude[2] (High-Z)[1]	Offset[2] (High-Z)[1]
RAMP	Use these commands to set the ramp signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236 ▪ ":WGEN<w>:FUNCtion:RAMP:SYMMetry" on page 1213 Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.	100 mHz to 200 kHz	20 mVpp to 10 Vpp	±5.00 V
PULSe	Use these commands to set the pulse signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236 ▪ ":WGEN<w>:FUNCtion:PULSe:WIDTh" on page 1212 The pulse width can be adjusted from 20 ns to the period minus 20 ns.	100 mHz to 10 MHz.	20 mVpp to 10 Vpp	±5.00 V
DC	Use this command to set the DC level: ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237	n/a	n/a	±10.00 V
NOISe	Use these commands to set the noise signal parameters: ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236	n/a	20 mVpp to 10 Vpp	±5.00 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude[2] (High-Z)[1]	Offset[2] (High-Z)[1]
SINC	Use these commands to set the sine cardinal signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237	100 mHz to 1 MHz	20 mVpp to 9 Vpp	±2.50 V
EXPRise	Use these commands to set the exponential rise signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236	100 mHz to 5 MHz	20 mVpp to 10 Vpp	±5.00 V
EXPFall	Use these commands to set the exponential fall signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236	100 mHz to 5 MHz	20 mVpp to 10 Vpp	±5.00 V
CARDiac	Use these commands to set the cardiac signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237	100 mHz to 200 kHz	20 mVpp to 9 Vpp	±2.50 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude[2] (High-Z)[1]	Offset[2] (High-Z)[1]
GAUSsian	Use these commands to set the gaussian pulse signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237	100 mHz to 5 MHz	20 mVpp to 7.5 Vpp	±2.50 V
ARBitrary	Use these commands to set the arbitrary signal parameters: ▪ ":WGEN<w>:FREQuency" on page 1207 ▪ ":WGEN<w>:PERiod" on page 1232 ▪ ":WGEN<w>:VOLTage" on page 1234 ▪ ":WGEN<w>:VOLTage:OFFSet" on page 1237 ▪ ":WGEN<w>:VOLTage:HIGH" on page 1235 ▪ ":WGEN<w>:VOLTage:LOW" on page 1236	100 mHz to 12 MHz	20 mVpp to 10 Vpp	±5.00 V

[1]When the output load is 50 Ω, these values are halved.

[2]The minimum amplitude is limited to 40 mVpp if the offset is greater than 500 mV or less than -500 mV. Likewise, the offset is limited to +/-500 mV if the amplitude is less than 40 mVpp.

Query Syntax

```
:WGEN<w>:FUNCtion?
```

The :WGEN<w>:FUNCtion? query returns the currently selected signal type.

Return Format

```
<signal><NL>

<signal> ::= {SIN | SQU | RAMP | PULS | DC | NOIS | SINC | EXPR | EXPF
             | CARD | GAUS | ARB}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:MODulation:NOISe" on page 1223

:WGEN<w>:FUNCtion:PULSe:WIDTh

 (see page 1354)

Command Syntax

```
:WGEN<w>:FUNCtion:PULSe:WIDTh <width>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<width> ::= pulse width in seconds in NR3 format
```

For Pulse waveforms, the :WGEN<w>:FUNCtion:PULSe:WIDTh command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

Query Syntax

```
:WGEN<w>:FUNCtion:PULSe:WIDTh?
```

The :WGEN<w>:FUNCtion:PULSe:WIDTh? query returns the currently set pulse width.

Return Format

```
<width><NL>

<width> ::= pulse width in seconds in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208

:WGEN<w>:FUNCtion:RAMP:SYMMetry

(see page 1354)

Command Syntax

```
:WGEN<w>:FUNCtion:RAMP:SYMMetry <percent>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

For Ramp waveforms, the :WGEN<w>:FUNCtion:RAMP:SYMMetry command specifies the symmetry of the waveform.

Symmetry represents the amount of time per cycle that the ramp waveform is rising.

Query Syntax

```
:WGEN<w>:FUNCtion:RAMP:SYMMetry?
```

The :WGEN<w>:FUNCtion:RAMP:SYMMetry? query returns the currently set ramp symmetry.

Return Format

```
<percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208

:WGEN<w>:FUNCtion:SQUare:DCYCle

 (see page 1354)

Command Syntax

```
:WGEN<w>:FUNCtion:SQUare:DCYCle <percent>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

For Square waveforms, the :WGEN<w>:FUNCtion:SQUare:DCYCle command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query Syntax

```
:WGEN<w>:FUNCtion:SQUare:DCYCle?
```

The :WGEN<w>:FUNCtion:SQUare:DCYCle? query returns the currently set square wave duty cycle.

Return Format

```
<percent><NL>

<percent> ::= duty cycle percentage from 20% to 80% in NR1 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208

:WGEN<w>:MODulation:AM:DEPTh

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:AM:DEPTh <percent>

<w> ::= 1 in NR1 format

<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

The :WGEN<w>:MODulation:AM:DEPTh command specifies the amount of amplitude modulation.

AM Depth refers to the portion of the amplitude range that will be used by the modulation. For example, a depth setting of 80% causes the output amplitude to vary from 10% to 90% (90% – 10% = 80%) of the original amplitude as the modulating signal goes from its minimum to maximum amplitude.

Query Syntax

```
:WGEN<w>:MODulation:AM:DEPTh?
```

The :WGEN<w>:MODulation:AM:DEPTh? query returns the AM depth percentage setting.

Return Format

```
<percent><NL>

<percent> ::= AM depth percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:AM:FREQuency

(see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:AM:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN<w>:MODulation:AM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax

```
:WGEN<w>:MODulation:AM:FREQuency?
```

The :WGEN<w>:MODulation:AM:FREQuency? query returns the frequency of the modulating signal.

Return Format

```
<frequency><NL>

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FM:DEViation

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FM:DEViation <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= frequency deviation in Hz in NR3 format
```

The :WGEN<w>:MODulation:FM:DEViation command specifies the frequency deviation from the original carrier signal frequency.

When the modulating signal is at its maximum amplitude, the output frequency is the carrier signal frequency plus the deviation amount, and when the modulating signal is at its minimum amplitude, the output frequency is the carrier signal frequency minus the deviation amount.

The frequency deviation cannot be greater than the original carrier signal frequency.

Also, the sum of the original carrier signal frequency and the frequency deviation must be less than or equal to the maximum frequency for the selected waveform generator function plus 100 kHz.

Query Syntax

```
:WGEN<w>:MODulation:FM:DEViation?
```

The :WGEN<w>:MODulation:FM:DEViation? query returns the frequency deviation setting.

Return Format

```
<frequency><NL>

<frequency> ::= frequency deviation in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FM:FREQuency

(see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FM:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

The :WGEN<w>:MODulation:FM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax

```
:WGEN<w>:MODulation:FM:FREQuency?
```

The :WGEN<w>:MODulation:FM:FREQuency? query returns the frequency of the modulating signal.

Return Format

```
<frequency><NL>

<frequency> ::= modulating waveform frequency in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FSKey:FREQuency

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FSKey:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= hop frequency in Hz in NR3 format
```

The :WGEN<w>:MODulation:FSKey:FREQuency command specifies the "hop frequency".

The output frequency "shifts" between the original carrier frequency and this "hop frequency".

Query Syntax

```
:WGEN<w>:MODulation:FSKey:FREQuency?
```

The :WGEN<w>:MODulation:FSKey:FREQuency? query returns the "hop frequency" setting.

Return Format

```
<frequency><NL>

<frequency> ::= hop frequency in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FSKey:RATE

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FSKey:RATE <rate>

<w> ::= 1 in NR1 format

<rate> ::= FSK modulation rate in Hz in NR3 format
```

The :WGEN<w>:MODulation:FSKey:RATE command specifies the rate at which the output frequency "shifts".

The FSK rate specifies a digital square wave modulating signal.

Query Syntax

```
:WGEN<w>:MODulation:FSKey:RATE?
```

The :WGEN<w>:MODulation:FSKey:RATE? query returns the FSK rate setting.

Return Format

```
<rate><NL>

<rate> ::= FSK modulation rate in Hz in NR3 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FUNCtion

(see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FUNCtion <shape>

<w> ::= 1 in NR1 format

<shape> ::= {SINusoid | SQUare| RAMP}
```

The :WGEN<w>: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<w>:MODulation:FUNCtion:RAMP:SYMMetry command.

This command applies to AM and FM modulation. (The FSK modulation signal is a square wave shape.)

Query Syntax

```
:WGEN<w>:MODulation:FUNCtion?
```

The :WGEN<w>:MODulation:FUNCtion? query returns the specified modulating signal shape.

Return Format

```
<shape><NL>

<shape> ::= {SIN | SQU| RAMP}
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry

 (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry <percent>

<w> ::= 1 in NR1 format

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

The :WGEN<w>: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<w>:MODulation:FUNCtion command.

Query Syntax

```
:WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry?
```

The :WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry? query returns ramp symmetry percentage setting.

Return Format

```
<percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR1 format
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:STATe" on page 1224
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:NOISe

(see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:NOISe <percent>

<w> ::= 1 in NR1 format

<percent> ::= 0 to 100
```

The :WGEN<w>:MODulation:NOISe command adds noise to the currently selected signal. The sum of the amplitude between the original signal and injected noise is limited to the regular amplitude limit (for example, 5 Vpp in 1 MOhm), so the range for <percent> varies according to current amplitude.

Note that adding noise affects edge triggering on the waveform generator source as well as the waveform generator sync pulse output signal (which can be sent to TRIG OUT). This is because the trigger comparator is located after the noise source.

Query Syntax

```
:WGEN<w>:MODulation:NOISe?
```

The :WGEN<w>:MODulation:NOISe query returns the percent of added noise.

Return Format

```
<percent><NL>

<percent> ::= 0 to 100
```

See Also

- ":WGEN<w>:FUNCtion" on page 1208

:WGEN<w>:MODulation:STATe

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:STATe <setting>

<w> ::= 1 in NR1 format

<setting> ::= {{OFF | 0} | {ON | 1}}
```

The :WGEN<w>:MODulation:STATe command enables or disables modulated waveform generator output.

You can enable modulation for all waveform generator function types except pulse, DC, and noise.

Query Syntax

```
:WGEN<w>:MODulation:STATe?
```

The :WGEN<w>:MODulation:STATe? query returns whether the modulated waveform generator output is enabled of disabled.

Return Format

```
<setting><NL>

<setting> ::= {0 | 1}
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218
- ":WGEN<w>:MODulation:FSKey:FREQuency" on page 1219
- ":WGEN<w>:MODulation:FSKey:RATE" on page 1220
- ":WGEN<w>:MODulation:FUNCtion" on page 1221
- ":WGEN<w>:MODulation:FUNCtion:RAMP:SYMMetry" on page 1222
- ":WGEN<w>:MODulation:TYPE" on page 1225

:WGEN<w>:MODulation:TYPE

N (see page 1354)

Command Syntax

```
:WGEN<w>:MODulation:TYPE <type>

<w> ::= 1 in NR1 format

<type> ::= {AM | FM | FSK}
```

The :WGEN<w>:MODulation:TYPE command selects the modulation type:

- AM (amplitude modulation) — the amplitude of the original carrier signal is modified according to the amplitude of the modulating signal.

  Use the :WGEN<w>:MODulation:AM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN<w>:MODulation:AM:DEPTh command to specify the amount of amplitude modulation.
- FM (frequency modulation) — the frequency of the original carrier signal is modified according to the amplitude of the modulating signal.

  Use the :WGEN<w>:MODulation:FM:FREQuency command to set the modulating signal frequency.

  Use the :WGEN<w>:MODulation:FM:DEViation command to specify the frequency deviation from the original carrier signal frequency.
- 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<w>:MODulation:FSKey:FREQuency command to specify the "hop frequency".

  Use the :WGEN<w>:MODulation:FSKey:RATE command to specify the rate at which the output frequency "shifts".

Query Syntax

```
:WGEN<w>:MODulation:TYPE?
```

The :WGEN<w>:MODulation:TYPE? query returns the selected modulation type.

Return Format

```
<type><NL>

<type> ::= {AM | FM | FSK}
```

See Also

- ":WGEN<w>:MODulation:AM:DEPTh" on page 1215
- ":WGEN<w>:MODulation:AM:FREQuency" on page 1216
- ":WGEN<w>:MODulation:FM:DEViation" on page 1217
- ":WGEN<w>:MODulation:FM:FREQuency" on page 1218

:WGEN<w>:OUTPut

 (see page 1354)

Command Syntax

```
:WGEN<w>:OUTPut <on_off>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<on_off> ::= {{1 | ON} | {0 | OFF}
```

The :WGEN<w>:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query Syntax

```
:WGEN<w>:OUTPut?
```

The :WGEN<w>:OUTPut? query returns the current state of the waveform generator output setting.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197

:WGEN<w>:OUTPut:LOAD

 (see page 1354)

Command Syntax

```
:WGEN<w>:OUTPut:LOAD <impedance>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<impedance> ::= {ONEMeg | FIFTy}
```

The :WGEN<w>:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax

```
:WGEN<w>:OUTPut:LOAD?
```

The :WGEN<w>:OUTPut:LOAD? query returns the current expected output load impedance.

Return Format

```
<impedance><NL>

<impedance> ::= {ONEM | FIFT}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197

:WGEN<w>:OUTPut:MODE

(see page 1354)

Command Syntax

```
:WGEN<w>:OUTPut:MODE <mode>

<mode> ::= {NORMal | SINGle}
```

The :WGEN<w>: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<w>:OUTPut:SINGle command.

  Not all waveform types allow single-shot output.

Query Syntax

```
:WGEN<w>:OUTPut:MODE?
```

The :WGEN<w>:OUTPut:MODE? query returns the output mode setting.

Return Format

```
<mode><NL>

<mode> ::= {NORMal | SINGle}
```

See Also

- ":WGEN<w>:OUTPut:SINGle" on page 1231

:WGEN<w>:OUTPut:POLarity

(see page 1354)

Command Syntax

```
:WGEN<w>:OUTPut:POLarity <polarity>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<polarity> ::= {NORMal | INVerted}
```

The :WGEN<w>:OUTPut:POLarity command specifies whether the waveform generator output is inverted..

Query Syntax

```
:WGEN<w>:OUTPut:POLarity?
```

The :WGEN<w>:OUTPut:POLarity? query returns the specified output polarity.

Return Format

```
<polarity><NL>

<polarity> ::= {NORM | INV}
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197

:WGEN<w>:OUTPut:SINGle

(see page 1354)

Command Syntax `:WGEN<w>:OUTPut:SINGle`

When the single-shot output mode is selected (by the :WGEN<w>:OUTPut:MODE command), the :WGEN<w>: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<w>:OUTPut:MODE" on page 1229

:WGEN<w>:PERiod

(see page 1354)

Command Syntax

```
:WGEN<w>:PERiod <period>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<period> ::= period in seconds in NR3 format
```

For all waveforms except Noise and DC, the :WGEN<w>:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN<w>:FREQuency command.

Query Syntax

```
:WGEN<w>:PERiod?
```

The :WGEN<w>:PERiod? query returns the currently set waveform generator period.

Return Format

```
<period><NL>

<period> ::= period in seconds in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:FREQuency" on page 1207

:WGEN<w>:RST

(see page 1354)

Command Syntax

```
:WGEN<w>:RST

<w> ::= 1 to (# WaveGen outputs) in NR1 format
```

The :WGEN<w>:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:FREQuency" on page 1207

:WGEN<w>:VOLTage

(see page 1354)

Command Syntax

```
:WGEN<w>:VOLTage <amplitude>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<amplitude> ::= amplitude in volts in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage command specifies the waveform's amplitude. Use the :WGEN<w>:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage?
```

The :WGEN<w>:VOLTage? query returns the currently specified waveform amplitude.

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:VOLTage:OFFSet" on page 1237
- ":WGEN<w>:VOLTage:HIGH" on page 1235
- ":WGEN<w>:VOLTage:LOW" on page 1236

:WGEN<w>:VOLTage:HIGH

N (see page 1354)

Command Syntax

```
:WGEN<w>:VOLTage:HIGH <high>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<high> ::= high-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN<w>:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:HIGH?
```

The :WGEN<w>:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

Return Format

```
<high><NL>

<high> ::= high-level voltage in volts, in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:VOLTage:LOW" on page 1236
- ":WGEN<w>:VOLTage" on page 1234
- ":WGEN<w>:VOLTage:OFFSet" on page 1237

:WGEN<w>:VOLTage:LOW

N (see page 1354)

Command Syntax

```
:WGEN<w>:VOLTage:LOW <low>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<low> ::= low-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN<w>:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN<w>:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:LOW?
```

The :WGEN<w>:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

Return Format

```
<low><NL>

<low> ::= low-level voltage in volts, in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:VOLTage:LOW" on page 1236
- ":WGEN<w>:VOLTage" on page 1234
- ":WGEN<w>:VOLTage:OFFSet" on page 1237

:WGEN<w>:VOLTage:OFFSet

(see page 1354)

Command Syntax

```
:WGEN<w>:VOLTage:OFFSet <offset>

<w> ::= 1 to (# WaveGen outputs) in NR1 format

<offset> ::= offset in volts in NR3 format
```

The :WGEN<w>:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN<w>:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

NOTE When the amplitude is below 40 mV, the offset is limited to ±500 mV.

Query Syntax

```
:WGEN<w>:VOLTage:OFFSet?
```

The :WGEN<w>:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

Return Format

```
<offset><NL>

<offset> ::= offset in volts in NR3 format
```

See Also

- "Introduction to :WGEN<w> Commands" on page 1197
- ":WGEN<w>:FUNCtion" on page 1208
- ":WGEN<w>:VOLTage" on page 1234
- ":WGEN<w>:VOLTage:HIGH" on page 1235
- ":WGEN<w>:VOLTage:LOW" on page 1236

34 :WMEMory<r> Commands

Control reference waveforms.

Table 146 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1241)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 1242)	:WMEMory<r>:DISPlay? (see page 1242)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 1243)	:WMEMory<r>:LABel? (see page 1243)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 1244)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1 to (# math functions) in NR1 format NOTE: Math functions whose x-axis is not frequency can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1245)	:WMEMory<r>:SKEW? (see page 1245)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format

Table 146 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YOFFset <offset>[suffix] (see page 1246)	:WMEMory<r>:YOFFset? (see page 1246)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YRANge <range>[suffix] (see page 1247)	:WMEMory<r>:YRANge? (see page 1247)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YSCale <scale>[suffix] (see page 1248)	:WMEMory<r>:YSCale? (see page 1248)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V | mV}

:WMEMory<r>:CLEar

N (see page 1354)

Command Syntax

```
:WMEMory<r>:CLEar

<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :WMEMory<r>:CLEar command clears the specified reference waveform location.

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:SAVE" on page 1244
- ":WMEMory<r>:DISPlay" on page 1242

:WMEMory<r>:DISPlay

N (see page 1354)

Command Syntax

```
:WMEMory<r>:DISPlay <on_off>

<r> ::= 1 to (# ref waveforms) in NR1 format

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :WMEMory<r>:DISPlay command turns the display of the specified reference waveform on or off.

There are two reference waveform locations, but only one reference waveform can be displayed at a time. That means, if :WMEMory1:DISPlay is ON, sending the :WMEMory2:DISPlay ON command will automatically set :WMEMory1:DISPlay OFF.

Query Syntax

```
:WMEMory<r>:DISPlay?
```

The :WMEMory<r>:DISPlay? query returns the current display setting for the reference waveform.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:CLEar" on page 1241
- ":WMEMory<r>:LABel" on page 1243

:WMEMory<r>:LABel

 (see page 1354)

Command Syntax

```
:WMEMory<r>:LABel <string>

<r> ::= 1 to (# ref waveforms) in NR1 format

<string> ::= quoted ASCII string
```

NOTE Label strings are 10 characters or less, and may contain any commonly used ASCII characters. Labels with more than 10 characters are truncated to 10 characters. Lower case characters are converted to upper case.

The :WMEMory<r>:LABel command sets the reference waveform label to the string that follows.

Setting a label for a reference waveform also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax

```
:WMEMory<r>:LABel?
```

The :WMEMory<r>:LABel? query returns the label associated with a particular reference waveform.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242

:WMEMory<r>:SAVE

N (see page 1354)

Command Syntax

```
:WMEMory<r>:SAVE <source>

<r> ::= 1 to (# ref waveforms) in NR1 format

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>}

<n> ::= 1 to (# analog channels) in NR1 format

<m> ::= 1 to (# math functions) in NR1 format
```

The :WMEMory<r>:SAVE command copies the analog channel or math function waveform to the specified reference waveform location.

NOTE Math functions whose x-axis is not frequency can be saved as reference waveforms.

NOTE :WMEMory<r>:SAVE is an overlapped command (see Chapter 4, “Sequential (Blocking) vs. Overlapped Commands,” starting on page 63).

See Also

- Chapter 34, “:WMEMory<r> Commands,” starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242

:WMEMory<r>:SKEW

 (see page 1354)

Command Syntax

```
:WMEMory<r>:SKEW <skew>

<r> ::= 1 to (# ref waveforms) in NR1 format

<skew> ::= time in seconds in NR3 format
```

The :WMEMory<r>:SKEW command sets the skew factor for the specified reference waveform.

Query Syntax

```
:WMEMory<r>:SKEW?
```

The :WMEMory<r>:SKEW? query returns the current skew setting for the selected reference waveform.

Return Format

```
<skew><NL>

<skew> ::= time in seconds in NR3 format
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242
- ":WMEMory<r>:YOFFset" on page 1246
- ":WMEMory<r>:YRANge" on page 1247
- ":WMEMory<r>:YSCale" on page 1248

:WMEMory<r>:YOFFset

 (see page 1354)

Command Syntax

```
:WMEMory<r>:YOFFset <offset> [<suffix>]

<r> ::= 1 to (# ref waveforms) in NR1 format

<offset> ::= vertical offset value in NR3 format

<suffix> ::= {V | mV}
```

The :WMEMory<r>:YOFFset command sets the value that is represented at center screen for the selected reference waveform.

The range of legal values varies with the value set by the :WMEMory<r>:YRANge or :WMEMory<r>:YSCale commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are affected by the probe attenuation setting.

Query Syntax

```
:WMEMory<r>:YOFFset?
```

The :WMEMory<r>:YOFFset? query returns the current offset value for the selected reference waveform.

Return Format

```
<offset><NL>

<offset> ::= vertical offset value in NR3 format
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242
- ":WMEMory<r>:YRANge" on page 1247
- ":WMEMory<r>:YSCale" on page 1248
- ":WMEMory<r>:SKEW" on page 1245

:WMEMory<r>:YRANge

 (see page 1354)

Command Syntax

```
:WMEMory<r>:YRANge <range>[<suffix>]

<r> ::= 1 to (# ref waveforms) in NR1 format

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}
```

The :WMEMory<r>:YRANge command defines the full-scale vertical axis of the selected reference waveform.

Legal values for the range are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax

```
:WMEMory<r>:YRANge?
```

The :WMEMory<r>:YRANge? query returns the current full-scale range setting for the specified reference waveform.

