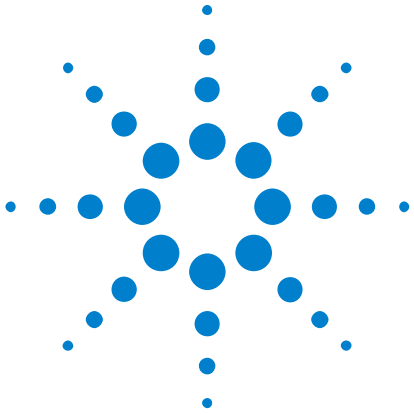


Agilent InfiniiVision 3000 X-Series Oscilloscopes

Programmer's Guide

Agilent Technologies

Notices

Trademarks

Microsoft®, MS-DOS®, Windows®, Windows 2000®, and Windows XP® are U.S. registered trademarks of Microsoft Corporation.

Adobe®, Acrobat®, and the Acrobat Logo® are trademarks of Adobe Systems Incorporated.

Manual Part Number

Version 02.10.0001

Edition

March 2, 2012

Available in electronic format only

Agilent 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, Agilent 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. Agilent 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 Agilent 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 Licenses

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.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

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

WARNING

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

In This Book

This book is your guide to programming the 3000 X-Series oscilloscopes:

Table 1 InfiniiVision 3000 X-Series Oscilloscope Models, Bandwidths, Sample Rates

Bandwidth	**100 MHz**	**200 MHz**	**350 MHz**	**500 MHz**	**1 GHz**
Sample Rate (interleaved, non-interleaved)	**4 GSa/s, 2 GSa/s**	**4 GSa/s, 2 GSa/s**	**4 GSa/s, 2 GSa/s**	**4 GSa/s, 2 GSa/s**	**5 GSa/s, 2.5 GSa/s**
4 analog + 16 digital (mixed signal) channels	MSO-X 3014A	MSO-X 3024A	MSO-X 3034A	MSO-X 3054A	MSO-X 3104A
2 analog + 16 digital (mixed signal) channels	MSO-X 3012A		MSO-X 3032A	MSO-X 3052A	MSO-X 3102A
4 analog channels	DSO-X 3014A	DSO-X 3024A	DSO-X 3034A	DSO-X 3054A	DSO-X 3104A
2 analog channels	DSO-X 3012A		DSO-X 3032A	DSO-X 3052A	DSO-X 3102A

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

- Chapter 1, “What's New,” starting on page 31, 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 55, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, “Commands Quick Reference,” starting on page 69, is a brief listing of the 3000 X-Series oscilloscope commands and syntax.

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

- Chapter 5, “Common (*) Commands,” starting on page 153, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- Chapter 6, “Root (:) Commands,” starting on page 179, describes commands that reside at the root level of the command tree and control many of the basic functions of the oscilloscope.
- Chapter 7, “:ACQuire Commands,” starting on page 219, describes commands for setting the parameters used when acquiring and storing data.
- Chapter 8, “:BUS<n> Commands,” starting on page 233, describes commands that control all oscilloscope functions associated with the digital channels bus display.

- Chapter 9, ":CALibrate Commands," starting on page 243, describes utility commands for determining the state of the calibration factor protection button.
- Chapter 10, ":CHANnel<n> Commands," starting on page 253, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- Chapter 11, ":DEMO Commands," starting on page 273, describes commands that control the education kit (Option EDU) demonstration signals that can be output on the oscilloscope's Demo 1 and Demo 2 terminals.
- Chapter 12, ":DIGital<d> Commands," starting on page 281, describes commands that control all oscilloscope functions associated with individual digital channels.
- Chapter 13, ":DISPlay Commands," starting on page 289, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- Chapter 14, ":EXTernal Trigger Commands," starting on page 301, describes commands that control the input characteristics of the external trigger input.
- Chapter 15, ":FUNCtion Commands," starting on page 307, describes commands that control math waveforms.
- Chapter 16, ":HARDcopy Commands," starting on page 341, describes commands that set and query the selection of hardcopy device and formatting options.
- Chapter 17, ":LISTer Commands," starting on page 359, describes commands that turn on/off the Lister display for decoded serial data and get the Lister data.
- Chapter 18, ":MARKer Commands," starting on page 363, 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 19, ":MEASure Commands," starting on page 379, describes commands that select automatic measurements (and control markers).
- Chapter 20, ":MEASure Power Commands," starting on page 447, describes measurement commands that are available when the DSOX3PWR power measurements and analysis application is licensed and enabled.
- Chapter 21, ":MTESt Commands," starting on page 467, describes commands that control the mask test features provided with Option LMT.
- Chapter 22, ":POD Commands," starting on page 501, describes commands that control all oscilloscope functions associated with groups of digital channels.
- Chapter 23, ":POWer Commands," starting on page 507, describes commands that control the DSOX3PWR power measurement application.