Return Format

```
<range><NL>

<range> ::= vertical full-scale range value in NR3 format
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242
- ":WMEMory<r>:YOFFset" on page 1246
- ":WMEMory<r>:SKEW" on page 1245
- ":WMEMory<r>:YSCale" on page 1248

:WMEMory<r>:YSCale

N (see page 1354)

Command Syntax

```
:WMEMory<r>:YSCale <scale>[<suffix>]

<r> ::= 1 to (# ref waveforms) in NR1 format

<scale> ::= vertical units per division in NR3 format

<suffix> ::= {V | mV}
```

The :WMEMory<r>:YSCale command sets the vertical scale, or units per division, of the selected reference waveform.

Legal values for the scale are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax

```
:WMEMory<r>:YSCale?
```

The :WMEMory<r>:YSCale? query returns the current scale setting for the specified reference waveform.

Return Format

```
<scale><NL>

<scale> ::= vertical units per division in NR3 format
```

See Also

- Chapter 34, ":WMEMory<r> Commands," starting on page 1239
- ":WMEMory<r>:DISPlay" on page 1242
- ":WMEMory<r>:YOFFset" on page 1246
- ":WMEMory<r>:YRANge" on page 1247
- ":WMEMory<r>:SKEW" on page 1245

35 Obsolete and Discontinued Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see"Obsolete Commands" on page 1354).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 275)	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see page 276)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 279)	
ANALog<n>:LABel	:CHANnel<n>:LABel (see page 280)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 281)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 282)	
ANALog<n>:PMODe	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see page 291)	
:CHANnel:LABel (see page 1255)	:CHANnel<n>:LABel (see page 280)	use CHANnel<n>:LABel for analog channels
:CHANnel2:SKEW (see page 1256)	:CHANnel<n>:PROBe:SKEW (see page 287)	
:CHANnel<n>:INPut (see page 1257)	:CHANnel<n>:IMPedance (see page 278)	
:CHANnel<n>:PMODe (see page 1258)	none	

Obsolete Command	Current Command Equivalent	Behavior Differences
:DISPlay:CONNect (see page 1259)	:DISPlay:VECTors (see page 332)	
:ERASe (see page 1260)	:DISplay:CLEar (see page 320)	
:EXTernal:PMODe (see page 1261)	none	
FUNCtion1, FUNCtion2	:FUNCtion Commands (see page 361)	ADD not included
:FUNCtion Commands	:FUNCtion2 Commands (see page 361)	:FUNCtion commands (with no <m> number) map to :FUNCtion2. This allows legacy programs to work without change.
:FUNCtion:GOFT:OPERation (see page 1262)	:FUNCtion1:OPERation (see page 390)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce1 (see page 1263)	:FUNCtion1:SOURce1 (see page 398)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce2 (see page 1264)	:FUNCtion1:SOURce2 (see page 400)	GOFT maps to FUNCtion1.
:FUNCtion:SOURce (see page 1265)	:FUNCtion:SOURce1 (see page 398)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNCtion:VIEW (see page 1266)	:FUNCtion:DISPlay (see page 369)	
:MEASure:LOWer (see page 1267)	:MEASure:DEFine:THResholds (see page 457)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see page 1268)	:MEASure:CLEar (see page 455)	
:MEASure:TDELta (see page 1269)	:MARKer:XDELta (see page 422)	
:MEASure:THResholds (see page 1270)	:MEASure:DEFine:THResholds (see page 457)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 1271)	:MEASure:XMAX (see page 517)	
:MEASure:TMIN (see page 1272)	:MEASure:XMIN (see page 518)	
:MEASure:TSTArt (see page 1273)	:MARKer:X1Position (see page 417)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:TSTOp (see page 1274)	:MARKer:X2Position (see page 420)	
:MEASure:TVOLt (see page 1275)	:MEASure:TVALue (see page 504)	TVALue measures additional values such as dB, Vs, etc.
:MEASure:UPPer (see page 1276)	:MEASure:DEFine:THResholds (see page 457)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 1277)	:MARKer:YDELta (see page 429)	
:MEASure:VSTArt (see page 1278)	:MARKer:Y1Position (see page 426)	
:MEASure:VSTOp (see page 1279)	:MARKer:Y2Position (see page 428)	
:MTESt:AMASk:{SAVE | STORe} (see page 1280)	:SAVE:MASK[:STARt] (see page 703)	
:MTESt:AVERage (see page 1281)	:ACQuire:TYPE AVERage (see page 257)	
:MTESt:AVERage:COUNt (see page 1282)	:ACQuire:COUNt (see page 246)	
:MTESt:LOAD (see page 1283)	:RECall:MASK[:STARt] (see page 687)	
:MTESt:RUMode (see page 1284)	:MTESt:RMODe (see page 562)	
:MTESt:RUMode:SOFailure (see page 1285)	:MTESt:RMODe:FACTion:STOP (see page 565)	
:MTESt:{STARt | STOP} (see page 1286)	:RUN (see page 233) or :STOP (see page 237)	
:MTESt:TRIGger:SOURce (see page 1287)	:TRIGger Commands (see page 1053)	There are various commands for setting the source with different types of triggers.
:SAVE:IMAGe:AREA (see page 1288)	none	
:TIMebase:DELay (see page 1291)	:TIMebase:POSition (see page 1042) or :TIMebase:WINDow:POSition (see page 1049)	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of zoomed (delayed) time base window.
:SBUS<n>:LIN:SIGNal:DEFinition (see page 1289)	none	

Obsolete Command	Current Command Equivalent	Behavior Differences
:SYSTem:MENU (see page 1290)	:DISPlay:MENU (see page 328)	No change in behavior.
:TRIGger:TV:TVMode (see page 1292)	:TRIGger:TV:MODE (see page 1142)	

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the M9241/42/43A PXIe oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERSistence INFinite (see page 330)	
CHANnel:MATH	:FUNCtion:OPERation (see page 390)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see page 290)	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMal.
DISPlay:INVerse	none	
DISPlay:COLumn	none	
DISPlay:FREeze	none	
DISPlay:GRID	none	
DISPLay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCtion:MOVE	none	
FUNCtion:PEAKs	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available

Discontinued Command	Current Command Equivalent	Comments
:POWer:SIGNals:CYCLes	:POWer:SIGNals:CYCLes:HARMonics (see page 649) :POWer:SIGNals:CYCLes:QUALity (see page 650)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:DURation	:POWer:SIGNals:DURation:EFFiciency (see page 651) :POWer:SIGNals:DURation:MODulation (see page 652) :POWer:SIGNals:DURation:ONOFf:OFF (see page 653) :POWer:SIGNals:DURation:ONOFf:ON (see page 654) :POWer:SIGNals:DURation:RIPPle (see page 655) :POWer:SIGNals:DURation:TRANsient (see page 656)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:VMAXimum	:POWer:SIGNals:VMAXimum:INRush (see page 659) :POWer:SIGNals:VMAXimum:ONOFf:OFF (see page 660) :POWer:SIGNals:VMAXimum:ONOFf:ON (see page 661)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:VSTeady	:POWer:SIGNals:VSTeady:ONOFf:OFF (see page 662) :POWer:SIGNals:VSTeady:ONOFf:ON (see page 663) :POWer:SIGNals:VSTeady:TRANsient (see page 664)	This command was separated into several other commands for specific types of power analysis.
:POWer:SLEW:VALue	none	Slew rate values are now displayed using max and min measurements of a differentiate math function signal.
:PWRenable	none	The Power Event Enable Register does not exist in the M9241/42/43A PXIe oscilloscopes.
:PWRRegister[:EVENt]	none	The Power Event Event Register does not exist in the M9241/42/43A PXIe oscilloscopes.
SYSTem:KEY	none	

Discontinued Command	Current Command Equivalent	Comments
TEST:ALL	*TST (Self Test) (see page 198)	
TRACE subsystem	none	All commands discontinued, feature not available
TRIGger:ADVanced subsystem		Use new GLITch, PATTern, or TV trigger modes
TRIGger:TV:FIELd	:TRIGger:TV:MODE (see page 1142)	
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
:TRIGger:USB:SOURce:DMINus	none	USB serial decode and triggering is not supported on the M9241/42/43A PXIe oscilloscopes.
:TRIGger:USB:SOURce:DPLus	none	
:TRIGger:USB:SPEed	none	
:TRIGger:USB:TRIGger	none	
VAUToscale	none	

Discontinued Parameters

Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision M9241/42/43A PXIe oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:LABel

(see page 1354)

Command Syntax

```
:CHANnel:LABel <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2}

<string> ::= quoted ASCII string
```

The :CHANnel:LABel command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

NOTE The :CHANnel:LABel command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABel command (see page 280).

Query Syntax

```
:CHANnel:LABel?
```

The :CHANnel:LABel? query returns the label associated with a particular analog channel.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

:CHANnel2:SKEW

(see page 1354)

Command Syntax

```
:CHANnel2:SKEW <skew value>

<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns
```

The :CHANnel2:SKEW command sets the skew between channels 1 and 2. The maximum skew is +/-100 ns. You can use the oscilloscope's analog probe skew control to remove cable delay errors between channel 1 and channel 2.

NOTE The :CHANnel2:SKEW command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:PROBe:SKEW command (see page 287) instead.

NOTE This command is only valid for the two channel oscilloscope models.

Query Syntax

```
:CHANnel2:SKEW?
```

The :CHANnel2:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

```
<skew value><NL>

<skew value> ::= skew value in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 273

:CHANnel<n>:INPut

(see page 1354)

Command Syntax

```
:CHANnel<n>:INPut <impedance>

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:INPut command selects the input impedance setting for the specified channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see page 278) instead.

Query Syntax

```
:CHANnel<n>:INPut?
```

The :CHANnel<n>:INPut? query returns the current input impedance setting for the specified channel.

Return Format

```
<impedance value><NL>

<impedance value> ::= {ONEM | FIFT}
```

:CHANnel<n>:PMODe

 (see page 1354)

Command Syntax

```
:CHANnel<n>:PMODe <pmode value>

<pmode value> ::= {AUTo | MANual}

<n> ::= 1 to (# analog channels) in NR1 format
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODe sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE **The :CHANnel<n>:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.**

Query Syntax

```
:CHANnel<n>:PMODe?
```

The :CHANnel<n>:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>

<pmode value> ::= {AUT | MAN}
```

:DISPlay:CONNect

 (see page 1354)

Command Syntax

```
:DISPlay:CONNect <connect>

<connect> ::= {{ 1 | ON} | {0 | OFF}}
```

The :DISPlay:CONNect command turns vectors on and off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

NOTE **The :DISPlay:CONNect command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:VECTors command (see page 332) instead.**

Query Syntax

```
:DISPlay:CONNect?
```

The :DISPlay:CONNect? query returns the current state of the vectors setting.

Return Format

```
<connect><NL>

<connect> ::= {1 | 0}
```

See Also

- ":DISPlay:VECTors" on page 332

:ERASe

(see page 1354)

Command Syntax `:ERASe`

The :ERASe command erases the screen.

NOTE **The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISplay:CLEar command (see page 320) instead.**

:EXTernal:PMODe

(see page 1354)

Command Syntax

```
:EXTernal:PMODe <pmode value>

<pmode value> ::= {AUTo | MANual}
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE The :EXTernal:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

```
:EXTernal:PMODe?
```

The :EXTernal:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>

<pmode value> ::= {AUT | MAN}
```

:FUNCtion:GOFT:OPERation

(see page 1354)

Command Syntax

```
:FUNCtion:GOFT:OPERation <operation>

<operation> ::= {ADD | SUBTract | MULTiply}
```

The :FUNCtion:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to transform or filter functions (if available):

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTiply – Source1 * source2.

The :FUNCtion:GOFT:SOURce1 and :FUNCtion:GOFT:SOURce2 commands are used to select source1 and source2.

:FUNCtion:GOFT:SOURce1

(see page 1354)

Command Syntax

```
:FUNCtion:GOFT:SOURce1 <value>

<value> ::= CHANnel<n>

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:GOFT:SOURce1 command selects the first input channel for the g(t) source that can be used as the input to transform or filter functions (if available).

Query Syntax

```
:FUNCtion:GOFT:SOURce1?
```

The :FUNCtion:GOFT:SOURce1? query returns the current selection for the first input channel for the g(t) source.

Return Format

```
<value><NL>

<value> ::= CHAN<n>

<n> ::= {1 | 2 | 3 | 4} for the 4ch models

<n> ::= {1 | 2} for the 2ch models
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion:GOFT:SOURce2" on page 1264
- ":FUNCtion:GOFT:OPERation" on page 1262

:FUNCtion:GOFT:SOURce2

(see page 1354)

Command Syntax

```
:FUNCtion:GOFT:SOURce2 <value>

<value> ::= CHANnel<n>

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to transform or filter functions (if available).

Query Syntax

```
:FUNCtion:GOFT:SOURce2?
```

The :FUNCtion:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

Return Format

```
<value><NL>

<value> ::= CHAN<n>

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion:GOFT:SOURce1" on page 1263
- ":FUNCtion:GOFT:OPERation" on page 1262

:FUNCtion:SOURce

(see page 1354)

Command Syntax

```
:FUNCtion:SOURce <value>

<value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :FUNCtion:SOURce command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFF (differentiate), INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

NOTE The :FUNCtion:SOURce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :FUNCtion:SOURce1 command (see page 398) instead.

Query Syntax

```
:FUNCtion:SOURce?
```

The :FUNCtion:SOURce? query returns the current source for function operations.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | ADD | SUBT | MULT}

<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "Introduction to :FUNCtion<m> Commands" on page 365
- ":FUNCtion<m>:OPERation" on page 390

:FUNCtion:VIEW

 (see page 1354)

Command Syntax

```
:FUNCtion:VIEW <view>

<view> ::= {{1 | ON} | (0 | OFF}}
```

The :FUNCtion:VIEW command turns the selected function on or off. When ON is selected, the function performs as specified using the other FUNCtion commands. When OFF is selected, function is neither calculated nor displayed.

NOTE

The :FUNCtion:VIEW command is provided for backward compatibility to previous oscilloscopes. Use the :FUNCtion:DISPlay command (see page 369) instead.

Query Syntax

```
:FUNCtion:VIEW?
```

The :FUNCtion:VIEW? query returns the current state of the selected function.

Return Format

```
<view><NL>

<view> ::= {1 | 0}
```

:MEASure:LOWer

(see page 1354)

Command Syntax `:MEASure:LOWer <voltage>`

The :MEASure:LOWer command sets the lower measurement threshold value. This value and the UPPer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE **The :MEASure:LOWer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 457) instead.**

Query Syntax `:MEASure:LOWer?`

The :MEASure:LOWer? query returns the current lower threshold level.

Return Format `<voltage><NL>`

`<voltage> ::= the user-defined lower threshold in volts in NR3 format`

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:THResholds" on page 1270
- ":MEASure:UPPer" on page 1276

:MEASure:SCRatch

(see page 1354)

Command Syntax `:MEASure:SCRatch`

The :MEASure:SCRatch command clears all selected measurements and markers from the screen.

NOTE **The :MEASure:SCRatch command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:CLEar command (see page 455) instead.**

:MEASure:TDELta

(see page 1354)

Query Syntax `:MEASure:TDELta?`

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

Tdelta = Tstop - Tstart

Tstart is the time at the start marker (X1 cursor) and Tstop is the time at the stop marker (X2 cursor). No measurement is made when the :MEASure:TDELta? query is received by the oscilloscope. The delta time value that is output is the current value. This is the same value as the front-panel cursors delta X value.

NOTE **The :MEASure:TDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:XDELta command (see page 422) instead.**

Return Format `<value><NL>`

`<value> ::= time difference between start and stop markers in NR3 format`

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:XDELta" on page 422
- ":MEASure:TSTArt" on page 1273
- ":MEASure:TSTOp" on page 1274

:MEASure:THResholds

(see page 1354)

Command Syntax `:MEASure:THResholds {T1090 | T2080 | VOLTage}`

The :MEASure:THResholds command selects the thresholds used when making time measurements.

NOTE **The :MEASure:THResholds command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 457) instead.**

Query Syntax `:MEASure:THResholds?`

The :MEASure:THResholds? query returns the current thresholds selected when making time measurements.

Return Format `{T1090 | T2080 | VOLTage}<NL>`

`{T1090} uses the 10% and 90% levels of the selected waveform.`

`{T2080} uses the 20% and 80% levels of the selected waveform.`

`{VOLTage} uses the upper and lower voltage thresholds set by the UPPer and LOWer commands on the selected waveform.`

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:LOWer" on page 1267
- ":MEASure:UPPer" on page 1276

:MEASure:TMAX

(see page 1354)

Command Syntax

```
:MEASure:TMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :MEASure:TMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE The :MEASure:TMAX command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMAX command (see page 517) instead.

Query Syntax

```
:MEASure:TMAX? [<source>]
```

The :MEASure:TMAX? query returns the horizontal axis value at which the maximum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= time at maximum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:TMIN" on page 1272
- ":MEASure:XMAX" on page 517
- ":MEASure:XMIN" on page 518

:MEASure:TMIN

(see page 1354)

Command Syntax

```
:MEASure:TMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= 1 to (# analog channels) in NR1 format
```

The :MEASure:TMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE **The :MEASure:TMIN command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMIN command (see page 518) instead.**

Query Syntax

```
:MEASure:TMIN? [<source>]
```

The :MEASure:TMIN? query returns the horizontal axis value at which the minimum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= time at minimum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:TMAX" on page 1271
- ":MEASure:XMAX" on page 517
- ":MEASure:XMIN" on page 518

:MEASure:TSTArt

(see page 1354)

Command Syntax

```
:MEASure:TSTArt <value> [suffix]

<value> ::= time at the start marker in seconds

[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTArt command moves the start marker (X1 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTA, does not follow the defined Long Form to Short Form Truncation Rules (see page 1356). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTArt command produces an error.

NOTE The :MEASure:TSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X1Position command (see page 417) instead.

Query Syntax

```
:MEASure:TSTArt?
```

The :MEASure:TSTArt? query returns the time at the start marker (X1 cursor).

Return Format

```
<value><NL>

<value> ::= time at the start marker in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:XDELta" on page 422
- ":MEASure:TDELta" on page 1269
- ":MEASure:TSTOp" on page 1274

:MEASure:TSTOp

(see page 1354)

Command Syntax

```
:MEASure:TSTOp <value> [suffix]

<value> ::= time at the stop marker in seconds

[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTOp command moves the stop marker (X2 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTO, does not follow the defined Long Form to Short Form Truncation Rules (see page 1356). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTOp command produces an error.

NOTE The :MEASure:TSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X2Position command (see page 420) instead.

Query Syntax

```
:MEASure:TSTOp?
```

The :MEASure:TSTOp? query returns the time at the stop marker (X2 cursor).

Return Format

```
<value><NL>

<value> ::= time at the stop marker in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:X1Position" on page 417
- ":MARKer:X2Position" on page 420
- ":MARKer:XDELta" on page 422
- ":MEASure:TDELta" on page 1269
- ":MEASure:TSTArt" on page 1273

:MEASure:TVOLt

(see page 1354)

Query Syntax

```
:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform.  A rising slope is indicated by
            a plus sign (+).  A falling edge is indicated by a minus
            sign (-).

<occurrence> ::= the transition to be reported.  If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= 1 to (# analog channels) in NR1 format
```

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command (see page 504).

Return Format

```
<value><NL>

<value> ::= time in seconds of the specified voltage crossing
            in NR3 format
```

:MEASure:UPPer

(see page 1354)

Command Syntax `:MEASure:UPPer <value>`

The :MEASure:UPPer command sets the upper measurement threshold value. This value and the LOWer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE **The :MEASure:UPPer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 457) instead.**

Query Syntax `:MEASure:UPPer?`

The :MEASure:UPPer? query returns the current upper threshold level.

Return Format `<value><NL>`

`<value> ::= the user-defined upper threshold in NR3 format`

See Also

- "Introduction to :MEASure Commands" on page 448
- ":MEASure:LOWer" on page 1267
- ":MEASure:THResholds" on page 1270

:MEASure:VDELta

(see page 1354)

Query Syntax `:MEASure:VDELta?`

The :MEASure:VDELta? query returns the voltage difference between vertical marker 1 (Y1 cursor) and vertical marker 2 (Y2 cursor). No measurement is made when the :MEASure:VDELta? query is received by the oscilloscope. The delta value that is returned is the current value. This is the same value as the front-panel cursors delta Y value.

VDELta = value at marker 2 - value at marker 1

NOTE **The :MEASure:VDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:YDELta command (see page 429) instead.**

Return Format `<value><NL>`

`<value> ::= delta V value in NR1 format`

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YDELta" on page 429
- ":MEASure:TDELta" on page 1269
- ":MEASure:TSTArt" on page 1273

:MEASure:VSTArt

(see page 1354)

Command Syntax

```
:MEASure:VSTArt <vstart_argument>

<vstart_argument> ::= value for vertical marker 1
```

The :MEASure:VSTArt command moves the vertical marker (Y1 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X1Y1source command.

NOTE The short form of this command, VSTA, does not follow the defined Long Form to Short Form Truncation Rules (see page 1356). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTArt command produces an error.

NOTE The :MEASure:VSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y1Position command (see page 426) instead.

Query Syntax

```
:MEASure:VSTArt?
```

The :MEASure:VSTArt? query returns the current value of the Y1 cursor.

Return Format

```
<value><NL>

<value> ::= voltage at voltage marker 1 in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YDELta" on page 429
- ":MARKer:X1Y1source" on page 418
- ":MEASure:SOURce" on page 494
- ":MEASure:TDELta" on page 1269
- ":MEASure:TSTArt" on page 1273

:MEASure:VSTOp

(see page 1354)

Command Syntax

```
:MEASure:VSTOp <vstop_argument>

<vstop_argument> ::= value for Y2 cursor
```

The :MEASure:VSTOp command moves the vertical marker 2 (Y2 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X2Y2source command.

NOTE The short form of this command, VSTO, does not follow the defined Long Form to Short Form Truncation Rules (see page 1356). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTOp command produces an error.

NOTE The :MEASure:VSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y2Position command (see page 428) instead.

Query Syntax

```
:MEASure:VSTOp?
```

The :MEASure:VSTOp? query returns the current value of the Y2 cursor.

Return Format

```
<value><NL>

<value> ::= value of the Y2 cursor in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 413
- "Introduction to :MEASure Commands" on page 448
- ":MARKer:Y1Position" on page 426
- ":MARKer:Y2Position" on page 428
- ":MARKer:YDELta" on page 429
- ":MARKer:X2Y2source" on page 421
- ":MEASure:SOURce" on page 494
- ":MEASure:TDELta" on page 1269
- ":MEASure:TSTArt" on page 1273

:MTESt:AMASk:{SAVE | STORe}

(see page 1354)

Command Syntax

```
:MTESt:AMASk:{SAVE | STORe} "<filename>"
```

The :MTESt:AMASk:SAVE command saves the automask generated mask to a file. If an automask has not been generated, an error occurs.

The <filename> parameter is 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.

NOTE

The :MTESt:AMASk:{SAVE | STORe} command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :SAVE:MASK[:STARt] command (see page 703) instead.

See Also

- "Introduction to :MTESt Commands" on page 545

:MTESt:AVERage

 (see page 1354)

Command Syntax

```
:MTESt:AVERage <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

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.

NOTE **The :MTESt:AVERage command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :ACQuire:TYPE AVERage command (see page 257) instead.**

Query Syntax

```
:MTESt:AVERage?
```

The :MTESt:AVERage? query returns the current setting for averaging.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AVERage:COUNt" on page 1282

:MTESt:AVERage:COUNt

(see page 1354)

Command Syntax

```
:MTESt:AVERage:COUNt <count>

<count> ::= an integer from 2 to 65536 in NR1 format
```

The :MTESt:AVERage:COUNt command sets the number of averages for the waveforms. With the AVERage acquisition type, 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.

NOTE **The :MTESt:AVERage:COUNt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :ACQuire:COUNt command (see page 246) instead.**

Query Syntax

```
:MTESt:AVERage:COUNt?
```

The :MTESt:AVERage:COUNt? query returns the currently selected count value.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 65536 in NR1 format
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AVERage" on page 1281

:MTESt:LOAD

(see page 1354)

Command Syntax `:MTESt:LOAD "<filename>"`

The :MTESt:LOAD command loads the specified mask file.

The <filename> parameter is an MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

NOTE **The :MTESt:LOAD command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :RECall:MASK[:STARt] command (see page 687) instead.**

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:AMASk:{SAVE | STORe}" on page 1280

:MTESt:RUMode

(see page 1354)

Command Syntax

```
:MTESt:RUMode {FORever | TIME,<seconds> | {WAVeforms,<wfm_count>}}

<seconds> ::= from 1 to 86400 in NR3 format

<wfm_count> ::= number of waveforms in NR1 format
                from 1 to 1,000,000,000
```

The :MTESt:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

- FORever — runs the Mask Test until the test is turned off.
- TIME — sets the amount of time in seconds that a mask test will run before it terminates. The <seconds> parameter is a real number from 1 to 86400 seconds.
- WAVeforms — sets the maximum number of waveforms that are required before the mask test terminates. The <wfm_count> parameter indicates the number of waveforms that are to be acquired; it is an integer from 1 to 1,000,000,000.

NOTE The :MTESt:RUMode command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MTESt:RMODe command (see page 562) instead.

Query Syntax

```
:MTESt:RUMode?
```

The :MTESt:RUMode? query returns the currently selected termination condition and value.

Return Format

```
{FOR | TIME,<seconds> | {WAV,<wfm_count>}}<NL>

<seconds> ::= from 1 to 86400 in NR3 format

<wfm_count> ::= number of waveforms in NR1 format
                from 1 to 1,000,000,000
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RUMode:SOFailure" on page 1285

:MTESt:RUMode:SOFailure

(see page 1354)

Command Syntax

```
:MTESt:RUMode:SOFailure <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

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.

NOTE The :MTESt:RUMode:SOFailure command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MTESt:RMODe:FACTion:STOP command (see page 565) instead.

Query Syntax

```
:MTESt:RUMode:SOFailure?
```

The :MTESt:RUMode:SOFailure? query returns the current state of the Stop on Failure control.

Return Format

```
<on_off><NL>

<on_off> ::= {1 | 0}
```

See Also

- "Introduction to :MTESt Commands" on page 545
- ":MTESt:RUMode" on page 1284

:MTESt:{STARt | STOP}

(see page 1354)

Command Syntax `:MTESt:{STARt | STOP}`

The :MTESt:{STARt | STOP} command starts or stops the acquisition system.

NOTE The :MTESt:STARt and :MTESt:STOP commands are obsolete and are provided for backward compatibility to previous oscilloscopes. Use the :RUN command (see page 233) and :STOP command (see page 237) instead.

See Also
- "Introduction to :MTESt Commands" on page 545

:MTESt:TRIGger:SOURce

(see page 1354)

Command Syntax

```
:MTESt:TRIGger:SOURce <source>
<source> ::= CHANnel<n>
<n> ::= 1 to (# analog channels) in NR1 format
```

The :MTESt:TRIGger:SOURce command sets the channel to use as the trigger.

NOTE The :MTESt:TRIGger:SOURce command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the trigger source commands (see page 1053) instead.

Query Syntax

```
:MTESt:TRIGger:SOURce?
```

The :MTESt:TRIGger:SOURce? query returns the currently selected trigger source.

Return Format

```
<source> ::= CHAN<n>
<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "Introduction to :MTESt Commands" on page 545

:SAVE:IMAGe:AREA

 (see page 1354)

Query Syntax `:SAVE:IMAGe:AREA?`

The :SAVE:IMAGe:AREA? query returns the selected image area.

When saving images, this query returns SCR (screen). When saving setups or waveform data, this query returns GRAT (graticule) even though graticule images are not saved.

Return Format `<area><NL>`

`<area> ::= {GRAT | SCR}`

See Also

- "Introduction to :SAVE Commands" on page 694
- ":SAVE:IMAGe[:STARt]" on page 697
- ":SAVE:IMAGe:FACTors" on page 698
- ":SAVE:IMAGe:FORMat" on page 699
- ":SAVE:IMAGe:INKSaver" on page 700
- ":SAVE:IMAGe:PALette" on page 701

:SBUS<n>:LIN:SIGNal:DEFinition

(see page 1354)

Command Syntax

```
:SBUS<n>:LIN:SIGNal:DEFinition <value>

<value> ::= {LIN | RX | TX}
```

The :SBUS<n>:LIN:SIGNal:DEFinition command sets the LIN signal type. These signals can be set to:

Dominant low signals:

- LIN – the actual LIN single-end bus signal line.
- RX – the Receive signal from the LIN bus transceiver.
- TX – the Transmit signal to the LIN bus transceiver.

NOTE **This command is available, but the only legal value is LIN.**

Query Syntax

```
:SBUS<n>:LIN:SIGNal:DEFinition?
```

The :SBUS<n>:LIN:SIGNal:DEFinition? query returns the current LIN signal type.

Return Format

```
<value><NL>

<value> ::= LIN
```

See Also

- "Introduction to :TRIGger Commands" on page 1053
- ":TRIGger:MODE" on page 1066
- ":SBUS<n>:LIN:SIGNal:BAUDrate" on page 807
- ":SBUS<n>:LIN:SOURce" on page 808

:SYSTem:MENU

(see page 1354)

Command Syntax `:SYSTem:MENU <menu>`

`<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}`

The :SYSTem:MENU command changes the front panel softkey menu.

:TIMebase:DELay

(see page 1354)

Command Syntax

```
:TIMebase:DELay <delay_value>

<delay_value> ::= time in seconds from trigger to the delay reference
                  point on the screen.

The valid range for delay settings depends on the time/division
setting for the main time base.
```

The :TIMebase:DELay command sets the main time base delay. This delay is the time between the trigger event and the delay reference point on the screen. The delay reference point is set with the :TIMebase:REFerence command (see page 1045).

NOTE The :TIMebase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase:POSition command (see page 1042) instead.

Query Syntax

```
:TIMebase:DELay?
```

The :TIMebase:DELay query returns the current delay value.

Return Format

```
<delay_value><NL>

<delay_value> ::= time from trigger to display reference in seconds
                  in NR3 format.
```

Example Code

```
' TIMEBASE_DELAY - Sets the time base delay.  This delay
' is the internal time between the trigger event and the
' onscreen delay reference point.

' Set time base delay to 0.0.
myScope.WriteString ":TIMEBASE:DELAY 0.0"
```

See complete example programs at: Chapter 40, “Programming Examples,” starting on page 1363

:TRIGger:TV:TVMode

(see page 1354)

Command Syntax

```
:TRIGger:TV:TVMode <mode>

<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical
           | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERtical parameter is only available when :TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric (see page 1145).

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIeld	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERtical	LINEVert

NOTE The :TRIGger:TV:TVMode command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see page 1142) instead.

Query Syntax

```
:TRIGger:TV:TVMode?
```

The :TRIGger:TV:TVMode? query returns the TV trigger mode.

Return Format

```
<value><NL>

<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
            | LALT | LVER}
```

36 Error Messages

-440, Query UNTERMINATED after indefinite response

-430, Query DEADLOCKED

-420, Query UNTERMINATED

-410, Query INTERRUPTED

-400, Query error

-340, Calibration failed

-330, Self-test failed

-321, Out of memory

-320, Storage fault

-315, Configuration memory lost

-314, Save/recall memory lost

-313, Calibration memory lost

-311, Memory error

-310, System error

-300, Device specific error

-278, Macro header not found

-277, Macro redefinition not allowed

-276, Macro recursion error

-273, Illegal macro label

-272, Macro execution error

-258, Media protected

-257, File name error

-256, File name not found

-255, Directory full

-254, Media full

-253, Corrupt media

-252, Missing media

-251, Missing mass storage

-250, Mass storage error

-241, Hardware missing

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

-240, Hardware error

-231, Data questionable

-230, Data corrupt or stale

-224, Illegal parameter value

-223, Too much data

-222, Data out of range

-221, Settings conflict

-220, Parameter error

-200, Execution error

-183, Invalid inside macro definition

-181, Invalid outside macro definition

-178, Expression data not allowed

-171, Invalid expression

-170, Expression error

-168, Block data not allowed

-161, Invalid block data

-158, String data not allowed

-151, Invalid string data

-150, String data error

-148, Character data not allowed

-138, Suffix not allowed

-134, Suffix too long

-131, Invalid suffix

-128, Numeric data not allowed

-124, Too many digits

-123, Exponent too large

-121, Invalid character in number

-120, Numeric data error

-114, Header suffix out of range

-113, Undefined header

-112, Program mnemonic too long

-109, Missing parameter

-108, Parameter not allowed

-105, GET not allowed

-104, Data type error

-103, Invalid separator

-102, Syntax error

-101, Invalid character

-100, Command error

+10, Software Fault Occurred

+100, File Exists

+101, End-Of-File Found

+102, Read Error

+103, Write Error

+104, Illegal Operation

+105, Print Canceled

+106, Print Initialization Failed

+107, Invalid Trace File

+108, Compression Error

+109, No Data For Operation

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPlay:DATA? query, but there may be no image stored.

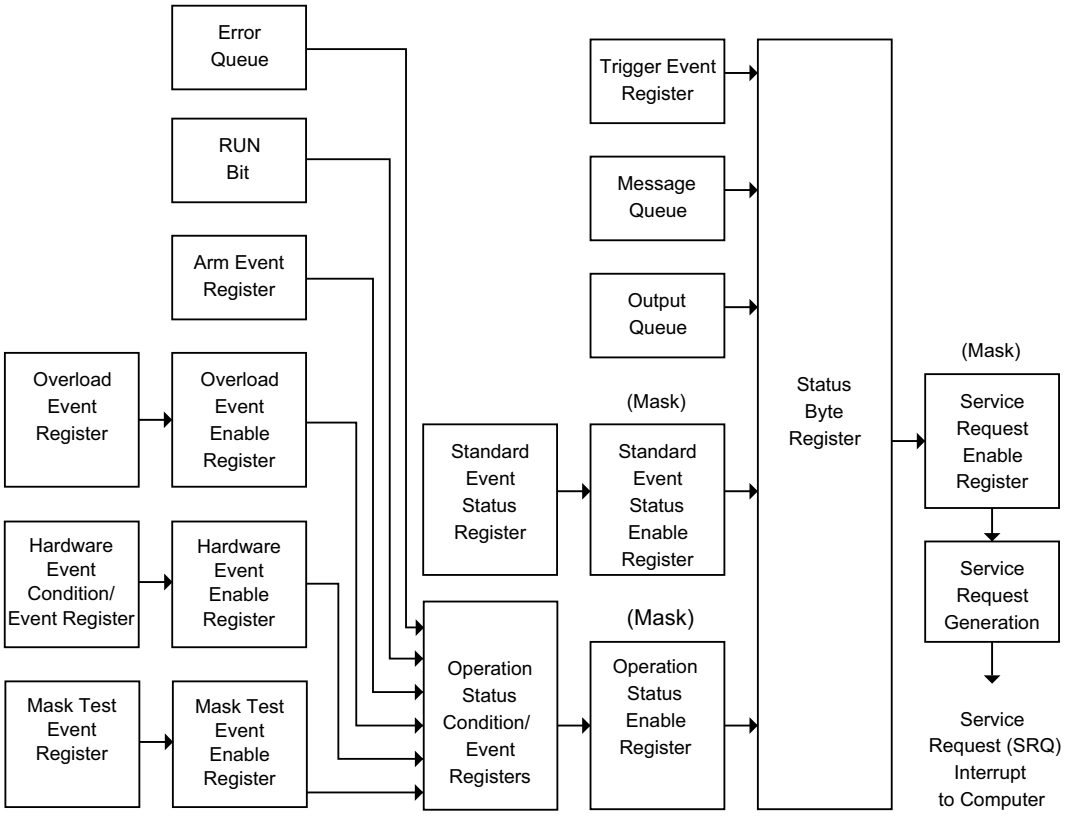
+112, Unknown File Type

+113, Directory Not Supported

37 Status Reporting

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting (for example, the Status Byte Register and the Standard Event Status Register). There are also instrument-defined structures and bits (for example, the Operation Status Event Register and the Overload Event Register).

An overview of the oscilloscope's status reporting structure is shown in the following block diagram. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.



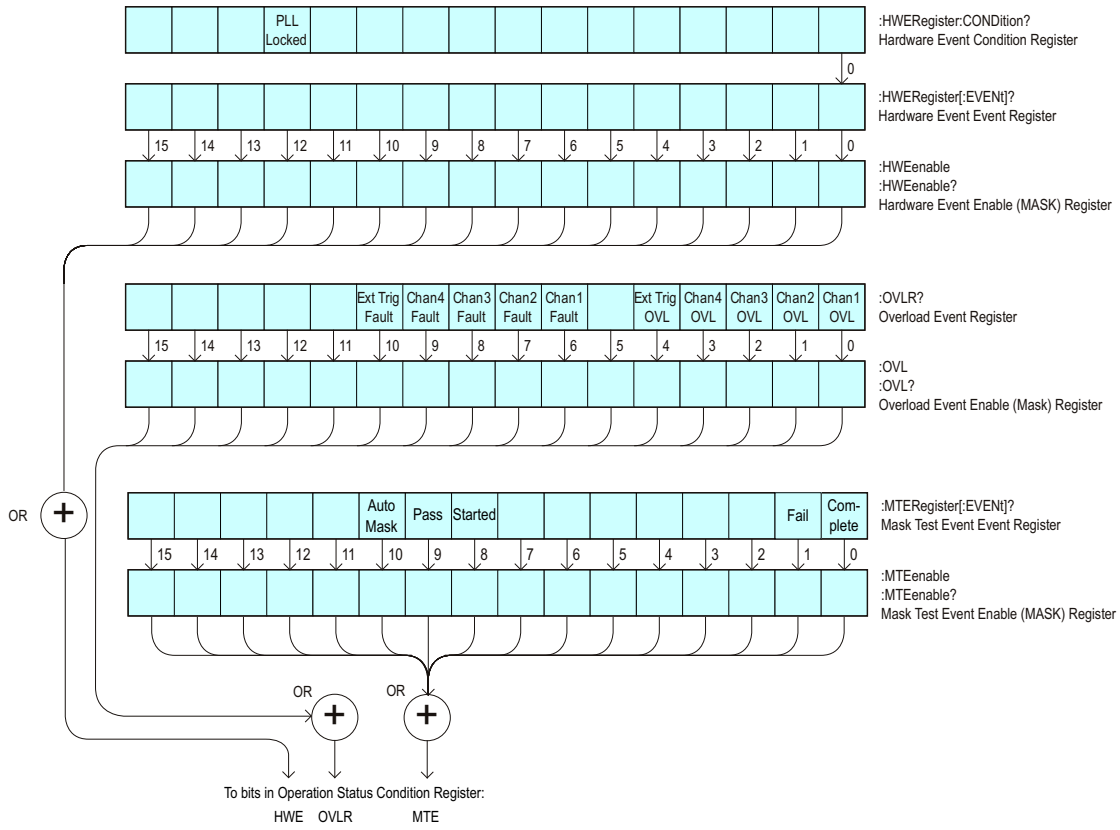
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

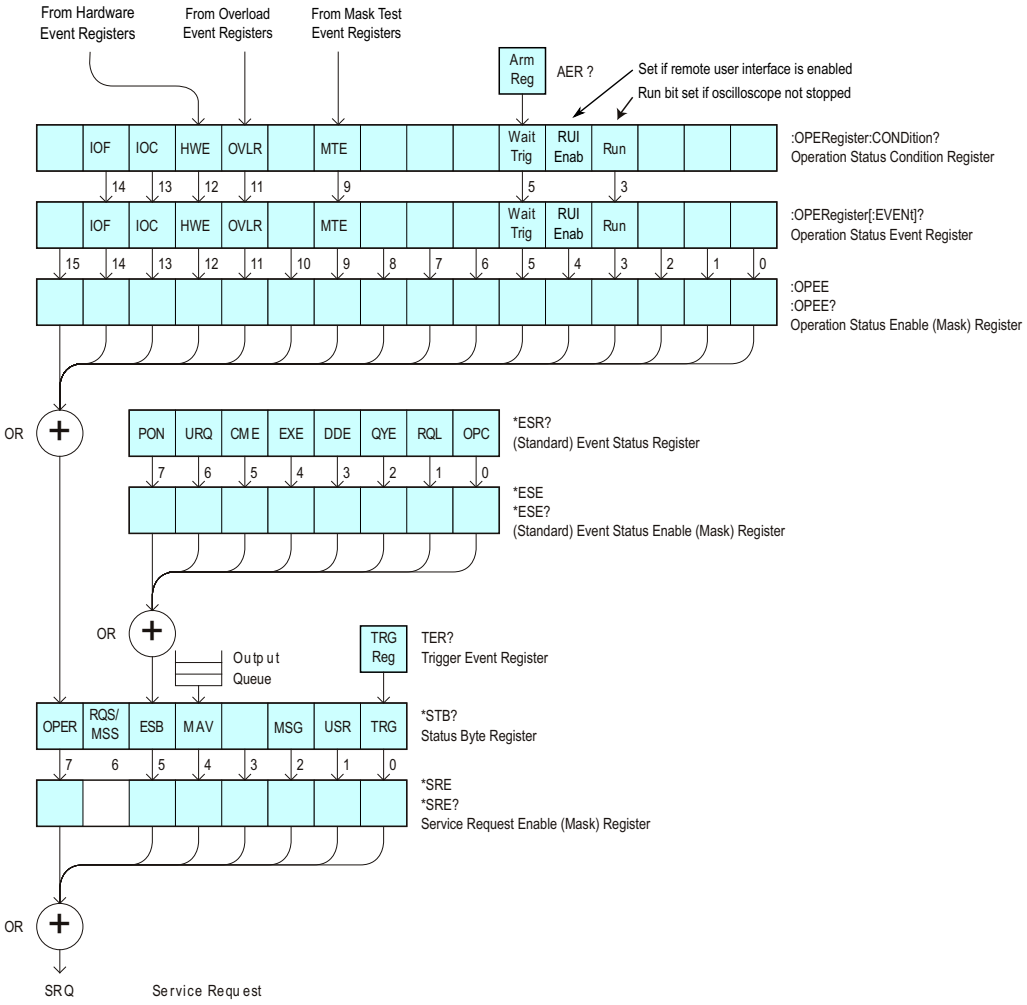
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled with the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The *CLS command clears all event registers and all queues except the output queue. If you send *CLS immediately after a program message terminator, the output queue is also cleared.

Status Reporting Data Structures

The following figure shows how the status register bits are masked and logically OR'ed to generate service requests (SRQ) on particular events.





The status register bits are described in more detail in the following tables:

- Table 71
- Table 69
- Table 79
- Table 80
- Table 82
- Table 74
- Table 75
- Table 77

The status registers picture above shows how the different status reporting data structures work together. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to an external controller, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Status Byte Register (STB)

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

The Status Byte Register can be read using either the *STB? Common Command or the programming interface serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? command reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

No other bits in the Status Byte Register are cleared by either the *STB? query or the serial poll, except the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? command.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

The following example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register.

```
myScope.WriteString "*STB?"
varQueryResult = myScope.ReadNumber
MsgBox "Status Byte Register, Read: 0x" + Hex(varQueryResult)
```

The next program prints 0xD1 and clears bit 6 (RQS) and bit 4 (MAV) of the Status Byte Register. The difference in the output value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set and is cleared when the Status Byte Register is read by the serial poll command.

Example The following example uses the resource session object's ReadSTB method to read the contents of the oscilloscope's Status Byte Register.

```
varQueryResult = myScope.IO.ReadSTB
MsgBox "Status Byte Register, Serial Poll: 0x" + Hex(varQueryResult)
```

NOTE **Use Serial Polling to Read Status Byte Register**. Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

See Also

- "*STB (Read Status Byte)" on page 195

Service Request Enable Register (SRE)

Setting the Service Request Enable Register bits enable corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command and the bits that are set are read with the *SRE? query.

Example The following example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
myScope.WriteString "*SRE " + CStr(CInt("&H30"))
```

This example uses the decimal parameter value of 48, the string returned by CStr(CInt("&H30")), to enable the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

See Also

- "*SRE (Service Request Enable)" on page 193

Trigger Event Register (TER)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TER event register stays set until it is cleared by reading the register or using the *CLS command. If your application needs to detect multiple triggers, the TER event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation, you must clear the event register each time the trigger bit is set.

See Also

- ":TER (Trigger Event Register)" on page 238

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

When using the Keysight VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The length of the oscilloscope's message queue is 1. Note that messages sent with the :SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

(Standard) Event Status Register (ESR)

The (Standard) Event Status Register (ESR) monitors the following oscilloscope status events:

- PON - Power On
- URQ - User Request
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occur, the event sets the corresponding bit in the register. If the bits are enabled in the Standard Event Status Enable Register, the bits set in this register generate a summary bit to set bit 5 (ESB) in the Status Byte Register.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all of the bits that are set at the present time.

Example The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
MsgBox "Standard Event Status Register: 0x" + Hex(varQueryResult)
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

See Also

- "*ESR (Standard Event Status Register)" on page 181

(Standard) Event Status Enable Register (ESE)

To allow any of the (Standard) Event Status Register (ESR) bits to generate a summary bit, you must first enable that bit. Enable the bit by using the *ESE (Event Status Enable) common command to set the corresponding bit in the (Standard) Event Status Enable Register (ESE).

Set bits are read with the *ESE? query.

Example Suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the (Standard) Event Status Register are bits 2 through 5 (hexadecimal value 0x3C). Therefore, you can enable any of these bits to generate the summary bit by sending:

```
myScope.WriteString "*ESE " + CStr(CInt("&H3C"))
```

Whenever an error occurs, it sets one of these bits in the (Standard) Event Status Register. Because all the error related bits are enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), an SRQ service request interrupt is sent to the controller PC.

NOTE **Disabled (Standard) Event Status Register bits respond but do not generate a summary bit.** (Standard) Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit to the Status Byte Register.

See Also

- "*ESE (Standard Event Status Enable)" on page 179

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error 350, Queue overflow. Any time the queue overflows, the least recent errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the Queue overflow message).

The error queue is read with the :SYSTem:ERRor? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return "0, No error".

The error queue is cleared when:

- the instrument is powered up,
- the instrument receives the *CLS common command, or
- the last item is read from the error queue.

Operation Status Event Register (:OPERegister[:EVENt])

The Operation Status Event Register register hosts these bits:

Name	Location	Description
RUN bit	bit 3	Is set whenever the instrument goes from a stop state to a single or running state.
RUI Enab bit	bit 4	The remote user interface has gone from a disabled state to an enabled state. The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, for example when: ▪ a modal dialog box is open ▪ the user is being prompted ▪ all segments are being analyzed ▪ there is a channel overload ▪ certain compliance applications are running When disabled, commands or queries are accepted, but "settings conflict" errors are returned. The status model is the only part of the remote user interface that is enabled. To determine when the remote interface is re-enabled, you can read this bit or wait for the event that gets generated when its status goes from disabled to enabled.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.
HWE bit	bit 12	Comes from the Hardware Event Registers.
IOC bit	bit 13	Is set when any IO operation completes. IO operations are any remote data request using any interface (USB, LAN, or GPIB). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.
IOF bit	bit 14	Is set only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.

If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Event Register is read and cleared with the :OPERegister[:EVENt]? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

See Also

- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 226

Operation Status Condition Register (:OPERegister:CONDition)

The Operation Status Condition Register register hosts these bits:

Name	Location	Description
RUN bit	bit 3	Is set whenever the instrument is not stopped.
RUI Enab bit	bit 4	Shows whether the remote user interface is enabled. The front-panel graphical user interface can disable most of the remote interface, including the *OPC syntax, for example when: ▪ a modal dialog box is open ▪ the user is being prompted ▪ all segments are being analyzed ▪ there is a channel overload ▪ certain compliance applications are running When disabled, commands or queries are accepted, but "settings conflict" errors are returned. The status model is the only part of the remote user interface that is enabled. To determine when the remote interface is re-enabled, you can read this bit or wait for the event that gets generated when its status goes from disabled to enabled.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.
HWE bit	bit 12	Comes from the Hardware Event Registers.
IOC bit	bit 13	Is set when any IO operation completes. IO operations are any remote data request using any interface (USB, LAN, or GPIB). For example, if you connect to an oscilloscope using the USB interface and then request waveform data, the IOC bit will be set when the IO operation completes.
IOF bit	bit 14	Is set only when something causes the IO operation to fail, like disconnecting a USB device or interrupting the operation from the oscilloscope's front panel.

The :OPERegister:CONDition? query returns the value of the Operation Status Condition Register.

See Also
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 223

Arm Event Register (AER)

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and the OPER bit (bit 7) in the Status Byte Register when the instrument becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query or using the *CLS command. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, then you must clear the event register after each time it has been set.

See Also

- ":AER (Arm Event Register)" on page 204

Overload Event Register (:OVLRegister)

The Overload Event Register register hosts these bits:

Name	Location	Description
Channel 1 OVL	bit 0	Overload has occurred on Channel 1 input.
Channel 2 OVL	bit 1	Overload has occurred on Channel 2 input.
Channel 3 OVL	bit 2	Overload has occurred on Channel 3 input.
Channel 4 OVL	bit 3	Overload has occurred on Channel 4 input.
External Trigger OVL	bit 4	Overload has occurred on External Trigger input.
Channel 1 Fault	bit 6	Fault has occurred on Channel 1 input.
Channel 2 Fault	bit 7	Fault has occurred on Channel 2 input.
Channel 3 Fault	bit 8	Fault has occurred on Channel 3 input.
Channel 4 Fault	bit 9	Fault has occurred on Channel 4 input.
External Trigger Fault	bit 10	Fault has occurred on External Trigger input.

See Also
- ":OVLRegister (Overload Event Register)" on page 231

Hardware Event Event Register (:HWERegister[:EVENt])

This register hosts the PLL LOCKED bit (bit 12).

- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

See Also

- ":HWERegister[:EVENt] (Hardware Event Event Register)" on page 216

Hardware Event Condition Register (:HWERegister:CONDition)

This register hosts the PLL LOCKED bit (bit 12).

- The :HWERegister:CONDition? query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

See Also

- ":HWERegister:CONDition (Hardware Event Condition Register)" on page 215

Mask Test Event Event Register (:MTERegister[:EVENt])

The Mask Test Event Event Register register hosts these bits:

Name	Location	Description
Complete	bit 0	Is set when the mask test is complete.
Fail	bit 1	Is set when there is a mask test failure.
Started	bit 8	Is set when mask testing is started.
Pass	bit 9	Is set when the mask test passes.
Auto Mask	bit 10	Is set when auto mask creation is completed.

The :MTERegister[:EVENt]? query returns the value of, and clears, the Mask Test Event Event Register.

See Also
- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 219

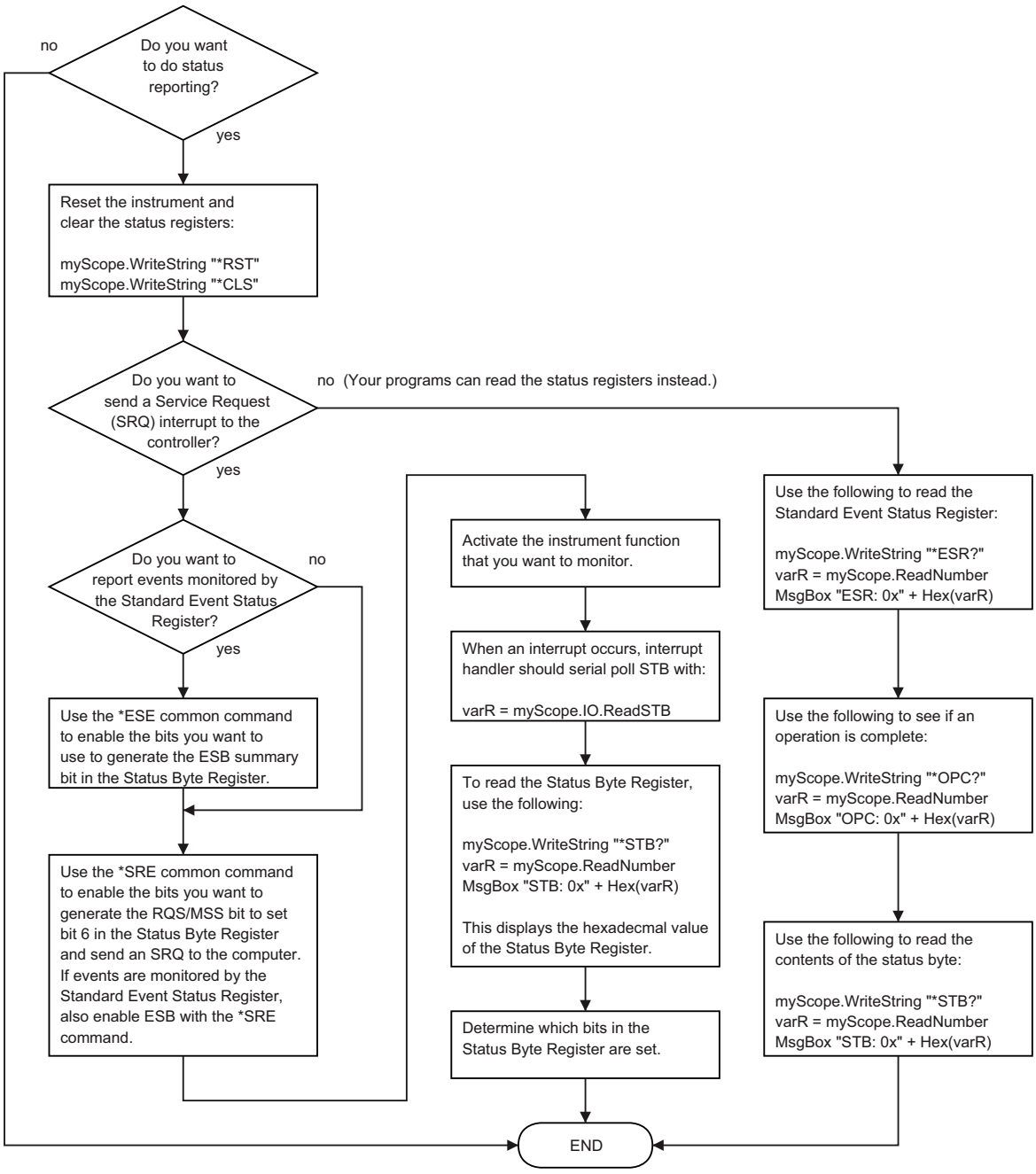
Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately after a program message terminator, the output queue is also cleared.

See Also

- "*CLS (Clear Status)" on page 178

Status Reporting Decision Chart



Example: Checking for Armed Status

```
# -*- coding: utf-8 -*-

# ********************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows three
# methods to tell whether a Keysight InfiniiVision oscilloscope is
# armed.
# ********************************************************************

# Import modules
# --------------------------------------------------------------------
import sys
import visa
import time

# Initialization constants
# --------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = \
    "TCPIP0::a-mx4154a-60014.cos.is.keysight.com::inst0::INSTR"
GLOBAL_TOUT =  20000   # IO timeout in milliseconds


# ====================================================================
# Method 1: Query the Armed Event Register with :AER?
# --------------------------------------------------------------------
# This method reads the 1-bit Armed Event Register using the :AER?
# query.
#
# The Armed Event Register bit goes low (0) when it is read using
# :AER? or when a *CLS command is issued.
# ====================================================================
def method_1():

    # Stop the oscilloscope.
    KsInfiniiVisionX.query(":STOP;*OPC?")

    # Method 1: Initiate capture using :SINGle
    # ----------------------------------------------------------------
    print "Acquiring signal (Method 1, using :SINGle)...\n"
    now = time.clock()

    # Clear all status registers before checking for new events.
    KsInfiniiVisionX.write("*CLS")

    # Because the :AER? query will not work with :DIGitize (which is
    # blocking), use the :SINGle command to start the acquisition.
    KsInfiniiVisionX.write(":SINGle")

    # Method 1: Determine if armed using :AER? query.
    # ----------------------------------------------------------------
    # Define armed criteria.
    ARMED = 1
```

```
    # Test for armed.
    ARMED_STATUS = int(KsInfiniiVisionX.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(KsInfiniiVisionX.query(":AER?"))

    print "Oscilloscope is armed (method 1, using :AER? query)!"
    print "It took " + str(time.clock() - now) +\
          " seconds to arm.\n"


# ======================================================================
# Method 2: Read the Status Byte
# ----------------------------------------------------------------------
# This method reads the Status Byte register's OPER bit (bit 7) using
# the "read status byte" function in VISA, which works during blocking
# commands and can therefore be used with the :DIGitize command.
#
# The Status Byte bits do NOT go low (0) when the register is read.
#
# The *CLS command will clear the Status Byte bits.
#
# CAUTION: The oscilloscope's status registers are not updated until
# the :DIGitize completes.  So, while the ARM may go true midway
# through the :DIGitize, it does not get reported to the status model
# until the :DIGitize completes, and therefore the STB does not
# reflect it as true until the :DIGitize completes.
# ======================================================================
def method_2():

    # Stop the oscilloscope.
    KsInfiniiVisionX.query(":STOP;*OPC?")

    # Method 2: Initiate capture using :DIGitize or :SINGle
    # ------------------------------------------------------------------
    print "Acquiring signal (Method 2, using :DIGitize)...\n"
    now = time.clock()