- Chapter 24, ":RECall Commands," starting on page 565, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- Chapter 25, ":SAVE Commands," starting on page 573, describes commands that save oscilloscope setups, screen images, and data.
- Chapter 26, ":SBUS<n> Commands," starting on page 595, describes commands that control oscilloscope functions associated with the serial decode bus and serial triggering.
- Chapter 27, ":SEARch Commands," starting on page 743, describes commands that control oscilloscope functions associated with searching for waveform events.
- Chapter 28, ":SYSTem Commands," starting on page 817, describes commands that control basic system functions of the oscilloscope.
- Chapter 29, ":TIMebase Commands," starting on page 831, describes commands that control all horizontal sweep functions.
- Chapter 30, ":TRIGger Commands," starting on page 843, describes commands that control the trigger modes and parameters for each trigger type.
- Chapter 31, ":WAVeform Commands," starting on page 923, describes commands that provide access to waveform data.
- Chapter 32, ":WGEN Commands," starting on page 959, describes commands that control waveform generator (Option WGN) functions and parameters.
- Chapter 33, ":WMEMory<r> Commands," starting on page 985, describes commands that control reference waveforms.
- Chapter 34, "Obsolete and Discontinued Commands," starting on page 995, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 35, "Error Messages," starting on page 1047, 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 36, "Status Reporting," starting on page 1055, describes the oscilloscope's status registers and how to check the status of the instrument.
- Chapter 37, "Synchronizing Acquisitions," starting on page 1077, describes how to wait for acquisitions to complete before querying

measurement results or performing other operations with the captured data.

- Chapter 38, “More About Oscilloscope Commands,” starting on page 1087, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 39, “Programming Examples,” starting on page 1097.

Mixed-Signal Oscilloscope Channel Differences

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

See Also

- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Agilent IO Libraries Suite.
- For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Agilent 82350B GPIB interface).
- For information on oscilloscope front-panel operation, see the *User's Guide*.
- For detailed connectivity information, refer to the *Agilent Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to "www.agilent.com" and search for "Connectivity Guide".
- For the latest versions of this and other manuals, see: "http://www.agilent.com/find/3000X-Series-manual"

Contents

6 Root (:) Commands

7 :ACQuire Commands

18 :MARKer Commands

19 :MEASure Commands

21 :MTESt Commands

22 :POD Commands

23 :POWer Commands

27 :SEARch Commands

32 :WGEN Commands

33 :WMEMory<r> Commands

34 Obsolete and Discontinued Commands

1 What's New

What's New in Version 2.10

New features in version 2.10 of the InfiniiVision 3000 X-Series oscilloscope software are:

- Support for adding an annotation to the display.

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

New Commands

Command	Description
:DISPlay:ANNotation (see page 291)	Turns screen annotation on or off.
:DISPlay:ANNotation:BACKground (see page 292)	Specifies the background of the annotation to be either opaque, inverted, or transparent.
:DISPlay:ANNotation:COLor (see page 293)	Specifies the color of the annotation.
:DISPlay:ANNotation:TEXT (see page 294)	Specifies the annotation string, up to 254 characters.

What's New in Version 2.00

New features in version 2.00 of the InfiniiVision 3000 X-Series oscilloscope software are:

- Support for the DSOX3WAVEGEN waveform generator's new arbitrary waveform type.
- Support for the new DSOX3VID extended Video triggering license.
- Support for the new DSOX3AERO MIL-STD-1553 and ARINC 429 triggering and serial decode license.
- Support for the new DSOX3FLEX FlexRay triggering and serial decode license.
- Support for the new DSOX3PWR power measurements and analysis license.
- Support for the new DSOX3ADVMATH advanced math measurements license.

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

New Commands

Command	Description
:FUNCtion:BUS:CLOCk (see page 312)	Selects the clock signal source for the Chart Logic Bus State operation. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:BUS:SLOPe (see page 313)	Specifies the clock signal edge for the Chart Logic Bus State operation. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:BUS:YINCrement (see page 314)	Specifies the value associated with each increment in Chart Logic Bus data. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:BUS:YORigin (see page 315)	Specifies the value associated with Chart Logic Bus data equal to zero. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:BUS:YUNits (see page 316)	Specifies the vertical units for the Chart Logic Bus operations. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:FREQuency:HIGHpass (see page 322)	Sets the high-pass filter's -3 dB cutoff frequency. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:FREQuency:LOWPass (see page 323)	Sets the low-pass filter's -3 dB cutoff frequency. Available with the DSOX3ADVMATH advanced math license.