    # Clear all status registers before checking for new events.
    KsInfiniiVisionX.write("*CLS")

    # Mask out all bits in the Operation Status Event Register except
    # for the ARM bit.
    KsInfiniiVisionX.write(":OPEE 32")  # "Unmask" only the arm bit

    # Use the :DIGitize command to start the acquisition.
    KsInfiniiVisionX.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(KsInfiniiVisionX.read_stb())
    ARMED_STATUS = STATUS_BYTE & ARM_MASK
    # Note that you could also do:
    #  ARMED_STATUS = int(KsInfiniiVisionX.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(KsInfiniiVisionX.read_stb())
        ARMED_STATUS = STATUS_BYTE & ARM_MASK

    print "Oscilloscope is armed (method 2, using Read STB function)!"
    print "It took " + str(time.clock() - now) +\
          " seconds to arm.\n"


# ======================================================================
# Method 3: Query the Operation Status Event Register with :OPER?
# ----------------------------------------------------------------------
# This method reads the Operation Status Event Register's Wait Trig
# bit (bit 5) using the :OPER? query.
#
# The Operation Status Event Register bits are cleared (0) when the
# register is read.
#
# Also, the Wait Trig bit does NOT go low (0) when the oscilloscope
# becomes unarmed by starting or stopping another acquisition (before
# the first one finishes) or by changing the time scale.
#
# The Wait Trig bit is cleared by a *CLS command, by reading the
# Operation Status Event Register with the :OPER? query, or by reading
# the Armed Event Register regsiter with the :AER? query.
# ======================================================================
def method_3():

    # Stop the oscilloscope.
    KsInfiniiVisionX.query(":STOP;*OPC?")

    # Method 3: Initiate capture using :SINGle
    # ------------------------------------------------------------------
    print "Acquiring signal (Method 3, using :SINGle)...\n"
    now = time.clock()

    # Clear all status registers before checking for new events.
    KsInfiniiVisionX.write("*CLS")

    # Because the :OPER? query will not work with :DIGitize (which is
    # blocking), use the :SINGle command to start the acquisition.
    KsInfiniiVisionX.write(":SINGle")
```

```
    # Method 3: Determine if armed using :OPER? query.
    # ----------------------------------------------------------------
    # Define bit masks for the Operation Status Event Register
    ARM_BIT = 5
    # 1 leftshift 5 = 32 (bit 5 in the Operation Status Event
    # Register)
    ARM_MASK = 1 << ARM_BIT

    # Define armed criteria.
    ARMED = 1 << ARM_BIT   # 1 leftshift 5 = 32

    # Test for armed.
    STATUS_REGISTER = int(KsInfiniiVisionX.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(KsInfiniiVisionX.query(":OPER?"))
       ARMED_STATUS = STATUS_REGISTER & ARM_MASK

    print "Oscilloscope is armed (method 3, using :OPER? query)!"
    print "It took " + str(time.clock() - now) +\
          " seconds to arm.\n"


# ======================================================================
# Main
# ======================================================================

# Connect and initialize oscilloscope
# ----------------------------------------------------------------------
# Define VISA Resource Manager & Install directory
rm = visa.ResourceManager('C:\\Windows\\System32\\agvisa32.dll')

# Define and open the oscilloscope using the VISA address
KsInfiniiVisionX = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
KsInfiniiVisionX.timeout = GLOBAL_TOUT

# Clear the instrument bus
KsInfiniiVisionX.clear()

# Reset the oscilloscope.
KsInfiniiVisionX.write("*RST")

# Autoscale to set up vertical scale and trigger level on Probe Comp.
KsInfiniiVisionX.write(":CHANnel1:DISPlay OFF")
KsInfiniiVisionX.write(":CHANnel2:DISPlay ON")
KsInfiniiVisionX.write(":AUToscale:CHANnels DISPlayed")
KsInfiniiVisionX.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
KsInfiniiVisionX.write(":TIMebase:RANGe 10")
# Prevent Auto trigger.
KsInfiniiVisionX.write(":TRIGger:SWEep NORMal")

# Use the three methods to check whether the oscilloscope is armed.
# ------------------------------------------------------------------
method_1()
method_2()
method_3()

# End of Script
# ------------------------------------------------------------------
KsInfiniiVisionX.clear()   # Clear communications interface
KsInfiniiVisionX.close()   # Close communications interface
print "All done."
```

Example: Waiting for IO Operation Complete

NOTE The IOC (IO Operation Complete) and IOF (IO Operation Failed) bits in the Operation Status Event Register identify when :WMEMory<r>:SAVE and other :SAVE and :RECall commands are completely done. To determine when a IO operation is complete, you should use these bits instead of the OPC (Operation Complete) bit in the Standard Event Status Register or the *OPC? query.

```
# -*- coding: utf-8 -*-

# ********************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows how to
# wait for IO operation completion in a Keysight InfiniiVision
# oscilloscope.
# ********************************************************************

# Import modules
# --------------------------------------------------------------------
import sys
import visa
import time

# Initialization constants
# --------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = \
    "TCPIP0::a-mx4154a-60014.cos.is.keysight.com::inst0::INSTR"
GLOBAL_TOUT =  20000   # IO timeout in milliseconds


# ====================================================================
# Check for IO Operation Complete using :OPER? query.
# --------------------------------------------------------------------
# This method reads the Operation Status Register's IOC bit
# (bit 13) using the :OPER? query.
#
# The Operation Status Event Register bits are cleared (0) when the
# register is read.
#
# All status bits are cleared by a *CLS command.
# ====================================================================
def wait_io_operation():

    # ----------------------------------------------------------------
    now = time.clock()
    # Define bit masks for the Operation Status Event Register
    IOC_BIT = 13
    # 1 leftshift 13 = 8192 (bit 13 in the Operation Status Event
    # Register)
    IOC_MASK = 1 << IOC_BIT

    # Define IO complete criteria.
```

```
    IO_COMPLETE = 1 << IOC_BIT   # 1 leftshift 13 = 8192

    # Test for armed.
    STATUS_REGISTER = int(KsInfiniiVisionX.query(":OPER?"))
    IO_COMPLETE_STATUS = STATUS_REGISTER & IOC_MASK

    # Wait indefinitely until armed.
    while IO_COMPLETE_STATUS != IO_COMPLETE:
       # Check the status again after small delay.
       time.sleep(0.1)   # 100 ms delay to prevent excessive queries.
       STATUS_REGISTER = int(KsInfiniiVisionX.query(":OPER?"))
       IO_COMPLETE_STATUS = STATUS_REGISTER & IOC_MASK

    time_in_seconds = str(time.clock() - now)
    print "IO Operation Complete (from :OPER? query)."
    print "It took %s seconds for IO operation to complete.\n" % \
          time_in_seconds


# ======================================================================
# Main
# ======================================================================

# Connect and initialize oscilloscope
# ----------------------------------------------------------------------
# Define VISA Resource Manager & Install directory
rm = visa.ResourceManager('C:\\Windows\\System32\\agvisa32.dll')

# Define and open the oscilloscope using the VISA address
KsInfiniiVisionX = rm.open_resource(VISA_ADDRESS)

# Set the Global Timeout
KsInfiniiVisionX.timeout = GLOBAL_TOUT

# Clear the instrument bus
KsInfiniiVisionX.clear()

# Reset the oscilloscope.
# KsInfiniiVisionX.write("*RST")
# Or comment out to use the current oscilloscope setup.

# Autoscale to set up vertical scale and trigger level on channels 1 and
 2.
KsInfiniiVisionX.write(":CHANnel1:DISPlay ON")
KsInfiniiVisionX.write(":CHANnel2:DISPlay ON")
KsInfiniiVisionX.write(":AUToscale:CHANnels DISPlayed")
KsInfiniiVisionX.write(":AUToscale")

# Between :WMEMory<r>:SAVE commands, wait for IO complete.
# ----------------------------------------------------------------------
# Clear all status registers before checking for new events.
KsInfiniiVisionX.write("*CLS")

KsInfiniiVisionX.write(":WMEMory1:SAVE CHANnel1")
wait_io_operation()
KsInfiniiVisionX.write(":WMEMory2:SAVE CHANnel2")
wait_io_operation()
```

```
# End of Script
# ------------------------------------------------------------------------
KsInfiniiVisionX.clear()   # Clear communications interface
KsInfiniiVisionX.close()   # Close communications interface
print "All done."
```

38 Synchronizing Acquisitions

When remotely controlling an oscilloscope with programming commands, it is often necessary to know when the oscilloscope has finished the previous operation and is ready for the next command. The most common example is when an acquisition is started using the :DIGitize, :RUN, or :SINGle commands. Before a measurement result can be queried, the acquisition must complete. Too often fixed delays are used to accomplish this wait, but fixed delays often use excessive time or the time may not be long enough. A better solution is to use synchronous commands and status to know when the oscilloscope is ready for the next request.

Synchronization in the Programming Flow

Most remote programming follows these three general steps:

1. Set up the oscilloscope and device under test (see page 1334).
2. Acquire a waveform (see page 1334).
3. Retrieve results (see page 1334).

Set Up the Oscilloscope

Before making changes to the oscilloscope setup, it is best to make sure it is stopped using the :STOP command followed by the *OPC? query.

NOTE **It is not necessary to use *OPC?, hard coded waits, or status checking when setting up the oscilloscope. After the oscilloscope is configured, it is ready for an acquisition.**

Acquire a Waveform

When acquiring a waveform there are two possible methods used to wait for the acquisition to complete. These methods are blocking and polling. The table below details when each method should be chosen and why.

	Blocking Wait	Polling Wait
Use When	You know the oscilloscope *will* trigger based on the oscilloscope setup and device under test.	You know the oscilloscope *may or may not* trigger on the oscilloscope setup and device under test.
Advantages	No need for polling. Fastest method.	Remote interface will not timeout No need for device clear if no trigger.
Disadvantages	Remote interface may timeout. Device clear only way to get control of oscilloscope if there is no trigger.	Slower method. Requires polling loop. Requires known maximum wait time.
Implementation Details	See "Blocking Synchronization" on page 1335.	See "Polling Synchronization With Timeout" on page 1336.

Retrieve Results

Once the acquisition is complete, it is safe to retrieve measurements and statistics.

Blocking Synchronization

Use the :DIGitize command to start the acquisition. This blocks subsequent queries until the acquisition and processing is complete. For example:

```
'
' Synchronizing acquisition using blocking.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Set up.
  ' ---------------------------------------------------------------
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Acquire.
  ' ---------------------------------------------------------------
  myScope.WriteString ":DIGitize"

  ' Get results.
  ' ---------------------------------------------------------------
  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Polling Synchronization With Timeout

This example requires a timeout value so the operation can abort if an acquisition does not occur within the timeout period:

```
'
' Synchronizing acquisition using polling.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Set up.
  ' --------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Acquire.
  ' --------------------------------------------------------------------
  ' Start a single acquisition.
  myScope.WriteString ":SINGle"

  ' Oscilloscope is armed and ready, enable DUT here.
  Debug.Print "Oscilloscope is armed and ready, enable DUT."

  ' Look for RUN bit = stopped (acquisition complete).
  Dim lngTimeout As Long   ' Max millisecs to wait for single-shot.
  Dim lngElapsed As Long
  lngTimeout = 10000   ' 10 seconds.
  lngElapsed = 0

  Do While lngElapsed <= lngTimeout
```

```
    myScope.WriteString ":OPERegister:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Mask RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get results.
  ' ------------------------------------------------------------------
  If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber   ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
  Else
    Debug.Print "Timeout waiting for single-shot trigger."
  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Synchronizing with a Single-Shot Device Under Test (DUT)

The examples in "Blocking Synchronization" on page 1335 and "Polling Synchronization With Timeout" on page 1336 assume the DUT is continually running and therefore the oscilloscope will have more than one opportunity to trigger. With a single shot DUT, there is only one opportunity for the oscilloscope to trigger, so it is necessary for the oscilloscope to be armed and ready before the DUT is enabled.

NOTE **The blocking :DIGitize command cannot be used for a single shot DUT because once the :DIGitize command is issued, the oscilloscope is blocked from any further commands until the acquisition is complete.**

This example is the same "Polling Synchronization With Timeout" on page 1336 with the addition of checking for the armed event status.

```
'
' Synchronizing single-shot acquisition using polling.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Set up.
  ' ---------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Acquire.
```

```
  ' ------------------------------------------------------------------
  ' Start a single acquisition.
  myScope.WriteString ":SINGle"

  ' Wait until the trigger system is armed.
  Do
    Sleep 100   ' Small wait to prevent excessive queries.
    myScope.WriteString ":AER?"
    varQueryResult = myScope.ReadNumber
  Loop Until varQueryResult = 1

  ' Oscilloscope is armed and ready, enable DUT here.
  Debug.Print "Oscilloscope is armed and ready, enable DUT."

  ' Now, look for RUN bit = stopped (acquisition complete).
  Dim lngTimeout As Long   ' Max millisecs to wait for single-shot.
  Dim lngElapsed As Long
  lngTimeout = 10000   ' 10 seconds.
  lngElapsed = 0

  Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERegister:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Mask RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get results.
  ' ------------------------------------------------------------------
  If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber   ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
  Else
    Debug.Print "Timeout waiting for single-shot trigger."
  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Synchronization with an Averaging Acquisition

When averaging, it is necessary to know when the average count has been reached. The :SINGle command does not average.

If it is known that a trigger will occur, a :DIGitize will acquire the complete number of averages, but if the number of averages is large, a timeout on the connection can occur.

The example below polls during the :DIGitize to prevent a timeout on the connection.

```
'
' Synchronizing in averaging acquisition mode.
' ====================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

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::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 5000

  ' Set up.
  ' --------------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:SWEep NORMal"
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Set up average acquisition mode.
  Dim lngAverages As Long
  lngAverages = 256
  myScope.WriteString ":ACQuire:COUNt " + CStr(lngAverages)
  myScope.WriteString ":ACQuire:TYPE AVERage"
```

```
  ' Save *ESE (Standard Event Status Enable register) mask
  ' (so it can be restored later).
  Dim varInitialESE As Variant
  myScope.WriteString "*ESE?"
  varInitialESE = myScope.ReadNumber

  ' Set *ESE mask to allow only OPC (Operation Complete) bit.
  myScope.WriteString "*ESE " + CStr(CInt("&H01"))

  ' Acquire using :DIGitize.  Set up OPC bit to be set when the
  ' operation is complete.
  ' ------------------------------------------------------------------
  myScope.WriteString ":DIGitize"
  myScope.WriteString "*OPC"

  ' Assume the oscilloscope will trigger, if not put a check here.

  ' Wait until OPC becomes true (bit 5 of Status Byte register, STB,
  ' from Standard Event Status register, ESR, is set).  STB can be
  ' read during :DIGitize without generating a timeout.
  Do
    Sleep 4000   ' Poll more often than the timeout setting.
    varQueryResult = myScope.IO.ReadSTB
  Loop While (varQueryResult And &H20) = 0

  ' Clear ESR and restore previously saved *ESE mask.
  myScope.WriteString "*ESR?"   ' Clear ESR by reading it.
  varQueryResult = myScope.ReadNumber
  myScope.WriteString "*ESE " + CStr(varInitialESE)

  ' Get results.
  ' ------------------------------------------------------------------
  myScope.WriteString ":WAVeform:COUNt?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Averaged waveforms: " + CStr(varQueryResult)

  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Example: Blocking and Polling Synchronization

```
# -*- coding: utf-8 -*-

# ******************************************************************
# This script using the Python language (http://www.python.org/) and
# the PyVISA package (http://pyvisa.readthedocs.org/) shows the two
# best synchronization methods for InfiniiVision oscilloscopes.
# Benefits and drawbacks of each method are described.  No error
# handling is provided except in the actual synchronization methods.
# ******************************************************************

# Import modules
# ----------------------------------------------------------------------
import sys
import visa
import time

# Initialization constants
# ----------------------------------------------------------------------
# Get VISA address from Keysight IO Libraries Connection Expert
VISA_ADDRESS = \
    "TCPIP0::a-mx4154a-60014.cos.is.keysight.com::inst0::INSTR"
GLOBAL_TOUT =  10000   # IO timeout in milliseconds

TIME_TO_TRIGGER = 10   # Time in seconds
# ----------------------------------------------------------------------
# This is the time until the FIRST trigger event.
#
# While the script calculates a general timeout for the given setup,
# it cannot know when a trigger event will occur.  Thus, you must
# still set this value.
#
# This time is in addition to the calculated minimum timeout... so, if
# an oscilloscope might take say, 1 us to arm and acquire data, the
# signal might take 100 seconds before it occurs... this accounts for
# that.
#
# The SCOPE_ACQUISITION_TIME_OUT calculation pads this by 1.1.
# ----------------------------------------------------------------------

TIME_BETWEEN_TRIGGERS = 0.025   # Time in seconds - for Average,
# Segmented, and Equivalent Time modes, else set to 0
# ----------------------------------------------------------------------
# In Average, Segmented, and Equivalent Time modes, the oscilloscope
# makes repeated acquisitions.  This is similar to the above
# TIME_TO_TRIGGER, but it is the time BETWEEN triggers.  For example,
# it might take 10 seconds for the first trigger event, and then they
# might start occurring regularly at say, 1 ms intervals.  In that
# scenario, 15 seconds (a conservative number for 10s) would be good
# for TIME_TO_TRIGGER, and 2 ms (again conservative) would be good for
# TIME_BETWEEN_TRIGGERS.
#
# The default in this sample script is 0.025 seconds.  This is to make
# the sample work for the LINE trigger used in this script when the
```

```
# oscilloscope is in Average, Segmented, and Equivalent Time modes to
# force a trigger off of the AC input line (:TRIGger:EDGE:SOURce LINE)
# which runs at 50 or 60 Hz in most of the world (1/50 Hz -> 20 ms, so
# use 25 ms to be conservative).
#
# The SCOPE_ACQUISITION_TIME_OUT calculation pads this by 1.1.
# ------------------------------------------------------------------


# ==================================================================
# Define a simple and fast function utilizing the blocking :DIGitize
# command in conjunction with *OPC?.
# ------------------------------------------------------------------
#
# Benefits of this method:
#
#   -  Fastest, compact
#   -  Only way for Average Acquisition type:
#         -  The :SINGle command does not do a complete average.
#         -  Counting triggers with :RUN is much too slow.
#   -  Allows for easy synchronization with math functions
#   -  Don't have to deal with the status registers, which can be
#      confusing.
#   -  Potentially faster than polling_method(), for better throughput
#   -  Because it's faster, one can retrieve more accurate acquisition
#      times than with a polling method.
#   -  Works best for segmented memory if any post processing is done
#      on the oscilloscope, for example, measurements, lister, math,
#      as this does not come back until the processing is all done
#         -  In this scenario, :DIGitize does not reduce the sample
#            rate or memory depth.
#
# Drawbacks of this method:
#
#   -  Usually does not fill acquisition memory to the maximum
#      available, usually only on-screen data.
#
#   -  May not be at the highest sample rate (compared with the
#      polling_method).
#
#   -  Requires a well-chosen, hard-set timeout that will cover the
#      time to arm, trigger, and finish acquisition.
#
#   -  Requires Exception handling and a device_clear() for a
#      possible timeout (no trigger event).
#
#   -  Socket connection cannot do device_clear()
#
#   -  Because :DIGitize is a "specialized form of the :RUN" command,
#      on these oscilloscopes, that results in:
#
#         -  the sample rate MAY be reduced from using :SINGle -
#            usually at longer time scales - typically only acquires
#            what is on screen, though at the fastest time scales,
#            more than on screen data may be acquired.  Thus, for max
#            memory and max sample rate, use the polling_method(),
#            which uses :SINGle.
```

```
#
# How it works:
#
#   -  The :DIGitize command is a blocking command, and thus, all
#      other SCPI commands are blocked until :DIGitize is completely
#      done.  This includes any subsequent processing that is already
#      set up, such as math and measurements.
#
#      KEY POINT: However, :DIGitize does not prevent additional
#      commands from being sent to the queue or cause the remote
#      program to wait.  For example, if your program does something
#      like:
#
#         KsInfiniiVisionX.write(":DIGitize")
#         print("Signal acquired.\n")
#
#      The "Signal acquired" message will be written immediately
#      after the :DIGitize is sent, not after the acquisition and
#      processing is complete.
#
#      To pause the program until the :DIGitize is complete, you must
#      wait for a query result after the :DIGitize.  For example, in
#      this case:
#
#         query_result = KsInfiniiVisionX.query(":DIGitize;*OPC?")
#         print("Signal acquired.\n")
#
#      The "Signal acquired" message will be written after the
#      acquisition and processing is complete.  The *OPC? query is
#      appended to :DIGitize with a semi-colon (;), which
#      essentially ties it to the same thread in the parser. It is
#      immediately dealt with once :DIGitize finishes and gives a "1"
#      back to the program (whether the program uses it or not),
#      allowing the program to move on.
#
# Other Notes:
#
#   -  If you DO NOT know when a trigger will occur, you should set a
#      very long timeout.
#
#   -  The timeout should be adjusted before and after the :DIGitize
#      operation, though this is not absolutely required.
#
#   -  A :DIGitize can be aborted with a device clear, which also
#      stops the oscilloscope:
#      KsInfiniiVisionX.clear()
#
#   -  :DIGItize disables the anti-aliasing feature (sample rate
#      dither) on all InfiniiVision and InfiniiVision X-Series
#      oscilloscopes.
#
#   -  :DIGitize temporarily blocks the front panel, and all front
#      panel presses are queued until :DIGitize is done.  So if you
#      change the vertical scale, it will not happen until the
#      acquisition is done.
#
#      The exception is that the Run/Stop button on the front panel
```

```
#      is NOT blocked (unless the front panel is otherwise locked by
#      ":SYSTem:LOCK ON").
# ======================================================================
def blocking_method():

    KsInfiniiVisionX.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,
    # KsInfiniiVisionX.
    #
# If doing repeated acquisitions, this should be done BEFORE the
    # loop, and changed again after the loop if the goal is to
    # achieve the best throughput.

    print("Acquiring signal(s)...")
    # Set up a try/except block to catch a possible timeout and exit.
    try:
        KsInfiniiVisionX.query(":DIGitize;*OPC?")
        # Acquire the signal(s) with :DIGItize (blocking) and wait
        # until *OPC? comes back with a one.
        print("Signal acquired.")

        # Reset timeout back to what it was, GLOBAL_TOUT.
        KsInfiniiVisionX.timeout =  GLOBAL_TOUT

    # Catch a possible timeout and exit.
    except Exception:
        print "The acquisition timed out, most likely due to no " \
              "trigger or improper setup causing no trigger. " \
              "Properly closing the oscilloscope connection and " \
              "exiting script.\n"
        KsInfiniiVisionX.clear() # Clear communications interface;
                                 # A device clear also aborts digitize.
        KsInfiniiVisionX.close() # Close communications interface
        sys.exit("Exiting script.")


# ======================================================================
# Define a function using the non-blocking :SINGle command and polling
# on the Operation Status Condition Register
# ----------------------------------------------------------------------
#
# Benefits of this method:
#
#  -  Don't have to worry about interface timeouts.
#  -  Easy to expand to know when the oscilloscope is armed.
#  -  Don't have to worry about interface timeouts
#  -  Easy to expand to know when scope is armed, and triggered
#  -  MAY result in a higher sample rate than the blocking method
#  -  Always fills max available memory
#  -  Can use with a socket connection if desired
#
#
# Drawbacks of this method:
```

```
#
#  -  Slow, as you don't want to poll the oscilloscope too fast.
#
#  -  DOES NOT work for Equivalent time mode.  MUST use the blocking
#     method.
#
#  -  Slow
#
#  -  Does NOT work for Average Acquisition type
#       -  :SINGle does not do a complete average
#            -  It does a single acquisition as if it were in NORMal
#               acq. type
#            -  Counting triggers in :RUN is much too slow
#
#  -  Works for Segmented Memory, BUT if any post processing is done
#     on the oscilloscope, for example measurements, lister, math, as
#     this reports that the acquisition is done, which is correct,
#     BUT the processing is NOT done, and it will take an indefinite
#     amount of time to wait for that, though there is no way to tell
#     if it is done.  Use the blocking_method for Segmented Memory.
#
#  -  Can't be used effectively for synchronizing math functions
#
#     It can be done by applying an additional hard coded wait after
#     the acquisition is done.  At least 200 ms is suggested, more
#     may be required.
#
#     However, as long as the timeout is not excessively short, the
#     math happens fast enough that once :OPERegister:CONDition?
#     comes back as done that one can just wait for it when it is
#     time to pull the math waveform.  The exception would be for eye
#     or jitter mode on a 6000 X-Series oscilloscope, where the
#     processing time can be long.
#
#  -  Still need some maximum timeout (here MAX_TIME_TO_WAIT),
#     ideally, or the script will sit in the while loop forever if
#     there is no trigger event
#
#     Max timeout (here MAX_TIME_TO_WAIT) must also account for any
#     processing done (see comments on math above)
#
#     Max timeout (here MAX_TIME_TO_WAIT) must also account for time
#     to arm the scope and finish the acquisition
#
#     This arm/trigger/finish part is accounted for in the main script.
#
# How it works:
#
#  -  What really matters is the RUN bit in the Operation Condition
#     (not Event) Register.  This bit changes based on the
#     oscilloscope state.
#
#     If the oscilloscope is running, it is high (8), and low (0) if
#     it is stopped.
#
#     The only (best) way to get at this bit is with the
#     :OPERation:CONDition? query.  The Operation Condition Register
```

```
#     can reflect states for other oscilloscope properties, for
#     example, if the oscilloscope is armed, thus it can produce
#     values other than 0 (stopped) or 8 (running).
#
#     To handle that, the result of :OPERation:Condition? is bitwise
#     ANDed (& in Python) with an 8.  This is called "unmasking".
#
#     Here, the "unmasking" is done in the script.  On the other
#     hand, it is possible to "mask" which bits get passed to the
#     summary bit to the next register below on the instrument
#     itself.  However, this method it typically only used when
#     working with the Status Byte, and not used here.
#
#  -  Why 8 = running = not done?
#
#     The Run bit is the 4th bit of the Operation Status Condition
#     (and Event) Registers.
#
#     The registers are binary and start counting at zero, thus the
#     4th bit is bit number 3, and 2^3 = 8, and thus it returns an
#     8 for high and a 0 for low.
#
#  -  Why the CONDITION and NOT the EVENT register?
#
#     The Condition register reflects the CURRENT state, while the
#     EVENT register reflects the first event that occurred since it
#     was cleared or read (as in: has it EVER happened?), thus the
#     CONDITION register is used.
#
#     Note that with this method using :SINGle, for InfiniiVision
#     X-Series oscilloscopes only, :SINGle itself forces the trigger
#     sweep mode into NORMal.  This does not happen with the blocking
#     method, using :DIGitize, or on the InfiniiVision notXs.
# ====================================================================
def polling_method():

    MAX_TIME_TO_WAIT = SCOPE_ACQUISITION_TIME_OUT/float(1000)
    # Time in seconds to wait for the oscilloscope to arm, trigger,
    # and finish acquisition.
    #
    # Note that this is NOT a property of the device interface,
    # KsInfiniiVisionX, but rather some constant in the script to be
    # used later with the Python module "time", and will be used with
    # time.clock().

    # Define "mask" bits.
    #
    # Mask condition for Run state in the Operation Status Condition
    # (and Event) Register.
    #
    # Refer to Programmer's Guide chapters on Status Reporting and
    # Synchronizing Acquisitions.
    RUN_BIT = 3
    # The run bit is the 4th bit (see next set of comments @
    # Completion Criteria).
    RUN_MASK = 1<<RUN_BIT
    # This basically means:  2^3 = 8, or rather, in Python 2**3
```

```
# (<< is a left shift; left shift is fastest); this is used later
# to "unmask" the result of the Operation Status Event Register
# as there is no direct access to the RUN bit.

# Define completion criterion:
ACQ_DONE = 0
# Means the oscilloscope is stopped
ACQ_NOT_DONE = 1<<RUN_BIT
# Means the oscilloscope is running; value is 8.  This is the
# 4th bit of the Operation Status Condition (and Event) Register.
# The registers are binary and start counting at zero, thus the
# 4th bit (4th position in a binary representation of decimal
# 8 = 2^3 = (1 left shift 3).  This is either High (running = 8)
# or low (stopped and therefore done with acquisition = 0).

print "Acquiring signal(s)..."

# Define acquisition start time.  This is in seconds.
StartTime = time.clock()

# Begin Acquisition with *CLS and the non-blocking :SINGle
# command, concatenated together. The *CLS clears all (non-mask)
# registers & sets them to 0;
KsInfiniiVisionX.write("*CLS;:SINGle")

# Initialize the loop entry condition (assume Acq. is not done).
Acq_State = ACQ_NOT_DONE

# Poll the oscilloscope until Acq_State is a one. (This is NOT a
# "Serial Poll.")
while Acq_State == ACQ_NOT_DONE and (time.clock() - StartTime
                                        <= MAX_TIME_TO_WAIT):

    # Ask oscilloscope if it is done with the acquisition via the
    # Operation Status Condition (not Event) Register.
    # The Condition register reflects the CURRENT state, while
    # the EVENT register reflects the first event that occurred
    # since it was cleared or read, thus the CONDITION register
    # is used.
    #
    # DO NOT do:
    # KsInfiniiVisionX.query("*CLS;SINGle;OPERegister:CONDition?")
    # as putting :OPERegister:CONDition? right after :SINgle
    # doesn't work reliably
    #
    # The oscilloscope SHOULD trigger, but it sits there with the
    # Single hard key on the oscilloscope lit yellow; pressing
    # this key causes a trigger.
    Status = int(KsInfiniiVisionX.query(":OPERegister:CONDition?"))

    # Bitwise AND of the Status and RUN_MASK.  This exposes ONLY
    # the 4th bit, which is either High (running = 8) or low
    # (stopped and therefore done with acquisition = 0)
    Acq_State = (Status & RUN_MASK)
    if Acq_State == ACQ_DONE:
        break # Break out of while loop so that the 100 ms pause
        # below is not incurred if done.
```

```
            time.sleep(0.1)   # Pause 100 ms to prevent excessive queries
            # This can actually be set a little faster, at 0.045.  The
            # point here is that:
            #
            #   1.  If there are other things being controlled, going too
            #       fast can tie up the bus.
            #   2.  Going faster does not work on all scopes.  The
            #       symptom of this not working is:
            #
            # The oscilloscope SHOULD trigger, but it sits there with the
            # Single hard key on the oscilloscope lit yellow; pressing
            # this key causes a trigger.
            #
            # The pause should be at the end of the loop, so that the
            # oscilloscope is immediately asked if it is done.
            #
            # Loop exits when Acq_State != NOT_DONE, that is, it exits
            # the loop when it is DONE or if the max wait time is
            # exceeded.

        if Acq_State == ACQ_DONE: # Acquisition fully completed
            print "Signal acquired."
        else: # Acquisition failed for some reason
            print "Max wait time exceeded."
            print "This happens if there was no trigger event."
            print "Adjust settings accordingly.\n"
            print "Properly closing oscilloscope connection and exiting " \
                  "script.\n"
            KsInfiniiVisionX.clear() # Clear communications interface
            KsInfiniiVisionX.query(":STOP;*OPC?")
            # Stop the oscilloscope
            KsInfiniiVisionX.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(KsInfiniiVisionX.query("MEASure:VPP? CHANnel1")).strip("\n")
        # The result has a newline, so remove it with .strip("\n")
        print "Vpp Ch1 = " + Vpp_Ch1 + " V\n"


    # ================================================================
    # Main code
    # ================================================================

    # Connect and initialize oscilloscope
    # ----------------------------------------------------------------
    # Define VISA Resource Manager & Install directory
    rm = visa.ResourceManager('C:\\Windows\\System32\\agvisa32.dll')

    # Define and open the oscilloscope using the VISA address
```

```
try:
    KsInfiniiVisionX = rm.open_resource(VISA_ADDRESS)
except Exception:
    print "Unable to connect to oscilloscope at " + str(VISA_ADDRESS)\
          + ". Aborting script.\n"
    sys.exit()

# Set the Global Timeout
KsInfiniiVisionX.timeout = GLOBAL_TOUT

# Clear the instrument bus
KsInfiniiVisionX.clear()

# Clear all status registers and errors
KsInfiniiVisionX.write("*CLS")

try:

    # Set up the oscilloscope
    # ------------------------------------------------------------------
    # Note that you would normally perform a reset (default setup) if
    # you were to create the setup from scratch... But here we will
    # use the oscilloscope "as is" for the most part.
    # KsInfiniiVisionX.query("*RST;*OPC?")   # Resets the oscilloscope

    # Always stop the oscilloscope when making any changes.
    KsInfiniiVisionX.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.
    KsInfiniiVisionX.write(":TRIGger:SWEep NORMal")
    # This line simply gives the oscilloscope something to trigger on.
    KsInfiniiVisionX.query(":TRIGger:EDGE:SOURce LINE;*OPC?")

    # Clear the display (so you can see the waveform being acquired -
    # otherwise, there is no need for this).
    KsInfiniiVisionX.write(":CDISplay")

    # Calculate acquisition timeout/wait time by short, overestimate
    # method
    # ------------------------------------------------------------------

    # Create some default variables
    N_AVERAGES = 1
    N_SEGMENTS = 1

    # Get some info about the scope time base setup
    HO         = float(KsInfiniiVisionX.query(":TRIGger:HOLDoff?"))
    T_RANGE    = float(KsInfiniiVisionX.query(":TIMebase:RANGe?"))
    T_POSITION = float(KsInfiniiVisionX.query(":TIMebase:POSition?"))

    # Determine Acquisition Type and Mode:
    ACQ_TYPE = str(KsInfiniiVisionX.query(":ACQuire:TYPE?").strip("\n"))
    ACQ_MODE = str(KsInfiniiVisionX.query(":ACQuire:MODE?").strip("\n"))

    if ACQ_MODE == "SEGM":
```

```
    N_SEGMENTS = \
        float(KsInfiniiVisionX.query(":ACQuire:SEGMented:COUNt?"))
    # Note that if there are a lot of analysis associated segments,
    # for example, serial data decode, the timeout will likely need
    # to be longer than calculated.
    #
    # You are encouraged to manually set up the oscilloscope in
    # this case, as it will be used, time it, and use that, with
    # a little overhead.
    #
    # Blocking method is recommended for Segmented Memory mode.
elif ACQ_TYPE == "AVER":
    N_AVERAGES = float(KsInfiniiVisionX.query(":ACQuire:COUNt?"))

# Calculate acuisition timeout by overestimate method:
# Recall that PyVisa timeouts are in ms, so multiply by 1000.
SCOPE_ACQUISITION_TIME_OUT = (float(TIME_TO_TRIGGER)*1.1 +
    (T_RANGE*2.0 + abs(T_POSITION)*2.0 + HO*1.1 +
    float(TIME_BETWEEN_TRIGGERS)*1.1)*N_SEGMENTS*N_AVERAGES)*1000.0

## Ensure the timeout is no less than 10 seconds
if SCOPE_ACQUISITION_TIME_OUT < 10000.0:
    SCOPE_ACQUISITION_TIME_OUT = 10000.0

# What about Equivalent Time Mode and other odd modes such as
# Jitter or Eye (the last two only being found on the
# 6000 X-Series), and math functions?
#
# In most cases, the padding and 10 second minimum timeout will
# take care of this.
#
# Equivalent Time Mode has effects only at the fastest time
# scales, so it really doesn't make a difference as long as a
# trigger signal is present.  If trigger signal occurs rarely,
# adjust the TIME_BETWEEN_TRIGGERS constant accordingly.
#
# For math, the math will happen fast enough that the "padding"
# in the timeout calculation takes care of this.
#
# For jitter mode on the 6000 X-Series, you can try this method,
# typically there is always a signal present, and the 10 second
# minimum should work out.  If not, make it bigger, or increase
# padding.
#
# For Eye mode on the 6000 X-Series, none of this works anyway,
# and you have to use :RUN (or :RTEYe:ACQuire) and :STOP.

# Acquire Signal
# ----------------------------------------------------------------
# Choose blocking_method or polling_method.  These were defined as
# functions in case you want to use them repeatedly.
blocking_method()
# If Acquisition Type is Average, always use blocking_method() to
# get complete average.
do_something_with_data()

polling_method()
```

```
    do_something_with_data()

    # Done - cleanup
    KsInfiniiVisionX.clear() # Clear communications interface
    KsInfiniiVisionX.close() # Close communications interface
except KeyboardInterrupt:
    KsInfiniiVisionX.clear()
    KsInfiniiVisionX.query(":STOP;*OPC?")
    KsInfiniiVisionX.write(":SYSTem:LOCK OFF")
    KsInfiniiVisionX.clear()
    KsInfiniiVisionX.close()
    sys.exit("User Interupt.  Properly closing oscilloscope and "
             "aborting script.")
except Exception:
    KsInfiniiVisionX.clear()
    KsInfiniiVisionX.query(":STOP;*OPC?")
    KsInfiniiVisionX.write(":SYSTem:LOCK OFF")
    KsInfiniiVisionX.clear()
    KsInfiniiVisionX.close()
    sys.exit("Something went wrong.  Properly closing oscilloscope "
             "and aborting script.")

# End of Script
# ------------------------------------------------------------------------
print "Done."
```

39 More About Oscilloscope Commands

Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of Keysight InfiniiVision oscilloscopes, commands are classified by the following categories:

- "Core Commands" on page 1354
- "Non-Core Commands" on page 1354
- "Obsolete Commands" on page 1354

Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Keysight InfiniiVision oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Keysight InfiniiVision oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Keysight's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

Obsolete Commands

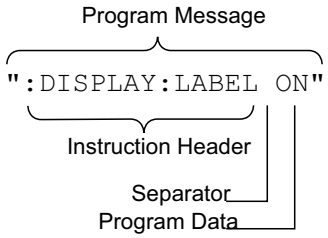
Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

- Chapter 35, “Obsolete and Discontinued Commands,” starting on page 1249

Valid Command/Query Strings

Program Message Syntax

To program the instrument remotely, you must understand the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. The following figure shows the main syntactical parts of a typical program statement.



Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as Visual Basic or C/C++. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as <learn string>. There are only a few instructions that use block data.

Program messages can have long or short form commands (and data in some cases – see "Long Form to Short Form Truncation Rules" on page 1356), and upper and/or lower case ASCII characters may be used. (Query responses, however, are always returned in upper case.)

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provide additional information needed to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument.

":DISPlay:LABel ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, ":DISPlay:LABel?". Many instructions can be used as either commands or queries, depending on whether or

not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- "Simple Command Headers" on page 1357
- "Compound Command Headers" on page 1357
- "Common Command Headers" on page 1357

White Space (Separator) White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. White space is defined as one or more space characters. ASCII defines a space to be character 32 (in decimal).

Program Data Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. "Program Data Syntax Rules" on page 1358 describes all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas(,). Spaces can be added around the commas to improve readability.

Program Message Terminator The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the programming interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

NOTE **New Line Terminator Functions.** The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGe	RANG
PATTern	PATT

Long Form	Short form
TIMebase	TIM
DELay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of controller memory needed for program storage and reduces I/O activity.

Simple Command Headers

Simple command headers contain a single mnemonic. :AUToscale and :DIGitize are examples of simple command headers typically used in the oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGitize CHANnel1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLimit ON

Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data><terminator>
```

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

```
<program mnemonic><separator><data>,<data><terminator>
```

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMebase:MODE command can be set to normal, zoomed (delayed), XY, or ROLL. The character program data in this case may be MAIN, WINDow, XY, or ROLL. The command :TIMebase:MODE WINDow sets the time base mode to zoomed.

The available mnemonics for character program data are always included with the command's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMebase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

```
28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.
```

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGe may be used to change the vertical range or to change the horizontal range:

```
:CHANnel1:RANGe .4
```

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMebase:RANGe 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMebase are subsystem selectors and determine which range is being modified.

Tree Traversal Rules and Multiple Commands

Command headers are created by traversing down the command tree. A legal command header would be :TIMebase:RANGe. This is referred to as a *compound header*. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces.

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.
- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Keysight VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

```
:<subsystem>:<function><separator><data>;<function><separator><data><ter
minator>
```

For example:

```
myScope.WriteString ":TIMebase:RANGe 0.5;POSition 0"
```

NOTE

The colon between TIMebase and RANGe is necessary because TIMebase:RANGe is a compound command. The semicolon between the RANGe command and the POSition command is the required program message unit separator. The POSition command does not need TIMebase preceding it because the TIMebase:RANGe command sets the parser to the TIMebase node in the tree.

Example 2: Program Message Terminator Sets Parser Back to Root

```
myScope.WriteString ":TIMebase:REFerence CENTer;POSition 0.00001"
```

or

```
myScope.WriteString ":TIMebase:REFerence CENTer"
myScope.WriteString ":TIMebase:POSition 0.00001"
```

NOTE

In the first line of example 2, the subsystem selector is implied for the POSition command in the compound command. The POSition command must be in the same program message as the REFerence command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMebase: before the POSition command as shown in the second part of example 2. The space after POSition is required.

Example 3: Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

For example:

```
myScope.WriteString ":TIMebase:REFerence CENTer;:DISPlay:VECTors ON"
```

NOTE

The leading colon before DISPlay:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:VECTors ON command. The space between REFerence and CENter is required; so is the space between VECTors and ON.

Multiple commands may be any combination of compound and simple commands.

Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMebase:RANGe? places the current time base setting in the output queue. When using the Keysight VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String
myScope.WriteString ":TIMebase:RANGe?"
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

NOTE **Read Query Results Before Sending Another Command.** Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

Infinity Representation

The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

40 Programming Examples

The example programs in this manual are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

See Also

- You can find additional programming examples for the InfiniiVision M9241/42/43A PXIe oscilloscopes on the Keysight Technologies website at: www.keysight.com/find/M924x-examples

VISA COM Examples

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Keysight VISA COM library:
   a. Choose **Tools>References...** from the main menu.
   b. In the References dialog, check the "VISA COM 5.5 Type Library".
   c. Click **OK**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the VISA address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Keysight VISA COM Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

' For Sleep subroutine.
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::10.112.94.136::hislip9-0.0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 10000   ' Set I/O communication timeout.

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  Debug.Print "Identification string: " + strQueryResult

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Capture the waveform.
' ---------------------------------------------------------------------

Private Sub Capture()
```

```
On Error GoTo VisaComError

' Use auto-scale to automatically configure oscilloscope.
' ------------------------------------------------------------------
DoCommand ":AUToscale"

' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
    DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURce CHANnel1"
Debug.Print "Trigger edge source: " + _
    DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
    DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
    DoQueryString(":TRIGger:EDGE:SLOPe?")

' Save oscilloscope configuration.
' ------------------------------------------------------------------
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 settings with individual commands:
' ------------------------------------------------------------------

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and offset.
DoCommand ":TIMebase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMebase:SCALe?")

DoCommand ":TIMebase:POSition 0.0"
Debug.Print "Timebase position: " + _
```

```
      DoQueryString(":TIMebase:POSition?")

  ' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
  DoCommand ":ACQuire:TYPE NORMal"
  Debug.Print "Acquire type: " + _
      DoQueryString(":ACQuire:TYPE?")

  ' 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))

  ' Capture an acquisition using :DIGitize.
  ' --------------------------------------------------------------------
  DoCommand ":DIGitize CHANnel1"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Analyze the captured waveform.
' --------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' Make a couple of measurements.
  ' --------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  varQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(varQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  varQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(varQueryResult, 4) + " V"


  ' Download the screen image.
```

```
' ----------------------------------------------------------------------
' Get screen image.
DoCommand ":HARDcopy:INKSaver OFF"
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG, COLor")

' 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.
' ----------------------------------------------------------------------

' Set the waveform points mode.
DoCommand ":WAVeform:POINts:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINts:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
```

```
Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 4 Then
  Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMal"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
  Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
varQueryResult = DoQueryIEEEBlock_UI1(":WAVeform:DATA?")
```

```
  Debug.Print "Number of data values: " + _
      CStr(UBound(varQueryResult) + 1)

  ' 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 - lngYReference) * _
        sngYIncrement) + sngYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo VisaComError

  myScope.WriteString command
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)
```

```
  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteIEEEBlock command, data
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Function DoQueryString(query As String) As String

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryString = myScope.ReadString
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumber(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryNumber = myScope.ReadNumber
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

  On Error GoTo VisaComError
```

```
  Dim strErrors As String

  myScope.WriteString query
  DoQueryNumbers = myScope.ReadList
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo VisaComError

  Dim strErrVal As String
  Dim strOut As String

  myScope.WriteString ":SYSTem:ERRor?"    ' Query any errors data.
  strErrVal = myScope.ReadString          ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0               ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    myScope.WriteString ":SYSTem:ERRor?"   ' Request error message.
    strErrVal = myScope.ReadString         ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    myScope.FlushWrite (False)
    myScope.FlushRead

  End If

  Exit Sub
```

```
VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + Err.Description

End Sub
```

VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 5.5 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 Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 5.5 Type Library**; then click **OK**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite 16.

```
/*
 * Keysight VISA COM Example in C#
 * ------------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * ------------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
```

```
            VisaComInstrument("TCPIP0::10.112.94.136::hislip9-0.0::INSTR")
;
        myScope.SetTimeoutSeconds(10);

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();
      }
      catch (System.ApplicationException err)
      {
        Console.WriteLine("*** VISA COM Error : " + err.Message);
      }
      catch (System.SystemException err)
      {
        Console.WriteLine("*** System Error Message : " + err.Message);
      }
      catch (System.Exception err)
      {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
      }
      finally
      {
        myScope.Close();
      }
    }

    /*
     * Initialize the oscilloscope to a known state.
     * -------------------------------------------------------------
     */
    private static void Initialize()
    {
      string strResults;

      // Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?");
      Console.WriteLine("*IDN? result is: {0}", strResults);

      // Clear status and load the default setup.
      myScope.DoCommand("*CLS");
      myScope.DoCommand("*RST");
    }

    /*
     * Capture the waveform.
     * -------------------------------------------------------------
     */
    private static void Capture()
    {
      // Use auto-scale to automatically configure oscilloscope.
      myScope.DoCommand(":AUToscale");
```

```
        // Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
        myScope.DoCommand(":TRIGger:MODE EDGE");
        Console.WriteLine("Trigger mode: {0}",
            myScope.DoQueryString(":TRIGger:MODE?"));

        // Set EDGE trigger parameters.
        myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1");
        Console.WriteLine("Trigger edge source: {0}",
            myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

        myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
        Console.WriteLine("Trigger edge level: {0}",
            myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

        myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
        Console.WriteLine("Trigger edge slope: {0}",
            myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

        // Save oscilloscope configuration.
        byte[] ResultsArray;   // Results array.
        int nLength;   // Number of bytes returned from instrument.
        string strPath;

        // Query and read setup string.
        ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
        nLength = ResultsArray.Length;

        // Write setup string to file.
        strPath = "c:\\scope\\config\\setup.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 0.05");
        Console.WriteLine("Channel 1 vertical scale: {0}",
            myScope.DoQueryString(":CHANnel1:SCALe?"));

        myScope.DoCommand(":CHANnel1:OFFSet -1.5");
        Console.WriteLine("Channel 1 vertical offset: {0}",
            myScope.DoQueryString(":CHANnel1:OFFSet?"));

        // Set horizontal scale and offset.
        myScope.DoCommand(":TIMebase:SCALe 0.0002");
        Console.WriteLine("Timebase scale: {0}",
            myScope.DoQueryString(":TIMebase:SCALe?"));

        myScope.DoCommand(":TIMebase:POSition 0.0");
        Console.WriteLine("Timebase position: {0}",
            myScope.DoQueryString(":TIMebase:POSition?"));

        // Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution
    ).
```

```
  myScope.DoCommand(":ACQuire:TYPE NORMal");
  Console.WriteLine("Acquire type: {0}",
      myScope.DoQueryString(":ACQuire:TYPE?"));

  // Or, configure by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);
  nBytesWritten = DataArray.Length;

  // Restore setup string.
  myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Capture an acquisition using :DIGitize.
  myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make a couple of measurements.
  // ----------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"));

  double fResult;
  myScope.DoCommand(":MEASure:FREQuency");
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  myScope.DoCommand(":MEASure:VAMPlitude");
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

  // Download the screen image.
  // ----------------------------------------------------------
  myScope.DoCommand(":HARDcopy:INKSaver OFF");

  // Get the screen data.
  ResultsArray =
      myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor");
  nLength = ResultsArray.Length;

  // Store the screen data to a file.
  strPath = "c:\\scope\\data\\screen.png";
```

```
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"));

// Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
```

```
      Console.WriteLine("Acquire type: AVERage");
    }
    else if (fType == 3.0)
    {
      Console.WriteLine("Acquire type: HRESolution");
    }

    double fPoints = fResultsArray[2];
    Console.WriteLine("Waveform points: {0:e}", fPoints);

    double fCount = fResultsArray[3];
    Console.WriteLine("Waveform average count: {0:e}", fCount);

    double fXincrement = fResultsArray[4];
    Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

    double fXorigin = fResultsArray[5];
    Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

    double fXreference = fResultsArray[6];
    Console.WriteLine("Waveform X reference: {0:e}", fXreference);

    double fYincrement = fResultsArray[7];
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

    double fYorigin = fResultsArray[8];
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

    double fYreference = fResultsArray[9];
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

    // Read waveform data.
    ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?");
    nLength = ResultsArray.Length;
    Console.WriteLine("Number of data values: {0}", nLength);

    // 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)ResultsArray[i] - fYreference)
          * fYincrement) + fYorigin);

    // Close output file.
    writer.Close();
    Console.WriteLine("Waveform format BYTE data written to {0}",
        strPath);
  }
}
```

```
class VisaComInstrument
{
  private ResourceManagerClass m_ResourceManager;
  private FormattedIO488Class m_IoObject;
  private string m_strVisaAddress;

  // Constructor.
  public VisaComInstrument(string strVisaAddress)
  {
    // Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress;

    // Open the default VISA COM IO object.
    OpenIo();

    // Clear the interface.
    m_IoObject.IO.Clear();
  }

  public void DoCommand(string strCommand)
  {
    // Send the command.
    m_IoObject.WriteString(strCommand, true);

    // Check for inst errors.
    CheckInstrumentErrors(strCommand);
  }

  public void DoCommandIEEEBlock(string strCommand,
      byte[] DataArray)
  {
    // Send the command to the device.
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

    // Check for inst errors.
    CheckInstrumentErrors(strCommand);
  }

  public string DoQueryString(string strQuery)
  {
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result string.
    string strResults;
    strResults = m_IoObject.ReadString();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return results string.
    return strResults;
  }

  public double DoQueryNumber(string strQuery)
  {
    // Send the query.
```

```
  m_IoObject.WriteString(strQuery, true);

  // Get the result number.
  double fResult;
  fResult = (double)m_IoObject.ReadNumber(
    IEEEASCIIType.ASCIIType_R8, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return result number.
  return fResult;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result numbers.
  double[] fResultsArray;
  fResultsArray = (double[])m_IoObject.ReadList(
    IEEEASCIIType.ASCIIType_R8, ",;");

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return result numbers.
  return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  byte[] ResultsArray;
  ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_UI1, false, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  string strInstrumentError;
  bool bFirstError = true;

  do   // While not "0,No error".
  {
```

```
    m_IoObject.WriteString(":SYSTem:ERRor?", true);
    strInstrumentError = m_IoObject.ReadString();

    if (!strInstrumentError.ToString().StartsWith("+0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("+0,"));
}

private void OpenIo()
{
  m_ResourceManager = new ResourceManagerClass();
  m_IoObject = new FormattedIO488Class();

  // Open the default VISA COM IO object.
  try
  {
    m_IoObject.IO =
      (IMessage)m_ResourceManager.Open(m_strVisaAddress,
      AccessMode.NO_LOCK, 0, "");
  }
  catch (Exception e)
  {
    Console.WriteLine("An error occurred: {0}", e.Message);
  }
}

public void SetTimeoutSeconds(int nSeconds)
{
  m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
  try
  {
    m_IoObject.IO.Close();
  }
  catch { }

  try
  {
    Marshal.ReleaseComObject(m_IoObject);
  }
  catch { }

  try
  {
    Marshal.ReleaseComObject(m_ResourceManager);
  }
```

```
      catch { }
    }
  }
}
```

VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 5.5 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 Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 5.5 Type Library**; then click **OK**.
   e. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Keysight IO Libraries Suite 16.

```
'
' Keysight VISA COM Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New  _
          VisaComInstrument("TCPIP0::10.112.94.136::hislip9-0.0::INSTR")
```

```
    myScope.SetTimeoutSeconds(10)

    ' Initialize - start from a known state.
    Initialize()

    ' Capture data.
    Capture()

    ' Analyze the captured waveform.
    Analyze()

  Catch err As System.ApplicationException
    Console.WriteLine("*** VISA Error Message : " + err.Message)
  Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
  Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
  Finally
    myScope.Close()
  End Try
End Sub

' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------

Private Shared Sub Initialize()
  Dim strResults As String

  ' Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?")
  Console.WriteLine("*IDN? result is: {0}", strResults)

  ' Clear status and load the default setup.
  myScope.DoCommand("*CLS")
  myScope.DoCommand("*RST")

End Sub

' Capture the waveform.
' ---------------------------------------------------------------

Private Shared Sub Capture()

  ' Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale")

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE")
  Console.WriteLine("Trigger mode: {0}", _
      myScope.DoQueryString(":TRIGger:MODE?"))

  ' Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1")
  Console.WriteLine("Trigger edge source: {0}", _
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))
```

```
myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()   ' Results array.
Dim nLength As Integer   ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.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 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMebase:SCALe?"))

myScope.DoCommand(":TIMebase:POSition 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMebase:POSition?"))