Command	Description
:FUNCtion:LINear:GAIN (see page 328)	Specifies the 'A' value in the Ax + B operation. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:LINear:OFFSet (see page 329)	Specifies the 'B' value in the Ax + B operation. Available with the DSOX3ADVMATH advanced math license.
:FUNCtion:TRENd:MEASurement (see page 339)	Selects the measurement whose trend is shown in the math waveform. Available with the DSOX3ADVMATH advanced math license.
:MEASure Power Commands (see page 447)	:MEASure commands available when the DSOX3PWR power measurements and analysis application is licensed and enabled.
:MEASure:STATistics:DISPlay (see page 424)	Specifies whether the display of measurement statistics is on or off.
:POWer Commands (see page 507)	Commands for the DSOX3PWR power measurements and analysis application.
:PWRenable (Power Event Enable Register) (see page 208)	For enabling bits in the Power Event Enable Register. This status register control is available when the DSOX3PWR power measurements and analysis application is licensed.
:PWRRegister[:EVENt] (Power Event Event Register) (see page 208)	For reading power application status bits in the Power Event Event Register. This query is available when the DSOX3PWR power measurements and analysis application is licensed.
:RECall:ARBitrary[:STARt] (see page 567)	Recalls waveform generator arbitrary waveforms from a file.
:SAVE:ARBitrary[:STARt] (see page 576)	Saves waveform generator arbitrary waveforms to a file.
:SAVE:POWer[:STARt] (see page 585)	Saves the power measurement application's current harmonics analysis results to a file.
:SBUS<n>:A429 Commands (see page 600)	Commands for ARINC 429 triggering and serial decode.
:SBUS<n>:FLEXray Commands (see page 635)	Commands for FlexRay triggering and serial decode.
:SBUS<n>:M1553 Commands (see page 697)	Commands for MIL-STD 1553 triggering and serial decode.
:SEARch:SERial:A429 Commands (see page 768)	Commands for finding ARINC 429 events in the captured data.

Command	Description
:SEARch:SERial:FLEXray Commands (see page 780)	Commands for finding FlexRay events in the captured data.
:SEARch:SERial:M1553 Commands (see page 805)	Commands for finding MIL-STD 1553 events in the captured data.
:TRIGger:TV:UDTV:ENUMber (see page 914)	Specifies the Nth edge to trigger on with the Generic video trigger. Available with the DSOX3VID extended Video triggering license.
:TRIGger:TV:UDTV:HSYNc (see page 915)	Enables or disables the horizontal sync control in the Generic video trigger. Available with the DSOX3VID extended Video triggering license.
:TRIGger:TV:UDTV:HTIMe (see page 916)	When the Generic video trigger's horizontal sync control is enabled, this command specifies the sync time. Available with the DSOX3VID extended Video triggering license.
:TRIGger:TV:UDTV:PGTHan (see page 917)	Specifies the "greater than the sync pulse width" time in the Generic video trigger. Available with the DSOX3VID extended Video triggering license.
:WGEN:ARBitrary:BYTeorder (see page 962)	Selects the byte order for arbitrary waveform binary transfers.
:WGEN:ARBitrary:DATA (see page 963)	Downloads an arbitrary waveform in floating-point values format.
:WGEN:ARBitrary:DATA:ATTRibute:POINts? (see page 964)	Returns the number of points used by the current arbitrary waveform.
:WGEN:ARBitrary:DATA:CLEar (see page 965)	Clears the arbitrary waveform memory and restores the default waveform.
:WGEN:ARBitrary:DATA:DAC (see page 966)	Downloads an arbitrary waveform in integer (DAC) values.
:WGEN:ARBitrary:INTerpolate (see page 967)	Enable or disables interpolated values between points in the arbitrary waveform.
:WGEN:ARBitrary:STORe (see page 968)	Captures a waveform and stores it into arbitrary waveform memory.
:WGEN:MODulation:NOISe (see page 976)	Adds noise to the waveform generator's output signal.