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution)
.
myScope.DoCommand(":ACQuire:TYPE NORMal")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
```

```
        DataArray = File.ReadAllBytes(strPath)
        nBytesWritten = DataArray.Length

        ' Restore setup string.
        myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
        Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

        ' Capture an acquisition using :DIGitize.
        myScope.DoCommand(":DIGitize CHANnel1")

    End Sub

    ' Analyze the captured waveform.
    ' --------------------------------------------------------------

    Private Shared Sub Analyze()

        Dim fResult As Double
        Dim ResultsArray As Byte()   ' Results array.
        Dim nLength As Integer   ' Number of bytes returned from inst.
        Dim strPath As String

        ' Make a couple of measurements.
        ' --------------------------------------------------------------
        myScope.DoCommand(":MEASure:SOURce CHANnel1")
        Console.WriteLine("Measure source: {0}", _
            myScope.DoQueryString(":MEASure:SOURce?"))

        myScope.DoCommand(":MEASure:FREQuency")
        fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
        Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

        myScope.DoCommand(":MEASure:VAMPlitude")
        fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
        Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

        ' Download the screen image.
        ' --------------------------------------------------------------
        myScope.DoCommand(":HARDcopy:INKSaver OFF")

        ' Get the screen data.
        ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor
")
        nLength = ResultsArray.Length

        ' Store the screen data to a file.
        strPath = "c:\scope\data\screen.png"
        Dim fStream As FileStream
        fStream = File.Open(strPath, FileMode.Create)
        fStream.Write(ResultsArray, 0, nLength)
        fStream.Close()
        Console.WriteLine("Screen image ({0} bytes) written to {1}", _
            nLength, strPath)

        ' Download waveform data.
        ' --------------------------------------------------------------
```

```
' Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVeform:POINts:MODE?"))

' Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERage")
ElseIf fType = 3 Then
  Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)
```

```
    Dim fYincrement As Double = fResultsArray(7)
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

    Dim fYorigin As Double = fResultsArray(8)
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

    Dim fYreference As Double = fResultsArray(9)
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

    ' Get the waveform data.
    ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?")
    nLength = ResultsArray.Length
    Console.WriteLine("Number of data values: {0}", nLength)

    ' 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(ResultsArray(index)) - fYreference) _
          * fYincrement) + fYorigin)
    Next

    ' Close output file.
    writer.Close()
    Console.WriteLine("Waveform format BYTE data written to {0}", _
        strPath)

  End Sub

End Class

Class VisaComInstrument
  Private m_ResourceManager As ResourceManagerClass
  Private m_IoObject As FormattedIO488Class
  Private m_strVisaAddress As String

  ' Constructor.
  Public Sub New(ByVal strVisaAddress As String)

    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA COM IO object.
    OpenIo()

    ' Clear the interface.
```

```
  m_IoObject.IO.Clear()

End Sub

Public Sub DoCommand(ByVal strCommand As String)

  ' Send the command.
  m_IoObject.WriteString(strCommand, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte())

  ' Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result string.
  Dim strResults As String
  strResults = m_IoObject.ReadString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results string.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result number.
  Dim fResult As Double
  fResult = _
      CDbl(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return result number.
  Return fResult
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) As _
```

```
    Double()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result numbers.
  Dim fResultsArray As Double()
  fResultsArray = _
      m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return result numbers.
  Return fResultsArray
End Function

Public _
    Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the results array.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim ResultsArray As Byte()
  ResultsArray = _
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
      False, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results array.
  Return ResultsArray
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As String
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error".
    m_IoObject.WriteString(":SYSTem:ERRor?", True)
    strInstrumentError = m_IoObject.ReadString()

    If Not strInstrumentError.ToString().StartsWith("+0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ", _
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenIo()
  m_ResourceManager = New ResourceManagerClass()
  m_IoObject = New FormattedIO488Class()
```

```
    ' Open the default VISA COM IO object.
    Try
      m_IoObject.IO = _
          DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
              AccessMode.NO_LOCK, 0, ""), IMessage)
    Catch e As Exception
      Console.WriteLine("An error occurred: {0}", e.Message)
    End Try
  End Sub

  Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    m_IoObject.IO.Timeout = nSeconds * 1000
  End Sub

  Public Sub Close()
    Try
      m_IoObject.IO.Close()
    Catch
    End Try

    Try
      Marshal.ReleaseComObject(m_IoObject)
    Catch
    End Try

    Try
      Marshal.ReleaseComObject(m_ResourceManager)
    Catch
    End Try
  End Sub
 End Class
End Namespace
```

VISA COM Example in Python

You can use the Python programming language with the "comtypes" package to control Keysight oscilloscopes.

The Python language and "comtypes" package can be downloaded from the web at http://www.python.org/ and http://starship.python.net/crew/theller/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
```

```
#
# Keysight VISA COM Example in Python using "comtypes"
```

```
# ********************************************************
# This program illustrates a few commonly used programming
# features of your Keysight oscilloscope.
# ********************************************************

# Import Python modules.
# ----------------------------------------------------------
import string
import time
import sys
import array

from comtypes.client import GetModule
from comtypes.client import CreateObject

# Run GetModule once to generate comtypes.gen.VisaComLib.
if not hasattr(sys, "frozen"):
 GetModule("C:\Program Files (x86)\IVI Foundation\VISA\VisaCom\
GlobMgr.dll")

import comtypes.gen.VisaComLib as VisaComLib


# Global variables (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():

 # 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:EDGE:LEVel 1.5")
qresult = do_query_string(":TRIGger:EDGE:LEVel?")
print "Trigger edge level: %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?")
nLength = len(setup_bytes)
f = open("c:\scope\config\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 0.05")
qresult = do_query_number(":CHANnel1:SCALe?")
print "Channel 1 vertical scale: %f" % qresult

do_command(":CHANnel1:OFFSet -1.5")
qresult = do_query_number(":CHANnel1:OFFSet?")
print "Channel 1 offset: %f" % qresult

# Set horizontal scale and offset.
do_command(":TIMebase:SCALe 0.0002")
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 type.
do_command(":ACQuire:TYPE NORMal")
qresult = do_query_string(":ACQuire:TYPE?")
print "Acquire type: %s" % qresult

# Or, configure by loading a previously saved setup.
f = open("c:\scope\config\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)

# Capture an acquisition using :DIGitize.
do_command(":DIGitize CHANnel1")


# ==========================================================
# 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.
 # ----------------------------------------------------------
 do_command(":HARDcopy:INKSaver OFF")

 image_bytes = do_query_ieee_block(":DISPlay:DATA? PNG, COLor")
 nLength = len(image_bytes)
 f = open("c:\scope\data\screen.png", "wb")
 f.write(bytearray(image_bytes))
 f.close()
 print "Screen image written to c:\scope\data\screen.png."

 # Download waveform data.
 # ----------------------------------------------------------

 # Set the waveform points mode.
 do_command(":WAVeform:POINts:MODE RAW")
 qresult = do_query_string(":WAVeform:POINts:MODE?")
 print "Waveform points mode: %s" % qresult

 # Get the number of waveform points available.
 do_command(":WAVeform:POINts 10240")
 qresult = do_query_string(":WAVeform:POINts?")
 print "Waveform points available: %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 : "BYTE",
  1 : "WORD",
  4 : "ASCii",
 }
 acq_type_dict = {
  0 : "NORMal",
```

```
 1 : "PEAK",
 2 : "AVERage",
 3 : "HRESolution",
}

(
 wav_form,
 acq_type,
 wfmpts,
 avgcnt,
 x_increment,
 x_origin,
 x_reference,
 y_increment,
 y_origin,
 y_reference
) = do_query_numbers(":WAVeform:PREamble?")

print "Waveform format: %s" % wav_form_dict[wav_form]
print "Acquire type: %s" % acq_type_dict[acq_type]
print "Waveform points desired: %d" % wfmpts
print "Waveform average count: %d" % avgcnt
print "Waveform X increment: %1.12f" % x_increment
print "Waveform X origin: %1.9f" % x_origin
print "Waveform X reference: %d" % x_reference   # Always 0.
print "Waveform Y increment: %f" % y_increment
print "Waveform Y origin: %f" % y_origin
print "Waveform Y reference: %d" % y_reference   # Always 125.

# Get numeric values for later calculations.
x_increment = do_query_number(":WAVeform:XINCrement?")
x_origin = do_query_number(":WAVeform:XORigin?")
y_increment = do_query_number(":WAVeform:YINCrement?")
y_origin = do_query_number(":WAVeform:YORigin?")
y_reference = do_query_number(":WAVeform:YREFerence?")

# Get the waveform data.
data_bytes = do_query_ieee_block(":WAVeform:DATA?")
nLength = len(data_bytes)
print "Number of data values: %d" % nLength

# Open file for output.
strPath = "c:\scope\data\waveform_data.csv"
f = open(strPath, "w")

# Output waveform data in CSV format.
for i in xrange(0, nLength - 1):
 time_val = x_origin + (i * x_increment)
 voltage = (data_bytes[i] - y_reference) * y_increment + y_origin
 f.write("%E, %f\n" % (time_val, voltage))

# Close output file.
f.close()
print "Waveform format BYTE 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, 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(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 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?", True)
  error_string = myScope.ReadString()
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -1:   # Not "No error".
    print "ERROR: %s, command: '%s'" % (error_string, command)
    print "Exited because of error."
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? should always return string.
   print "ERROR: :SYSTem:ERRor? returned nothing, command: '%s'" \
     % command
   print "Exited because of error."
   sys.exit(1)


# =======================================================
# Main program:
# =======================================================
rm = CreateObject("VISA.GlobalRM", \
 interface=VisaComLib.IResourceManager)
myScope = CreateObject("VISA.BasicFormattedIO", \
 interface=VisaComLib.IFormattedIO488)
myScope.IO = \
 rm.Open("TCPIP0::10.112.94.136::hislip9-0.0::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()

print "End of program"
```

VISA Examples

VISA Example in C

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
6. Edit the program to use the VISA address of your oscilloscope.
7. 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. Click **OK** to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, under Projects and Solutions, select **VC++ Directories**.
   c. Show directories for **Include files**, and add the include directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\Include).
   d. Show directories for **Library files**, and add the library files directory (for example, Program Files (x86)\IVI Foundation\VISA\WinNT\lib\msc).
   e. Click **OK** to close the Options dialog.
9. Build and run the program.

```
/*
 * Keysight VISA Example in C
 * ------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <visa.h>             /* Keysight VISA routines. */

#define VISA_ADDRESS "TCPIP0::10.112.94.136::hislip9-0.0::INSTR"
#define IEEEBLOCK_SPACE 5000000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);     /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE block. */
void check_instrument_errors();        /* Check for inst errors. */
void error_handler();                  /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi;        /* Device session ID. */
ViStatus err;                   /* VISA function return value. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ---------------------------------------------------------------- */
void main(void)
{
  /* Open the default resource manager session. */
  err = viOpenDefaultRM(&defaultRM);
  if (err != VI_SUCCESS) error_handler();

  /* Open the session using the oscilloscope's VISA address. */
  err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
  if (err != VI_SUCCESS) error_handler();

/* Set the I/O timeout to fifteen seconds. */
err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
  if (err != VI_SUCCESS) error_handler();

  /* Initialize - start from a known state. */
  initialize();
```

```
  /* Capture data. */
  capture();

  /* Analyze the captured waveform. */
  analyze();

  /* Close the vi session and the resource manager session. */
  viClose(vi);
  viClose(defaultRM);
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------ */
void initialize (void)
{
  /* Clear the interface. */
  err = viClear(vi);
  if (err != VI_SUCCESS) error_handler();

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Clear status and load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------ */
void capture (void)
{
  int num_bytes;
  FILE *fp;

  /* Use auto-scale to automatically configure oscilloscope. */
  do_command(":AUToscale");

  /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. */
  do_command(":TRIGger:MODE EDGE");
  do_query_string(":TRIGger:MODE?");
  printf("Trigger mode: %s\n", str_result);

  /* Set EDGE trigger parameters. */
  do_command(":TRIGger:EDGE:SOURce CHANnel1");
  do_query_string(":TRIGger:EDGE:SOURce?");
  printf("Trigger edge source: %s\n", str_result);

  do_command(":TRIGger:EDGE:LEVel 1.5");
  do_query_string(":TRIGger:EDGE:LEVel?");
  printf("Trigger edge level: %s\n", str_result);

  do_command(":TRIGger:EDGE:SLOPe POSitive");
  do_query_string(":TRIGger:EDGE:SLOPe?");
  printf("Trigger edge slope: %s\n", str_result);

  /* Save oscilloscope configuration. */
```

```
  /* Read system setup. */
  num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
  printf("Read setup string query (%d bytes).\n", num_bytes);

  /* Write setup string to file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
  num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
  fclose (fp);
  printf("Wrote setup string (%d bytes) to ", num_bytes);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Change settings with individual commands:

  /* Set vertical scale and offset. */
  do_command(":CHANnel1:SCALe 0.05");
  do_query_string(":CHANnel1:SCALe?");
  printf("Channel 1 vertical scale: %s\n", str_result);

  do_command(":CHANnel1:OFFSet -1.5");
  do_query_string(":CHANnel1:OFFSet?");
  printf("Channel 1 offset: %s\n", str_result);

  /* Set horizontal scale and offset. */
  do_command(":TIMebase:SCALe 0.0002");
  do_query_string(":TIMebase:SCALe?");
  printf("Timebase scale: %s\n", str_result);

  do_command(":TIMebase:POSition 0.0");
  do_query_string(":TIMebase:POSition?");
  printf("Timebase position: %s\n", str_result);

  /* Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution). *
/
  do_command(":ACQuire:TYPE NORMal");
  do_query_string(":ACQuire:TYPE?");
  printf("Acquire type: %s\n", str_result);

  /* Or, configure by loading a previously saved setup. */

  /* Read setup string from file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
  num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
  fclose (fp);
  printf("Read setup string (%d bytes) from file ", num_bytes);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Restore setup string. */
  num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
  printf("Restored setup string (%d bytes).\n", num_bytes);

  /* Capture an acquisition using :DIGitize. */
  do_command(":DIGitize CHANnel1");
}
```

```
/* Analyze the captured waveform.
 * ------------------------------------------------------------- */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double x_reference;
  double y_increment;
  double y_origin;
  double y_reference;

  FILE *fp;
  int num_bytes;   /* Number of bytes returned from instrument. */
  int i;

  /* Make a couple of measurements.
   * ------------------------------------------------------------- */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
  printf("Frequency: %.4f kHz\n", num_result / 1000);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  /* Download the screen image.
   * ------------------------------------------------------------- */
  do_command(":HARDcopy:INKSaver OFF");

  /* Read screen image. */
  num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLor");
  printf("Screen image bytes: %d\n", num_bytes);

  /* Write screen image bytes to file. */
  fp = fopen ("c:\\scope\\data\\screen.png", "wb");
  num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
  fclose (fp);
  printf("Wrote screen image (%d bytes) to ", num_bytes);
  printf("c:\\scope\\data\\screen.png.\n");

  /* Download waveform data.
   * ------------------------------------------------------------- */

  /* Set the waveform points mode. */
  do_command(":WAVeform:POINts:MODE RAW");
  do_query_string(":WAVeform:POINts:MODE?");
  printf("Waveform points mode: %s\n", str_result);
```

```
/* Get the number of waveform points available. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
  printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
  printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
  printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
  printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
  printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
  printf("Acquire type: AVERage\n");
}
else if (acq_type == 3.0)
{
  printf("Acquire type: HRESolution\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);
```

```
  x_origin = dbl_results[5];
  printf("Waveform X origin: %e\n", x_origin);

  x_reference = dbl_results[6];
  printf("Waveform X reference: %e\n", x_reference);

  y_increment = dbl_results[7];
  printf("Waveform Y increment: %e\n", y_increment);

  y_origin = dbl_results[8];
  printf("Waveform Y origin: %e\n", y_origin);

  y_reference = dbl_results[9];
  printf("Waveform Y reference: %e\n", y_reference);

  /* Read waveform data. */
  num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
  printf("Number of data values: %d\n", num_bytes);

  /* Open file for output. */
  fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

  /* Output waveform data in CSV format. */
  for (i = 0; i < num_bytes - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      (((float)ieeeblock_data[i] - y_reference) * y_increment)
      + y_origin);
  }

  /* Close output file. */
   fclose(fp);
   printf("Waveform format BYTE data written to ");
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];

  strcpy(message, command);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
```

```
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  err = viPrintf(vi, message, num_bytes);
  if (err != VI_SUCCESS) error_handler();

  err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------------ */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%t", str_result);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------------ */
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%lf", &num_result);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}
```

```
/* Query for numbers result.
 * ------------------------------------------------------------- */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%,10lf\n", dbl_results);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  data_length = IEEEBLOCK_SPACE;
  err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
  if (err != VI_SUCCESS) error_handler();

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
  if (err != VI_SUCCESS) error_handler();
  while(strncmp(str_err_val, "+0,No error", 3) != 0 )
```

```
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
    err = viFlush(vi, VI_READ_BUF);
    if (err != VI_SUCCESS) error_handler();
    err = viFlush(vi, VI_WRITE_BUF);
    if (err != VI_SUCCESS) error_handler();
  }
}

/* Handle VISA errors.
 * ------------------------------------------------------------ */
void error_handler()
{
  char err_msg[1024] = {0};

  viStatusDesc(vi, err, err_msg);
  printf("VISA Error: %s\n", err_msg);
  if (err < VI_SUCCESS)
  {
    exit(1);
  }
}
```

VISA Example in 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 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 retCount As Long
Public byteArray(ByteArraySize) As Byte
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::10.112.94.136::hislip9-0.0::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

  ' 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 the interface.
  err = viClear(vi)
  If Not (err = VI_SUCCESS) Then HandleVISAError vi

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

End Sub

'
' Capture the waveform.
' ---------------------------------------------------------------------

Private Sub Capture()

  ' Use auto-scale to automatically configure oscilloscope.
  ' -------------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURce CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:EDGE:LEVel 1.5"
  Debug.Print "Trigger edge level: " + _
      DoQueryString(":TRIGger:EDGE:LEVel?")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' -------------------------------------------------------------------
```

```
Dim 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 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and 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 type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' 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)

  ' Capture an acquisition using :DIGitize.
  ' -------------------------------------------------------------------
  DoCommand ":DIGitize CHANnel1"

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.
  ' -------------------------------------------------------------------
  DoCommand ":HARDcopy:INKSaver OFF"

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, COLor")
  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.
' ------------------------------------------------------------------

' Set the waveform points mode.
DoCommand ":WAVeform:POINts:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINts:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
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 lngYOrigin As Long
Dim lngYReference As Long
Dim strOutput As String

Dim lngNumNumbers As Long
lngNumNumbers = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
lngYOrigin = dblArray(8)
lngYReference = dblArray(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
  Debug.Print "Waveform format: ASCii"
```

```
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMal"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
  Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(lngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Number of data values: " + CStr(lngNumBytes)

' 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

For lngI = 0 To lngNumBytes - 1
  lngDataValue = CLng(byteArray(lngI))

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
      ", " + _
```

```
        FormatNumber(((lngDataValue - lngYReference) _
        * sngYIncrement) + lngYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format BYTE 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 Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String

  err = viVPrintf(vi, ":SYSTem:ERRor?" + vbLf, 0)   ' Query any errors.
  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?" + vbLf, 0)   ' Request error.
    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 2008:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Keysight's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. 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.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Keysight IO Libraries Suite 17.

```
/*
 * Keysight VISA Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
```

```
namespace InfiniiVision
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaInstrument("TCPIP0::10.112.94.136::hislip9-0.0::INSTR");
        myScope.SetTimeoutSeconds(10);

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();

      }
      catch (System.ApplicationException err)
      {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
      }
      catch (System.SystemException err)
      {
        Console.WriteLine("*** System Error Message : " + err.Message);
      }
      catch (System.Exception err)
      {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
      }
      finally
      {
        myScope.Close();
      }
    }

    /*
     * Initialize the oscilloscope to a known state.
     * ---------------------------------------------------------------
     */
    private static void Initialize()
    {
      StringBuilder strResults;

      // Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?");
      Console.WriteLine("*IDN? result is: {0}", strResults);

      // Clear status and load the default setup.
```

```
  myScope.DoCommand("*CLS");
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -------------------------------------------------------------------
 */
private static void Capture()
{
  // Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1");
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

  myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
  Console.WriteLine("Trigger edge level: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

  // Save oscilloscope configuration.
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Query and read setup string.
  nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?",
    out ResultsArray);

  // Write setup string to file.
  strPath = "c:\\scope\\config\\setup.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 0.05");
  Console.WriteLine("Channel 1 vertical scale: {0}",
      myScope.DoQueryString(":CHANnel1:SCALe?"));

  myScope.DoCommand(":CHANnel1:OFFSet -1.5");
  Console.WriteLine("Channel 1 vertical offset: {0}",
      myScope.DoQueryString(":CHANnel1:OFFSet?"));
```

```
        // Set horizontal scale and position.
        myScope.DoCommand(":TIMebase:SCALe 0.0002");
        Console.WriteLine("Timebase scale: {0}",
            myScope.DoQueryString(":TIMebase:SCALe?"));

        myScope.DoCommand(":TIMebase:POSition 0.0");
        Console.WriteLine("Timebase position: {0}",
            myScope.DoQueryString(":TIMebase:POSition?"));

        // Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution
).
        myScope.DoCommand(":ACQuire:TYPE NORMal");
        Console.WriteLine("Acquire type: {0}",
            myScope.DoQueryString(":ACQuire:TYPE?"));

        // Or, configure by loading a previously saved setup.
        byte[] DataArray;
        int nBytesWritten;

        // Read setup string from file.
        strPath = "c:\\scope\\config\\setup.stp";
        DataArray = File.ReadAllBytes(strPath);

        // Restore setup string.
        nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
          DataArray);
        Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

        // Capture an acquisition using :DIGitize.
        myScope.DoCommand(":DIGitize CHANnel1");
      }

      /*
       * Analyze the captured waveform.
       * -------------------------------------------------------------------
       */
      private static void Analyze()
      {
        byte[] ResultsArray;   // Results array.
        int nLength;   // Number of bytes returned from instrument.
        string strPath;

        // Make a couple of measurements.
        // -------------------------------------------------------------
        myScope.DoCommand(":MEASure:SOURce CHANnel1");
        Console.WriteLine("Measure source: {0}",
            myScope.DoQueryString(":MEASure:SOURce?"));

        double fResult;
        myScope.DoCommand(":MEASure:FREQuency");
        fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
        Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

        myScope.DoCommand(":MEASure:VAMPlitude");
        fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
        Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);
```

```
// Download the screen image.
// ------------------------------------------------------------
myScope.DoCommand(":HARDcopy:INKSaver OFF");

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"));

// Get the number of waveform points available.
myScope.DoCommand(":WAVeform:POINts 10240");
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}
```

```
double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERage");
}
else if (fType == 3.0)
{
  Console.WriteLine("Acquire type: HRESolution");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
    out ResultsArray);
Console.WriteLine("Number of data values: {0}", nLength);

// 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)ResultsArray[i] - fYreference) *
            fYincrement) + fYorigin);

      // Close output file.
      writer.Close();
      Console.WriteLine("Waveform format BYTE data written to {0}",
          strPath);
    }
  }

  class VisaInstrument
  {
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
      // Save VISA address in member variable.
      m_strVisaAddress = strVisaAddress;

      // Open the default VISA resource manager.
      OpenResourceManager();

      // Open a VISA resource session.
      OpenSession();

      // Clear the interface.
      int nViStatus;
      nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
      // Send the command.
      VisaSendCommandOrQuery(strCommand);

      // Check for inst errors.
      CheckInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
      byte[] DataArray)
    {
      // Send the command to the device.
      string strCommandAndLength;
      int nViStatus, nLength, nBytesWritten;

      nLength = DataArray.Length;
      strCommandAndLength = String.Format("{0} #8%08d",
        strCommand);

      // Write first part of command to formatted I/O write buffer.
      nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
```

```
    nLength);
  CheckVisaStatus(nViStatus);

  // Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
    out nBytesWritten);
  CheckVisaStatus(nViStatus);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);

  return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  StringBuilder strResults = new StringBuilder(1000);
  strResults = VisaGetResultString();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double fResults;
  fResults = VisaGetResultNumber();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double[] fResultsArray;
  fResultsArray = VisaGetResultNumbers();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);
```

```
  // Return string results.
  return fResultsArray;
}

public int DoQueryIEEEBlock(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(out ResultsArray);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
  // Send command or query to the device.
  string strWithNewline;
  strWithNewline = String.Format("{0}\n", strCommandOrQuery);
  int nViStatus;
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
  CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
  StringBuilder strResults = new StringBuilder(1000);

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
  CheckVisaStatus(nViStatus);

  return strResults;
}

private double VisaGetResultNumber()
{
  double fResults = 0;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
  CheckVisaStatus(nViStatus);

  return fResults;
}

private double[] VisaGetResultNumbers()
```

```
{
  double[] fResultsArray;
  fResultsArray = new double[10];

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
      fResultsArray);
  CheckVisaStatus(nViStatus);

  return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new byte[300000];
  int length;   // Number of bytes returned from instrument.

  // Set the default number of bytes that will be contained in
  // the ResultsArray to 300,000 (300kB).
  length = 300000;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
    ResultsArray);
  CheckVisaStatus(nViStatus);

  // Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
  CheckVisaStatus(nViStatus);

  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
  CheckVisaStatus(nViStatus);

  return length;
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  StringBuilder strInstrumentError = new StringBuilder(1000);
  bool bFirstError = true;

  do   // While not "0,No error"
  {
    VisaSendCommandOrQuery(":SYSTem:ERRor?");
    strInstrumentError = VisaGetResultString();

    if (!strInstrumentError.ToString().StartsWith("+0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
```

```
            }
            Console.Write(strInstrumentError);
          }
        } while (!strInstrumentError.ToString().StartsWith("+0,"));
      }

      private void OpenResourceManager()
      {
        int nViStatus;
        nViStatus =
          visa32.viOpenDefaultRM(out this.m_nResourceManager);
        if (nViStatus < visa32.VI_SUCCESS)
          throw new
            ApplicationException("Failed to open Resource Manager");
      }

      private void OpenSession()
      {
        int nViStatus;
        nViStatus = visa32.viOpen(this.m_nResourceManager,
          this.m_strVisaAddress, visa32.VI_NO_LOCK,
          visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
        CheckVisaStatus(nViStatus);
      }

      public void SetTimeoutSeconds(int nSeconds)
      {
        int nViStatus;
        nViStatus = visa32.viSetAttribute(this.m_nSession,
          visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
        CheckVisaStatus(nViStatus);
      }

      public void CheckVisaStatus(int nViStatus)
      {
        // If VISA error, throw exception.
        if (nViStatus < visa32.VI_SUCCESS)
        {
          StringBuilder strError = new StringBuilder(256);
          visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
            strError);
          throw new ApplicationException(strError.ToString());
        }
      }

      public void Close()
      {
        if (m_nSession != 0)
          visa32.viClose(m_nSession);
        if (m_nResourceManager != 0)
          visa32.viClose(m_nResourceManager);
      }
    }
  }
```

VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the Visual Basic .NET source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Keysight's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add** 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 "InfiniiVision.VisaInstrumentApp" as the **Startup object**.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Keysight IO Libraries Suite 17.

```
'
' Keysight VISA Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Keysight oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
          New VisaInstrument("TCPIP0::10.112.94.136::hislip9-0.0::INSTR"
)
        myScope.SetTimeoutSeconds(10)
```

```
    ' Initialize - start from a known state.
    Initialize()

    ' Capture data.
    Capture()

    ' Analyze the captured waveform.
    Analyze()

  Catch err As System.ApplicationException
    Console.WriteLine("*** VISA Error Message : " + err.Message)
  Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
  Catch err As System.Exception
    Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
  End Try
End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------

Private Shared Sub Initialize()
  Dim strResults As StringBuilder

  ' Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?")
  Console.WriteLine("*IDN? result is: {0}", strResults)

  ' Clear status and load the default setup.
  myScope.DoCommand("*CLS")
  myScope.DoCommand("*RST")

End Sub

'
' Capture the waveform.
' ---------------------------------------------------------------

Private Shared Sub Capture()

  ' Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale")

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE")
  Console.WriteLine("Trigger mode: {0}", _
      myScope.DoQueryString(":TRIGger:MODE?"))

  ' Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURce CHANnel1")
  Console.WriteLine("Trigger edge source: {0}", _
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

  myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
  Console.WriteLine("Trigger edge level: {0}", _
```

```
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()   ' Results array.
Dim nLength As Integer   ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?", _
    ResultsArray)

' Write setup string to file.
strPath = "c:\scope\config\setup.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 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and position.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMebase:SCALe?"))

myScope.DoCommand(":TIMebase:POSition 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMebase:POSition?"))

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution)
.
myScope.DoCommand(":ACQuire:TYPE NORMal")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
DataArray = File.ReadAllBytes(strPath)
```

```
  ' Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
      DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Capture an acquisition using :DIGitize.
  myScope.DoCommand(":DIGitize CHANnel1")

End Sub

'
' Analyze the captured waveform.
' ---------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()   ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make a couple of measurements.
  ' ---------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}", _
      myScope.DoQueryString(":MEASure:SOURce?"))

  myScope.DoCommand(":MEASure:FREQuency")
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  myScope.DoCommand(":MEASure:VAMPlitude")
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

  ' Download the screen image.
  ' ---------------------------------------------------------------
  myScope.DoCommand(":HARDcopy:INKSaver OFF")

  ' Get the screen data.
  nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor", _
      ResultsArray)

  ' Store the screen data to a file.
  strPath = "c:\scope\data\screen.png"
  Dim fStream As FileStream
  fStream = File.Open(strPath, FileMode.Create)
  fStream.Write(ResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Screen image ({0} bytes) written to {1}", _
      nLength, strPath)

  ' Download waveform data.
  ' ---------------------------------------------------------------

  ' Set the waveform points mode.
  myScope.DoCommand(":WAVeform:POINts:MODE RAW")
```

```
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVeform:POINts:MODE?"))

' Get the number of waveform points available.
myScope.DoCommand(":WAVeform:POINts 10240")
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERage")
ElseIf fType = 3 Then
  Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)
```

```
    Dim fYincrement As Double = fResultsArray(7)
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

    Dim fYorigin As Double = fResultsArray(8)
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

    Dim fYreference As Double = fResultsArray(9)
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

    ' Get the waveform data.
    nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?", _
        ResultsArray)
    Console.WriteLine("Number of data values: {0}", nLength)

    ' 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(ResultsArray(index)) - fYreference) _
          * fYincrement) + fYorigin)
    Next

    ' Close output file.
    writer.Close()
    Console.WriteLine("Waveform format BYTE data written to {0}", _
        strPath)

  End Sub

End Class

Class VisaInstrument
  Private m_nResourceManager As Integer
  Private m_nSession As Integer
  Private m_strVisaAddress As String

  ' Constructor.
  Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA resource manager.
    OpenResourceManager()

    ' Open a VISA resource session.
    OpenSession()
```

```
  ' Clear the interface.
  Dim nViStatus As Integer
  nViStatus = visa32.viClear(m_nSession)
End Sub

Public Sub DoCommand(ByVal strCommand As String)
  ' Send the command.
  VisaSendCommandOrQuery(strCommand)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte()) As Integer

  ' Send the command to the device.
  Dim strCommandAndLength As String
  Dim nViStatus As Integer
  Dim nLength As Integer
  Dim nBytesWritten As Integer

  nLength = DataArray.Length
  strCommandAndLength = [String].Format("{0} #8{1:D8}", _
      strCommand, nLength)

  ' Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
  CheckVisaStatus(nViStatus)

  ' Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength, _
      nBytesWritten)
  CheckVisaStatus(nViStatus)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

  Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
  As StringBuilder
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim strResults As New StringBuilder(1000)
  strResults = VisaGetResultString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return strResults
End Function
```

```
Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResults As Double
  fResults = VisaGetResultNumber()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResultsArray As Double()
  fResultsArray = VisaGetResultNumbers()

  ' Check for instrument errors (another command and result).
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
    ByRef ResultsArray As Byte()) As Integer
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim length As Integer
  ' Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(ResultsArray)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return length
End Function

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
     As String)
  ' Send command or query to the device.
  Dim strWithNewline As String
  strWithNewline = [String].Format("{0}" & Chr(10) & "", _
      strCommandOrQuery)
  Dim nViStatus As Integer
```

```
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
  CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
  Dim strResults As New StringBuilder(1000)

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
  CheckVisaStatus(nViStatus)

  Return strResults
End Function

Private Function VisaGetResultNumber() As Double
  Dim fResults As Double = 0

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
  CheckVisaStatus(nViStatus)

  Return fResults
End Function

Private Function VisaGetResultNumbers() As Double()
  Dim fResultsArray As Double()
  fResultsArray = New Double(9) {}

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, _
      "%,10lf" & Chr(10) & "", fResultsArray)
  CheckVisaStatus(nViStatus)

  Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
    As Byte()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Byte(299999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
  ' the ResultsArray to 300,000 (300kB).
  length = 300000

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
      ResultsArray)
  CheckVisaStatus(nViStatus)

  ' Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
```

```
  CheckVisaStatus(nViStatus)

  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
  CheckVisaStatus(nViStatus)

  Return length
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As New StringBuilder(1000)
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error"
    VisaSendCommandOrQuery(":SYSTem:ERRor?")
    strInstrumentError = VisaGetResultString()

    If Not strInstrumentError.ToString().StartsWith("+0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ", _
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenResourceManager()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
  If nViStatus < visa32.VI_SUCCESS Then
    Throw New  _
        ApplicationException("Failed to open Resource Manager")
  End If
End Sub

Private Sub OpenSession()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpen(Me.m_nResourceManager, _
      Me.m_strVisaAddress, visa32.VI_NO_LOCK, _
      visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
  CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
  Dim nViStatus As Integer
  nViStatus = visa32.viSetAttribute(Me.m_nSession, _
      visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
  CheckVisaStatus(nViStatus)
End Sub

Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
  ' If VISA error, throw exception.
  If nViStatus < visa32.VI_SUCCESS Then
    Dim strError As New StringBuilder(256)
    visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
    Throw New ApplicationException(strError.ToString())
```

```
      End If
    End Sub

    Public Sub Close()
      If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
      End If
      If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
      End If
    End Sub
  End Class
End Namespace
```

VISA Example in Python (PyVISA 1.5 and older)

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.sourceforge.net/, 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
```

```
# ********************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight oscilloscope.
# ********************************************************

# Import modules.
# --------------------------------------------------------
import visa
import string
import struct
import sys

# Global variables (booleans: 0 = False, 1 = True).
# --------------------------------------------------------
debug = 0


# ========================================================
# 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():

 # 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:EDGE:LEVel 1.5")
 qresult = do_query_string(":TRIGger:EDGE:LEVel?")
 print "Trigger edge level: %s" % qresult

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 sSetup = do_query_string(":SYSTem:SETup?")
 sSetup = get_definite_length_block_data(sSetup)

 f = open("setup.stp", "wb")
 f.write(sSetup)
 f.close()
 print "Setup bytes saved: %d" % len(sSetup)

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(":CHANnel1:SCALe 0.05")
 qresult = do_query_values(":CHANnel1:SCALe?")[0]
 print "Channel 1 vertical scale: %f" % qresult

 do_command(":CHANnel1:OFFSet -1.5")
 qresult = do_query_values(":CHANnel1:OFFSet?")[0]
 print "Channel 1 offset: %f" % qresult

 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 0.0002")
```

```
 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 type.
 do_command(":ACQuire:TYPE NORMal")
 qresult = do_query_string(":ACQuire:TYPE?")
 print "Acquire type: %s" % qresult

 # Or, set up oscilloscope by loading a previously saved setup.
 sSetup = ""
 f = open("setup.stp", "rb")
 sSetup = f.read()
 f.close()
 do_command(":SYSTem:SETup #8%08d%s" % (len(sSetup), sSetup), hide_param
s=True)
 print "Setup bytes restored: %d" % len(sSetup)

 # Capture an acquisition using :DIGitize.
 do_command(":DIGitize CHANnel1")


# ==========================================================
# 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.
 # ----------------------------------------------------------
 do_command(":HARDcopy:INKSaver OFF")

 sDisplay = do_query_string(":DISPlay:DATA? PNG, COLor")
 sDisplay = get_definite_length_block_data(sDisplay)

 # Save display data values to file.
 f = open("screen_image.png", "wb")
 f.write(sDisplay)
 f.close()
 print "Screen image written to screen_image.png."
```

```
# Download waveform data.
# ------------------------------------------------------

# Set the waveform points mode.
do_command(":WAVeform:POINts:MODE RAW")
qresult = do_query_string(":WAVeform:POINts:MODE?")
print "Waveform points mode: %s" % qresult

# Get the number of waveform points available.
do_command(":WAVeform:POINts 10240")
qresult = do_query_string(":WAVeform:POINts?")
print "Waveform points available: %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 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
 3 : "HRESolution",
}

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
) = string.split(preamble_string, ",")

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

# Get numeric values for later calculations.
x_increment = do_query_values(":WAVeform:XINCrement?")[0]
x_origin = do_query_values(":WAVeform:XORigin?")[0]
y_increment = do_query_values(":WAVeform:YINCrement?")[0]
y_origin = do_query_values(":WAVeform:YORigin?")[0]
```

```
 y_reference = do_query_values(":WAVeform:YREFerence?")[0]

 # Get the waveform data.
 sData = do_query_string(":WAVeform:DATA?")
 sData = get_definite_length_block_data(sData)

 # Unpack unsigned 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 xrange(0, len(values) - 1):
  time_val = x_origin + (i * x_increment)
  voltage = ((values[i] - y_reference) * 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) = string.split(command, " ", 1)
  if debug:
   print "\nCmd = '%s'" % header
 else:
  if debug:
   print "\nCmd = '%s'" % command

 InfiniiVision.write("%s\n" % command)

 if hide_params:
  check_instrument_errors(header)
 else:
  check_instrument_errors(command)


# ======================================================
# Send a query, check for errors, return string:
# ======================================================
def do_query_string(query):
 if debug:
  print "Qys = '%s'" % query
 result = InfiniiVision.ask("%s\n" % query)
 check_instrument_errors(query)
 return result


# ======================================================
# Send a query, check for errors, return values:
# ======================================================
```

```
def do_query_values(query):
 if debug:
  print "Qyv = '%s'" % query
 results = InfiniiVision.ask_for_values("%s\n" % query)
 check_instrument_errors(query)
 return results


# ==========================================================
# Check for instrument errors:
# ==========================================================
def check_instrument_errors(command):

 while True:
  error_string = InfiniiVision.ask(":SYSTem:ERRor?\n")
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -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? should always return string.
   print "ERROR: :SYSTem:ERRor? returned nothing, command: '%s'" % comma
nd
   print "Exited because of error."
   sys.exit(1)


# ==========================================================
# Returns data from definite-length block.
# ==========================================================
def get_definite_length_block_data(sBlock):

 # First character should be "#".
 pound = sBlock[0:1]
 if pound != "#":
  print "PROBLEM: Invalid binary block format, pound char is '%s'." % po
und
  print "Exited because of problem."
  sys.exit(1)

 # Second character is number of following digits for length value.
 digits = sBlock[1:2]

 # Get the data out of the block and return it.
 sData = sBlock[int(digits) + 2:]

 return sData


# ==========================================================
# Main program:
```

```
# ========================================================

InfiniiVision = visa.instrument("TCPIP0::10.112.94.136::hislip9-0.0::INS
TR")
InfiniiVision.timeout = 15
InfiniiVision.term_chars = ""
InfiniiVision.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."
```

VISA Example in Python (PyVISA 1.6 and newer)

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
   ```

```
# ********************************************************
# This program illustrates a few commonly-used programming
# features of your Keysight oscilloscope.
# ********************************************************

# Import modules.
# --------------------------------------------------------
import visa
import string
import struct
import sys

# Global variables (booleans: 0 = False, 1 = True).
# --------------------------------------------------------
debug = 0


# ========================================================
# 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():

 # 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:EDGE:LEVel 1.5")
 qresult = do_query_string(":TRIGger:EDGE:LEVel?")
 print "Trigger edge level: %s" % qresult

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 sSetup = do_query_ieee_block(":SYSTem:SETup?")

 f = open("setup.stp", "wb")
 f.write(sSetup)
 f.close()
 print "Setup bytes saved: %d" % len(sSetup)

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(":CHANnel1:SCALe 0.05")
 qresult = do_query_string(":CHANnel1:SCALe?")
 print "Channel 1 vertical scale: %s" % qresult

 do_command(":CHANnel1:OFFSet -1.5")
 qresult = do_query_string(":CHANnel1:OFFSet?")
 print "Channel 1 offset: %s" % qresult

 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 0.0002")
```

```
 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 type.
 do_command(":ACQuire:TYPE NORMal")
 qresult = do_query_string(":ACQuire:TYPE?")
 print "Acquire type: %s" % qresult

 # Or, set up oscilloscope by loading a previously saved setup.
 sSetup = ""
 f = open("setup.stp", "rb")
 sSetup = f.read()
 f.close()
 do_command_ieee_block(":SYSTem:SETup", sSetup)
 print "Setup bytes restored: %d" % len(sSetup)

 # Capture an acquisition using :DIGitize.
 do_command(":DIGitize CHANnel1")


# ==========================================================
# 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.
 # ----------------------------------------------------------
 do_command(":HARDcopy:INKSaver OFF")

 sDisplay = do_query_ieee_block(":DISPlay:DATA? PNG, COLor")

 # Save display data values to file.
 f = open("screen_image.png", "wb")
 f.write(sDisplay)
 f.close()
 print "Screen image written to screen_image.png."

 # Download waveform data.
 # ----------------------------------------------------------
```

```
# Set the waveform points mode.
do_command(":WAVeform:POINts:MODE RAW")
qresult = do_query_string(":WAVeform:POINts:MODE?")
print "Waveform points mode: %s" % qresult

# Get the number of waveform points available.
do_command(":WAVeform:POINts 10240")
qresult = do_query_string(":WAVeform:POINts?")
print "Waveform points available: %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 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
 3 : "HRESolution",
}

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
) = string.split(preamble_string, ",")

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

# Get numeric values for later calculations.
x_increment = do_query_number(":WAVeform:XINCrement?")
x_origin = do_query_number(":WAVeform:XORigin?")
y_increment = do_query_number(":WAVeform:YINCrement?")
y_origin = do_query_number(":WAVeform:YORigin?")
y_reference = do_query_number(":WAVeform:YREFerence?")
```

```
 # Get the waveform data.
 sData = do_query_ieee_block(":WAVeform:DATA?")

 # Unpack unsigned 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 xrange(0, len(values) - 1):
  time_val = x_origin + (i * x_increment)
  voltage = ((values[i] - y_reference) * 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) = string.split(command, " ", 1)
  if debug:
   print "\nCmd = '%s'" % header
 else:
  if debug:
   print "\nCmd = '%s'" % command

 InfiniiVision.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
 InfiniiVision.write_binary_values("%s " % command, values, datatype='c'
)
 check_instrument_errors(command)


# ========================================================
# Send a query, check for errors, return string:
# ========================================================
def do_query_string(query):
 if debug:
  print "Qys = '%s'" % query
```

```
 result = InfiniiVision.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 = InfiniiVision.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 "Qys = '%s'" % query
 result = InfiniiVision.query_binary_values("%s" % query, datatype='s')
 check_instrument_errors(query)
 return result[0]


# ============================================================
# Check for instrument errors:
# ============================================================
def check_instrument_errors(command):

 while True:
  error_string = InfiniiVision.query(":SYSTem:ERRor?")
  if error_string:   # If there is an error string value.

   if error_string.find("+0,", 0, 3) == -1:   # Not "No error".

    print "ERROR: %s, command: '%s'" % (error_string, command)
    print "Exited because of error."
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? should always return string.
   print "ERROR: :SYSTem:ERRor? returned nothing, command: '%s'" % comma
nd
   print "Exited because of error."
   sys.exit(1)


# ============================================================
# Main program:
# ============================================================

rm = visa.ResourceManager("C:\\Windows\\System32\\agvisa32.dll")
```

```
InfiniiVision= rm.open_resource("TCPIP0::10.112.94.136::hislip9-0.0::INS
TR")

InfiniiVision.timeout = 15000
InfiniiVision.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."
```

SICL Examples

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
6. Edit the program to use the SICL address of your oscilloscope.
7. 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. Click **OK** to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, select **VC++ Directories** under Projects and Solutions.
   c. Show directories for **Include files**, and add the include directory (for example, Program Files\Agilent\IO Libraries Suite\include).
   d. Show directories for **Library files**, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
   e. Click **OK** to close the Options dialog.
9. Build and run the program.

```
/*
 * Keysight SICL Example in C
 * ------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Keysight oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
```

```
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <sicl.h>             /* Keysight SICL routines. */

#define SICL_ADDRESS      "usb0[2391::6054::US50210029::0]"
#define TIMEOUT           5000
#define IEEEBLOCK_SPACE   300000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);    /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE block. */
void check_instrument_errors();        /* Check for inst errors. */

/* Global variables */
INST id;                        /* Device session ID. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------- */
void main(void)
{
  /* Install a default SICL error handler that logs an error message
   * and exits.  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");
  }

  /* Initialize - start from a known state. */
  initialize();

  /* Capture data. */
  capture();
```

```
  /* Analyze the captured waveform. */
  analyze();

  /* Close the device session to the instrument. */
  iclose(id);
  printf ("Program execution is complete...\n");

  /* For WIN16 programs, call _siclcleanup before exiting to release
   * resources allocated by SICL for this application.  This call is
   * a no-op for WIN32 programs.
   */
  _siclcleanup();
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize (void)
{
  /* Set the I/O timeout value for this session to 5 seconds. */
  itimeout(id, TIMEOUT);

  /* Clear the interface. */
  iclear(id);

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Clear status and load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture (void)
{
  int num_bytes;
  FILE *fp;

  /* Use auto-scale to automatically configure oscilloscope.
   * ------------------------------------------------------------ */
  do_command(":AUToscale");

  /* Set trigger mode (EDGE, PULSe, PATTern, etc., and input source. */
  do_command(":TRIGger:MODE EDGE");
  do_query_string(":TRIGger:MODE?");
  printf("Trigger mode: %s\n", str_result);

  /* Set EDGE trigger parameters. */
  do_command(":TRIGger:EDGE:SOURce CHANnel1");
  do_query_string(":TRIGger:EDGE:SOURce?");
  printf("Trigger edge source: %s\n", str_result);

  do_command(":TRIGger:EDGE:LEVel 1.5");
  do_query_string(":TRIGger:EDGE:LEVel?");
```

```
  printf("Trigger edge level: %s\n", str_result);

  do_command(":TRIGger:EDGE:SLOPe POSitive");
  do_query_string(":TRIGger:EDGE:SLOPe?");
  printf("Trigger edge slope: %s\n", str_result);

  /* Save oscilloscope configuration.
   * ------------------------------------------------------------ */

  /* Read system setup. */
  num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
  printf("Read setup string query (%d bytes).\n", num_bytes);

  /* Write setup string to file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
  num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
  fclose (fp);
  printf("Wrote setup string (%d bytes) to ", num_bytes);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Change settings with individual commands:
   * ------------------------------------------------------------ */

  /* Set vertical scale and offset. */
  do_command(":CHANnel1:SCALe 0.05");
  do_query_string(":CHANnel1:SCALe?");
  printf("Channel 1 vertical scale: %s\n", str_result);

  do_command(":CHANnel1:OFFSet -1.5");
  do_query_string(":CHANnel1:OFFSet?");
  printf("Channel 1 offset: %s\n", str_result);

  /* Set horizontal scale and position. */
  do_command(":TIMebase:SCALe 0.0002");
  do_query_string(":TIMebase:SCALe?");
  printf("Timebase scale: %s\n", str_result);

  do_command(":TIMebase:POSition 0.0");
  do_query_string(":TIMebase:POSition?");
  printf("Timebase position: %s\n", str_result);

  /* Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution). *
/
  do_command(":ACQuire:TYPE NORMal");
  do_query_string(":ACQuire:TYPE?");
  printf("Acquire type: %s\n", str_result);

  /* Or, configure by loading a previously saved setup.
  * ------------------------------------------------------------ */

  /* Read setup string from file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
  num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
  fclose (fp);
  printf("Read setup string (%d bytes) from file ", num_bytes);
```

```
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Restore setup string. */
  num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
  printf("Restored setup string (%d bytes).\n", num_bytes);

  /* Capture an acquisition using :DIGitize.
   * ------------------------------------------------------------- */
  do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------- */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double x_reference;
  double y_increment;
  double y_origin;
  double y_reference;

  FILE *fp;
  int num_bytes;   /* Number of bytes returned from instrument. */
  int i;

  /* Make a couple of measurements.
   * ------------------------------------------------------------- */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
  printf("Frequency: %.4f kHz\n", num_result / 1000);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  /* Download the screen image.
   * ------------------------------------------------------------- */
  do_command(":HARDcopy:INKSaver OFF");

  /* Read screen image. */
  num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLor");
  printf("Screen image bytes: %d\n", num_bytes);

  /* Write screen image bytes to file. */
  fp = fopen ("c:\\scope\\data\\screen.png", "wb");
  num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
  fclose (fp);
```

```
printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * ------------------------------------------------------------- */

/* Set the waveform points mode. */
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_command(":WAVeform:POINts 10240");
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
  printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
  printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
  printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
  printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
  printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
  printf("Acquire type: AVERage\n");
}
else if (acq_type == 3.0)
```

```
  {
    printf("Acquire type: HRESolution\n");
  }

  wav_points = dbl_results[2];
  printf("Waveform points: %e\n", wav_points);

  avg_count = dbl_results[3];
  printf("Waveform average count: %e\n", avg_count);

  x_increment = dbl_results[4];
  printf("Waveform X increment: %e\n", x_increment);

  x_origin = dbl_results[5];
  printf("Waveform X origin: %e\n", x_origin);

  x_reference = dbl_results[6];
  printf("Waveform X reference: %e\n", x_reference);

  y_increment = dbl_results[7];
  printf("Waveform Y increment: %e\n", y_increment);

  y_origin = dbl_results[8];
  printf("Waveform Y origin: %e\n", y_origin);

  y_reference = dbl_results[9];
  printf("Waveform Y reference: %e\n", y_reference);

  /* Read waveform data. */
  num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
  printf("Number of data values: %d\n", num_bytes);

  /* Open file for output. */
  fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

  /* Output waveform data in CSV format. */
  for (i = 0; i < num_bytes - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      (((float)ieeeblock_data[i] - y_reference) * y_increment)
      + y_origin);
  }

  /* Close output file. */
   fclose(fp);
   printf("Waveform format BYTE data written to ");
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];
```

```
  strcpy(message, command);
  strcat(message, "\n");
  iprintf(id, message);

  check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ------------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  iprintf(id, message, num_bytes);
  ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------------- */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%t\n", str_result);

  check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------------- */
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%lf", &num_result);

  check_instrument_errors();
```

```
}

/* Query for numbers result.
 * ------------------------------------------------------------- */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%,10lf\n", dbl_results);

  check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  data_length = IEEEBLOCK_SPACE;
  iscanf(id, "%#b", &data_length, ieeeblock_data);

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
  while(strncmp(str_err_val, "+0,No error", 3) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
```

```
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
    iflush(id, I_BUF_READ | I_BUF_WRITE);
  }
}
```

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 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("usb0[2391::6054::US50210029::0]")
  Call itimeout(id, 5000)

  ' 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 the interface.
  Call iclear(id)

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

  Exit Sub

ErrorHandler:
```

```
  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Capture the waveform.
' ----------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo ErrorHandler

  ' Use auto-scale to automatically configure oscilloscope.
  ' ----------------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURce CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:EDGE:LEVel 1.5"
  Debug.Print "Trigger edge level: " + _
      DoQueryString(":TRIGger:EDGE:LEVel?")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' ----------------------------------------------------------------------
  Dim 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 0.05"
  Debug.Print "Channel 1 vertical scale: " + _
      DoQueryString(":CHANnel1:SCALe?")

  DoCommand ":CHANnel1:OFFSet -1.5"
  Debug.Print "Channel 1 vertical offset: " + _
      DoQueryString(":CHANnel1:OFFSet?")

  ' Set horizontal scale and 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 type (NORMal, PEAK, AVERage, or HRESolution).
  DoCommand ":ACQuire:TYPE NORMal"
  Debug.Print "Acquire type: " + _
      DoQueryString(":ACQuire:TYPE?")

  ' 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)

  ' Capture an acquisition using :DIGitize.
  ' ------------------------------------------------------------------
  DoCommand ":DIGitize CHANnel1"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Analyze the captured waveform.
```

```
' ----------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo ErrorHandler

  ' 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.
  ' --------------------------------------------------------------------
  DoCommand ":HARDcopy:INKSaver OFF"

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, COLor")
  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.
  ' --------------------------------------------------------------------

  ' Set the waveform points mode.
  DoCommand ":WAVeform:POINts:MODE RAW"
  Debug.Print "Waveform points mode: " + _
      DoQueryString(":WAVeform:POINts:MODE?")

  ' Get the number of waveform points available.
```

```
DoCommand ":WAVeform:POINts 10240"
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim Preamble() As Double
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
  Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMal"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
  Debug.Print "Acquisition type: HRESolution"
End If
```

```
Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
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

' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngNumBytes - 2
  lngDataValue = CLng(byteArray(lngI))

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
      ", " + _
      FormatNumber(((lngDataValue - lngYReference) * _
      sngYIncrement) + sngYOrigin)

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.
  Sleep 2000   ' Delay before reading data.
  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?" + vbLf)   ' Query any errors data.
  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?" + vbLf)   ' Request error message
.
    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

These programming examples show how to use the SCPI.NET drivers that come with Keysight's free Command Expert software.

While you can write code manually using SCPI.NET drivers (as described in this section), 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.

For more information on Keysight Command Expert, and to download the software, see: http://www.keysight.com/find/commandexpert

- "SCPI.NET Example in C#" on page 1470
- "SCPI.NET Example in Visual Basic .NET" on page 1476
- "SCPI.NET Example in IronPython" on page 1482

SCPI.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Install the Keysight Command Expert software and the command set for the oscilloscope.
2. Open Visual Studio.
3. Create a new Visual C#, Windows, Console Application project.
4. Cut-and-paste the code that follows into the C# source file.
5. Edit the program to use the address of your oscilloscope.
6. Add a reference to the SCPI.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 Add Reference dialog, select the **Browse** tab, and navigate to the ScpiNetDrivers folder.

- Windows XP: C:\Documents and Settings\All Users\Keysight\Command Expert\ScpiNetDrivers
- Windows 7: C:\ProgramData\Keysight\Command Expert\ScpiNetDrivers

d Select the .dll file for your oscilloscope, for example **AgInfiniiVision3000X_02_00.dll**; then, click **OK**.

7 Build and run the program.

For more information, see the SCPI.NET driver help that comes with Keysight Command Expert.

```
/*
 * Keysight SCPI.NET Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Keysight oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Keysight.CommandExpert.ScpiNet.AgInfiniiVision3000X_02_00;

namespace InfiniiVision
{
  class ScpiNetInstrumentApp
  {
    private static AgInfiniiVision3000X myScope;

    static void Main(string[] args)
    {
      try
      {
        string strScopeAddress;
        strScopeAddress =
          "TCPIP0::10.112.94.136::hislip9-0.0::INSTR";
        Console.WriteLine("Connecting to oscilloscope...");
        Console.WriteLine();
        myScope = new AgInfiniiVision3000X(strScopeAddress);
        myScope.Transport.DefaultTimeout.Set(10000);

        // 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("*** SCPI.NET 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.Dispose();
  }

}

/*
 * Initialize the oscilloscope to a known state.
 * ---------------------------------------------------------------
 */
private static void Initialize()
{
  string strResults;

  // Get and display the device's *IDN? string.
  myScope.SCPI.IDN.Query(out strResults);
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.SCPI.CLS.Command();
  myScope.SCPI.RST.Command();
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------
 */
private static void Capture()
{
  string strResults;
  double fResult;

  // Use auto-scale to automatically configure oscilloscope.
  myScope.SCPI.AUToscale.Command(null, null, null, null, null);

  // Set trigger mode.
  myScope.SCPI.TRIGger.MODE.Command("EDGE");
  myScope.SCPI.TRIGger.MODE.Query(out strResults);
  Console.WriteLine("Trigger mode: {0}", strResults);

  // Set EDGE trigger parameters.
  myScope.SCPI.TRIGger.EDGE.SOURce.Command("CHANnel1");
  myScope.SCPI.TRIGger.EDGE.SOURce.Query(out strResults);
  Console.WriteLine("Trigger edge source: {0}", strResults);
```

```
myScope.SCPI.TRIGger.EDGE.LEVel.Command(1.5, "CHANnel1");
myScope.SCPI.TRIGger.EDGE.LEVel.Query("CHANnel1", out fResult);
Console.WriteLine("Trigger edge level: {0:F2}", fResult);

myScope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive");
myScope.SCPI.TRIGger.EDGE.SLOPe.Query(out strResults);
Console.WriteLine("Trigger edge slope: {0}", strResults);

// Save oscilloscope configuration.
string[] strResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
myScope.SCPI.SYSTem.SETup.Query(out strResultsArray);
nLength = strResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
File.WriteAllLines(strPath, strResultsArray);
Console.WriteLine("Setup bytes saved: {0}", nLength);


// Change settings with individual commands:

// Set vertical scale and offset.
myScope.SCPI.CHANnel.SCALe.Command(1, 0.05);
myScope.SCPI.CHANnel.SCALe.Query(1, out fResult);
Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult);

myScope.SCPI.CHANnel.OFFSet.Command(1, -1.5);
myScope.SCPI.CHANnel.OFFSet.Query(1, out fResult);
Console.WriteLine("Channel 1 vertical offset: {0:F4}", fResult);

// Set horizontal scale and offset.
myScope.SCPI.TIMebase.SCALe.Command(0.0002);
myScope.SCPI.TIMebase.SCALe.Query(out fResult);
Console.WriteLine("Timebase scale: {0:F4}", fResult);

myScope.SCPI.TIMebase.POSition.Command(0.0);
myScope.SCPI.TIMebase.POSition.Query(out fResult);
Console.WriteLine("Timebase position: {0:F2}", fResult);

// Set the acquisition type.
myScope.SCPI.ACQuire.TYPE.Command("NORMal");
myScope.SCPI.ACQuire.TYPE.Query(out strResults);
Console.WriteLine("Acquire type: {0}", strResults);

// Or, configure by loading a previously saved setup.
int nBytesWritten;

strPath = "c:\\scope\\config\\setup.stp";
strResultsArray = File.ReadAllLines(strPath);
nBytesWritten = strResultsArray.Length;

// Restore setup string.
```

```
  myScope.SCPI.SYSTem.SETup.Command(strResultsArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Capture an acquisition using :DIGitize.
  myScope.SCPI.DIGitize.Command("CHANnel1", null, null, null, null);
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------
 */
private static void Analyze()
{
  string strResults, source1, source2;
  double fResult;

  // Make a couple of measurements.
  // ------------------------------------------------------------
  myScope.SCPI.MEASure.SOURce.Command("CHANnel1", null);
  myScope.SCPI.MEASure.SOURce.Query(out source1, out source2);
  Console.WriteLine("Measure source: {0}", source1);

  myScope.SCPI.MEASure.FREQuency.Command("CHANnel1");
  myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", out fResult);
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  // Use direct command/query when commands not in command set.
  myScope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1");
  myScope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1",
    out strResults);
  Console.WriteLine("Vertical amplitude: {0} V", strResults);

  // Download the screen image.
  // ------------------------------------------------------------
  myScope.SCPI.HARDcopy.INKSaver.Command(false);

  // Get the screen data.
  byte[] byteResultsArray;   // Results array.
  myScope.SCPI.DISPlay.DATA.Query("PNG", "COLor",
    out byteResultsArray);
  int nLength;   // Number of bytes returned from instrument.
  nLength = byteResultsArray.Length;

  // Store the screen data to a file.
  string strPath;
  strPath = "c:\\scope\\data\\screen.png";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(byteResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Screen image ({0} bytes) written to {1}",
      nLength, strPath);

  // Download waveform data.
  // ------------------------------------------------------------

  // Set the waveform points mode.
  myScope.SCPI.WAVeform.POINts.MODE.Command("RAW");
```

```
myScope.SCPI.WAVeform.POINts.MODE.Query(out strResults);
Console.WriteLine("Waveform points mode: {0}", strResults);

// Get the number of waveform points available.
myScope.SCPI.WAVeform.POINts.CommandPoints(10240);
int nPointsAvail;
myScope.SCPI.WAVeform.POINts.Query1(out nPointsAvail);
Console.WriteLine("Waveform points available: {0}", nPointsAvail);

// Set the waveform source.
myScope.SCPI.WAVeform.SOURce.Command("CHANnel1");
myScope.SCPI.WAVeform.SOURce.Query(out strResults);
Console.WriteLine("Waveform source: {0}", strResults);

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.SCPI.WAVeform.FORMat.Command("BYTE");
myScope.SCPI.WAVeform.FORMat.Query(out strResults);
Console.WriteLine("Waveform format: {0}", strResults);

// Display the waveform settings:
int nFormat, nType, nPoints, nCount, nXreference, nYreference;
double dblXincrement, dblXorigin, dblYincrement, dblYorigin;
myScope.SCPI.WAVeform.PREamble.Query(
  out nFormat,
  out nType,
  out nPoints,
  out nCount,
  out dblXincrement,
  out dblXorigin,
  out nXreference,
  out dblYincrement,
  out dblYorigin,
  out nYreference);

if (nFormat == 0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (nFormat == 1)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (nFormat == 2)
{
  Console.WriteLine("Waveform format: ASCii");
}

if (nType == 0)
{
  Console.WriteLine("Acquire type: NORMal");
}
else if (nType == 1)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (nType == 2)
{
```

```
        Console.WriteLine("Acquire type: AVERage");
      }
      else if (nType == 3)
      {
        Console.WriteLine("Acquire type: HRESolution");
      }

      Console.WriteLine("Waveform points: {0:e}", nPoints);
      Console.WriteLine("Waveform average count: {0:e}", nCount);
      Console.WriteLine("Waveform X increment: {0:e}", dblXincrement);
      Console.WriteLine("Waveform X origin: {0:e}", dblXorigin);
      Console.WriteLine("Waveform X reference: {0:e}", nXreference);
      Console.WriteLine("Waveform Y increment: {0:e}", dblYincrement);
      Console.WriteLine("Waveform Y origin: {0:e}", dblYorigin);
      Console.WriteLine("Waveform Y reference: {0:e}", nYreference);

      // Read waveform data.
      myScope.SCPI.WAVeform.DATA.QueryBYTE(out byteResultsArray);
      nLength = byteResultsArray.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}",
            dblXorigin + ((float)i * dblXincrement),
            (((float)byteResultsArray[i] - nYreference)
            * dblYincrement) + dblYorigin);

      // Close output file.
      writer.Close();
      Console.WriteLine("Waveform format BYTE data written to {0}",
          strPath);
    }
  }
}
```

SCPI.NET Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Install the Keysight Command Expert software and the command set for the oscilloscope.
2. Open Visual Studio.
3. Create a new Visual Basic, Windows, 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 SCPI.NET 3.0 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 Add Reference dialog, select the **Browse** tab, and navigate to the ScpiNetDrivers folder.

      - Windows XP: C:\Documents and Settings\All Users\Keysight\Command Expert\ScpiNetDrivers
      - Windows 7: C:\ProgramData\Keysight\Command Expert\ScpiNetDrivers

   d Select the .dll file for your oscilloscope, for example **AgInfiniiVision3000X_02_00.dll**; then, click **OK**.

   e Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.ScpiNetInstrumentApp" as the **Startup object**.

7 Build and run the program.

For more information, see the SCPI.NET driver help that comes with Keysight Command Expert.

```
'
' Keysight SCPI.NET Example in Visual Basic .NET
' ------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Keysight oscilloscope.
' ------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Keysight.CommandExpert.ScpiNet.AgInfiniiVision3000X_02_00

Namespace InfiniiVision
  Class ScpiNetInstrumentApp
    Private Shared myScope As AgInfiniiVision3000X

    Public Shared Sub Main(ByVal args As String())
      Try
        Dim strScopeAddress As String
        strScopeAddress = _
          "TCPIP0::10.112.94.136::hislip9-0.0::INSTR"
        Console.WriteLine("Connecting to oscilloscope...")
        Console.WriteLine()
        myScope = New AgInfiniiVision3000X(strScopeAddress)
        myScope.Transport.DefaultTimeout.[Set](10000)

        ' 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 err As System.ApplicationException
    Console.WriteLine("*** SCPI.NET Error : " & 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)
    'myScope.Dispose();
  Finally
  End Try

End Sub

' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------

Private Shared Sub Initialize()
  Dim strResults As String

  ' Get and display the device's *IDN? string.
  myScope.SCPI.IDN.Query(strResults)
  Console.WriteLine("*IDN? result is: {0}", strResults)

  ' Clear status and load the default setup.
  myScope.SCPI.CLS.Command()
  myScope.SCPI.RST.Command()
End Sub

' Capture the waveform.
' ---------------------------------------------------------------

Private Shared Sub Capture()
  Dim strResults As String
  Dim fResult As Double

  ' Use auto-scale to automatically configure oscilloscope.
  myScope.SCPI.AUToscale.Command(Nothing, Nothing, Nothing, _
                                 Nothing, Nothing)

  ' Set trigger mode.
  myScope.SCPI.TRIGger.MODE.Command("EDGE")
  myScope.SCPI.TRIGger.MODE.Query(strResults)
  Console.WriteLine("Trigger mode: {0}", strResults)

  ' Set EDGE trigger parameters.
  myScope.SCPI.TRIGger.EDGE.SOURce.Command("CHANnel1")
  myScope.SCPI.TRIGger.EDGE.SOURce.Query(strResults)
  Console.WriteLine("Trigger edge source: {0}", strResults)

  myScope.SCPI.TRIGger.EDGE.LEVel.Command(1.5, "CHANnel1")
```

```
myScope.SCPI.TRIGger.EDGE.LEVel.Query("CHANnel1", fResult)
Console.WriteLine("Trigger edge level: {0:F2}", fResult)

myScope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive")
myScope.SCPI.TRIGger.EDGE.SLOPe.Query(strResults)
Console.WriteLine("Trigger edge slope: {0}", strResults)

' Save oscilloscope configuration.
Dim strResultsArray As String()
' Results array.
Dim nLength As Integer
' Number of bytes returned from instrument.
Dim strPath As String

' Query and read setup string.
myScope.SCPI.SYSTem.SETup.Query(strResultsArray)
nLength = strResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
File.WriteAllLines(strPath, strResultsArray)
Console.WriteLine("Setup bytes saved: {0}", nLength)


' Change settings with individual commands:

' Set vertical scale and offset.
myScope.SCPI.CHANnel.SCALe.Command(1, 0.05)
myScope.SCPI.CHANnel.SCALe.Query(1, fResult)
Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult)

myScope.SCPI.CHANnel.OFFSet.Command(1, -1.5)
myScope.SCPI.CHANnel.OFFSet.Query(1, fResult)
Console.WriteLine("Channel 1 vertical offset: {0:F4}", fResult)

' Set horizontal scale and offset.
myScope.SCPI.TIMebase.SCALe.Command(0.0002)
myScope.SCPI.TIMebase.SCALe.Query(fResult)
Console.WriteLine("Timebase scale: {0:F4}", fResult)

myScope.SCPI.TIMebase.POSition.Command(0.0)
myScope.SCPI.TIMebase.POSition.Query(fResult)
Console.WriteLine("Timebase position: {0:F2}", fResult)

' Set the acquisition type.
myScope.SCPI.ACQuire.TYPE.Command("NORMal")
myScope.SCPI.ACQuire.TYPE.Query(strResults)
Console.WriteLine("Acquire type: {0}", strResults)

' Or, configure by loading a previously saved setup.
Dim nBytesWritten As Integer

strPath = "c:\scope\config\setup.stp"
strResultsArray = File.ReadAllLines(strPath)
nBytesWritten = strResultsArray.Length

' Restore setup string.
```

```
  myScope.SCPI.SYSTem.SETup.Command(strResultsArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Capture an acquisition using :DIGitize.
  myScope.SCPI.DIGitize.Command("CHANnel1", Nothing, Nothing, _
                                Nothing, Nothing)
End Sub

' Analyze the captured waveform.
' ------------------------------------------------------------

Private Shared Sub Analyze()
  Dim strResults As String, source1 As String, source2 As String
  Dim fResult As Double

  ' Make a couple of measurements.
  ' ------------------------------------------------------------
  myScope.SCPI.MEASure.SOURce.Command("CHANnel1", Nothing)
  myScope.SCPI.MEASure.SOURce.Query(source1, source2)
  Console.WriteLine("Measure source: {0}", source1)

  myScope.SCPI.MEASure.FREQuency.Command("CHANnel1")
  myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", fResult)
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  ' Use direct command/query when commands not in command set.
  myScope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1")
  myScope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1", _
                                 strResults)
  Console.WriteLine("Vertical amplitude: {0} V", strResults)

  ' Download the screen image.
  ' ------------------------------------------------------------
  myScope.SCPI.HARDcopy.INKSaver.Command(False)

  ' Get the screen data.
  Dim byteResultsArray As Byte()
  ' Results array.
  myScope.SCPI.DISPlay.DATA.Query("PNG", "COLor", byteResultsArray)
  Dim nLength As Integer
  ' Number of bytes returned from instrument.
  nLength = byteResultsArray.Length

  ' Store the screen data to a file.
  Dim strPath As String
  strPath = "c:\scope\data\screen.png"
  Dim fStream As FileStream = File.Open(strPath, FileMode.Create)
  fStream.Write(byteResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Screen image ({0} bytes) written to {1}", _
                    nLength, strPath)

  ' Download waveform data.
  ' ------------------------------------------------------------

  ' Set the waveform points mode.
  myScope.SCPI.WAVeform.POINts.MODE.Command("RAW")
```

```
myScope.SCPI.WAVeform.POINts.MODE.Query(strResults)
Console.WriteLine("Waveform points mode: {0}", strResults)

' Get the number of waveform points available.
myScope.SCPI.WAVeform.POINts.CommandPoints(10240)
Dim nPointsAvail As Integer
myScope.SCPI.WAVeform.POINts.Query1(nPointsAvail)
Console.WriteLine("Waveform points available: {0}", nPointsAvail)

' Set the waveform source.
myScope.SCPI.WAVeform.SOURce.Command("CHANnel1")
myScope.SCPI.WAVeform.SOURce.Query(strResults)
Console.WriteLine("Waveform source: {0}", strResults)

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.SCPI.WAVeform.FORMat.Command("BYTE")
myScope.SCPI.WAVeform.FORMat.Query(strResults)
Console.WriteLine("Waveform format: {0}", strResults)

' Display the waveform settings:
Dim nFormat As Integer, nType As Integer, nPoints As Integer, _
  nCount As Integer, nXreference As Integer, _
  nYreference As Integer
Dim dblXincrement As Double, dblXorigin As Double, _
  dblYincrement As Double, dblYorigin As Double
myScope.SCPI.WAVeform.PREamble.Query(nFormat, nType, nPoints, _
  nCount, dblXincrement, dblXorigin, nXreference, _
  dblYincrement, dblYorigin, nYreference)

If nFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf nFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf nFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

If nType = 0 Then
  Console.WriteLine("Acquire type: NORMal")
ElseIf nType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf nType = 2 Then
  Console.WriteLine("Acquire type: AVERage")
ElseIf nType = 3 Then
  Console.WriteLine("Acquire type: HRESolution")
End If

Console.WriteLine("Waveform points: {0:e}", nPoints)
Console.WriteLine("Waveform average count: {0:e}", nCount)
Console.WriteLine("Waveform X increment: {0:e}", dblXincrement)
Console.WriteLine("Waveform X origin: {0:e}", dblXorigin)
Console.WriteLine("Waveform X reference: {0:e}", nXreference)
Console.WriteLine("Waveform Y increment: {0:e}", dblYincrement)
Console.WriteLine("Waveform Y origin: {0:e}", dblYorigin)
Console.WriteLine("Waveform Y reference: {0:e}", nYreference)

' Read waveform data.
```

```
        myScope.SCPI.WAVeform.DATA.QueryBYTE(byteResultsArray)
        nLength = byteResultsArray.Length
        Console.WriteLine("Number of data values: {0}", nLength)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For i As Integer = 0 To nLength - 2
          writer.WriteLine("{0:f9}, {1:f6}", _
            dblXorigin + (CSng(i) * dblXincrement), _
            ((CSng(byteResultsArray(i)) - nYreference) * _
             dblYincrement) + dblYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format BYTE data written to {0}", _
                          strPath)
    End Sub
  End Class
End Namespace
```

SCPI.NET Example in IronPython

You can also control Keysight oscilloscopes using the SCPI.NET library and Python programming language on the .NET platform using:

- IronPython (http://ironpython.codeplex.com/) which is an implementation of the Python programming language running under .NET.

To run this example with IronPython:

1. Install the Keysight Command Expert software and the command set for the oscilloscope.
2. Cut-and-paste the code that follows into a file named "example.py".
3. Edit the program to use the address of your oscilloscope.
4. If the IronPython "ipy.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:

```
ipy example.py
```

```
#
# Keysight SCPI.NET Example in IronPython
# ******************************************************
# This program illustrates a few commonly used programming
# features of your Keysight oscilloscope.
```

```
# ********************************************************

# Import Python modules.
# --------------------------------------------------------
import sys
sys.path.append("C:\Python26\Lib")   # Python Standard Library.
sys.path.append("C:\ProgramData\Keysight\Command Expert\ScpiNetDrivers")
import string

# Import .NET modules.
# --------------------------------------------------------
from System import *
from System.IO import *
from System.Text import *
from System.Runtime.InteropServices import *
import clr
clr.AddReference("AgInfiniiVision4000X_01_20")
from Keysight.CommandExpert.ScpiNet.AgInfiniiVision4000X_01_20 import *


# ========================================================
# Initialize:
# ========================================================
def initialize():

 # Get and display the device's *IDN? string.
 idn_string = scope.SCPI.IDN.Query()
 print "Identification string '%s'" % idn_string

 # Clear status and load the default setup.
 scope.SCPI.CLS.Command()
 scope.SCPI.RST.Command()


# ========================================================
# Capture:
# ========================================================
def capture():

 # Use auto-scale to automatically set up oscilloscope.
 print "Autoscale."
 scope.SCPI.AUToscale.Command(None, None, None, None, None)

 # Set trigger mode.
 scope.SCPI.TRIGger.MODE.Command("EDGE")
 qresult = scope.SCPI.TRIGger.MODE.Query()
 print "Trigger mode: %s" % qresult

 # Set EDGE trigger parameters.
 scope.SCPI.TRIGger.EDGE.SOURce.Command("CHANnel1")
 qresult = scope.SCPI.TRIGger.EDGE.SOURce.Query()
 print "Trigger edge source: %s" % qresult

 scope.SCPI.TRIGger.EDGE.LEVel.Command(1.5, "CHANnel1")
 qresult = scope.SCPI.TRIGger.EDGE.LEVel.Query("CHANnel1")
 print "Trigger edge level: %s" % qresult
```

```
 scope.SCPI.TRIGger.EDGE.SLOPe.Command("POSitive")
 qresult = scope.SCPI.TRIGger.EDGE.SLOPe.Query()
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 setup_lines = scope.SCPI.SYSTem.SETup.Query()
 nLength = len(setup_lines)
 File.WriteAllLines("setup.stp", setup_lines)
 print "Setup lines saved: %d" % nLength

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 scope.SCPI.CHANnel.SCALe.Command(1, 0.05)
 qresult = scope.SCPI.CHANnel.SCALe.Query(1)
 print "Channel 1 vertical scale: %f" % qresult

 scope.SCPI.CHANnel.OFFSet.Command(1, -1.5)
 qresult = scope.SCPI.CHANnel.OFFSet.Query(1)
 print "Channel 1 offset: %f" % qresult

 # Set horizontal scale and offset.
 scope.SCPI.TIMebase.SCALe.Command(0.0002)
 qresult = scope.SCPI.TIMebase.SCALe.Query()
 print "Timebase scale: %f" % qresult

 scope.SCPI.TIMebase.POSition.Command(0.0)
 qresult = scope.SCPI.TIMebase.POSition.Query()
 print "Timebase position: %f" % qresult

 # Set the acquisition type.
 scope.SCPI.ACQuire.TYPE.Command("NORMal")
 qresult = scope.SCPI.ACQuire.TYPE.Query()
 print "Acquire type: %s" % qresult

 # Or, set up oscilloscope by loading a previously saved setup.
 setup_lines = File.ReadAllLines("setup.stp")
 scope.SCPI.SYSTem.SETup.Command(setup_lines)
 print "Setup lines restored: %d" % len(setup_lines)

 # Capture an acquisition using :DIGitize.
 scope.SCPI.DIGitize.Command("CHANnel1", None, None, None, None)


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
 scope.SCPI.MEASure.SOURce.Command("CHANnel1", None)
 (source1, source2) = scope.SCPI.MEASure.SOURce.Query()
 print "Measure source: %s" % source1

 scope.SCPI.MEASure.FREQuency.Command("CHANnel1")
 qresult = scope.SCPI.MEASure.FREQuency.Query("CHANnel1")
```

```
print "Measured frequency on channel 1: %f" % qresult

# Use direct command/query when commands not in command set.
scope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1")
qresult = scope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1")
print "Measured vertical amplitude on channel 1: %s" % qresult

# Download the screen image.
# ------------------------------------------------------
scope.SCPI.HARDcopy.INKSaver.Command(False)

image_bytes = scope.SCPI.DISPlay.DATA.Query("PNG", "COLor")
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 points mode.
scope.SCPI.WAVeform.POINts.MODE.Command("RAW")
qresult = scope.SCPI.WAVeform.POINts.MODE.Query()
print "Waveform points mode: %s" % qresult

# Get the number of waveform points available.
scope.SCPI.WAVeform.POINts.CommandPoints(10240)
qresult = scope.SCPI.WAVeform.POINts.Query1()
print "Waveform points available: %s" % qresult

# Set the waveform source.
scope.SCPI.WAVeform.SOURce.Command("CHANnel1")
qresult = scope.SCPI.WAVeform.SOURce.Query()
print "Waveform source: %s" % qresult

# Choose the format of the data returned:
scope.SCPI.WAVeform.FORMat.Command("BYTE")
qresult = scope.SCPI.WAVeform.FORMat.Query()
print "Waveform format: %s" % qresult

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "BYTE",
 1 : "WORD",
 4 : "ASCii",
}
acq_type_dict = {
 0 : "NORMal",
 1 : "PEAK",
 2 : "AVERage",
 3 : "HRESolution",
}

(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
 x_reference, y_increment, y_origin, y_reference
```

```
 ) = scope.SCPI.WAVeform.PREamble.Query()

 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

 # Get numeric values for later calculations.
 x_increment = scope.SCPI.WAVeform.XINCrement.Query()
 x_origin = scope.SCPI.WAVeform.XORigin.Query()
 y_increment = scope.SCPI.WAVeform.YINCrement.Query()
 y_origin = scope.SCPI.WAVeform.YORigin.Query()
 y_reference = scope.SCPI.WAVeform.YREFerence.Query()

 # Get the waveform data.
 data_bytes = scope.SCPI.WAVeform.DATA.QueryBYTE()
 nLength = len(data_bytes)
 print "Number of data values: %d" % nLength

 # Open file for output.
 strPath = "waveform_data.csv"
 writer = File.CreateText(strPath)

 # Output waveform data in CSV format.
 for i in xrange(0, nLength - 1):
  time_val = x_origin + i * x_increment
  voltage = (data_bytes[i] - y_reference) * y_increment + y_origin
  writer.WriteLine("%E, %f" % (time_val, voltage))

 # Close output file.
 writer.Close()
 print "Waveform format BYTE data written to %s." % strPath


# ============================================================
# Main program:
# ============================================================
addr = "TCPIP0::10.112.94.136::hislip9-0.0::INSTR"
scope = AgInfiniiVision4000X(addr)
scope.Transport.DefaultTimeout.Set(10000)

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."

# Wait for a key press before exiting.
print "Press any key to exit..."
Console.ReadKey(True)
```

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z