Changed Commands

Command	Differences
:DEMO:FUNCtion (see page 274)	The FMBurst, ARINc, FLEXray, MIL, and MIL2 functions are now available with the DSOXEDK educator's kit license.
:FUNCtion:OPERation (see page 331)	The MAGNify, ABSolute, SQUare, LN, LOG, EXP, TEN, LOWPass, HIGHpass, DIVide, LINear, TRENd, BTIMing, and BSTate operations are now available with the DSOX3ADVMATH advanced math measurements license.
:FUNCtion:SOURce1 (see page 336)	The BUS<m> source is now available for the bus charting operations available with the DSOX3ADVMATH advanced math measurements license.
:SBUS<n>:MODE (see page 599)	The A429, M1553, and FLEXray modes are now available with the DSOX3AERO (MIL-STD-1553 and ARINC 429) and DSOX3FLEX (FlexRay) serial decode and triggering licenses.
:TRIGger:TV:MODE (see page 910)	The LINE mode is added for the video standards available with the extended Video triggering license.
:TRIGger:TV:STANdard (see page 913)	Lets you select additional video standards available with the extended Video triggering license.
:WGEN:FUNCtion (see page 970)	The ARBitrary waveform type can now be selected.

What's New in Version 1.20

New features in version 1.20 of the InfiniiVision 3000 X-Series oscilloscope software are:

- Edge Then Edge trigger.
- OR'ed edge trigger.
- Sine Cardinal, Exponential Rise, Exponential Fall, Cardiac, and Gaussian Pulse waveform generator waveforms.
- X cursor units that let you measure time (seconds), frequency (Hertz), phase (degrees), and ratio (percent), and Y cursor units that let you measure the channel units (base) or ratio (percent).
- Option for specifying FFT vertical units as V RMS as well as decibels.
- Option for entering a DC offset correction factor for the integrate math waveform input signal.
- Option for saving the maximum number of waveform data points.

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

New Commands

Command	Description
:FUNCtion:INTegrate:IOFFset (see page 327)	Lets you enter a DC offset correction factor for the integrate math waveform input signal to level a "ramp"ed waveform.
:FUNCtion[:FFT]:VTYPe (see page 320)	Specifies FFT vertical units as DECibel or VRMS.
:MARKer:XUNIts (see page 371)	Specifies the units for X cursors.
:MARKer:XUNIts:USE (see page 372)	Sets the current X1 and X2 cursor locations as 0 and 360 degrees if XUNIts is DEGRees or as 0 and 100 percent if XUNIts is PERCent.
:MARKer:YUNIts (see page 376)	Specifies the units for Y cursors.
:MARKer:YUNIts:USE (see page 377)	Sets the current Y1 and Y2 cursor locations as 0 and 100 percent if YUNIts is PERCent.
:MEASure:STATistics:MCOunt (see page 426)	Specifies the maximum number of values used when calculating measurement statistics.
:MEASure:STATistics:RSDeviation (see page 428)	Disables or enables relative standard deviations, that is, standard deviation/mean, in the measurement statistics.
:SAVE:WAVeform:LENGth:MAX (see page 591)	Enable or disables saving the maximum number of waveform data points.

Command	Description
:TRIGger:DELay:ARM:SLOPe (see page 855)	Specifies the arming edge slope for the Edge Then Edge trigger.
:TRIGger:DELay:ARM:SOURce (see page 856)	Specifies the arming edge source for the Edge Then Edge trigger.
:TRIGger:DELay:TDELay:TIME (see page 857)	Specifies the delay time for the Edge Then Edge trigger.
:TRIGger:DELay:TRIGger:COUNt (see page 858)	Specifies the trigger edge count for the Edge Then Edge trigger.
:TRIGger:DELay:TRIGger:SLOPe (see page 859)	Specifies the trigger edge slope for the Edge Then Edge trigger.
:TRIGger:DELay:TRIGger:SOURce (see page 860)	Specifies the trigger edge source for the Edge Then Edge trigger.
:TRIGger:FORCe (see page 846)	Now documented, this command is equivalent to the front panel **[Force Trigger]** key which causes an acquisition to be captured even though the trigger condition has not been met.
:TRIGger:OR (see page 882)	Specifies edges for the OR'ed edge trigger.

Changed Commands

Command	Differences
:DEMO:FUNCtion (see page 274)	The ETE (Edge then Edge) function has been added.
:TRIGger:MODE (see page 851)	The OR and DELay modes are added for the OR'ed edge trigger and the Edge Then Edge trigger.
:WGEN:FUNCtion (see page 970)	The SINC, EXPRise, EXPFall, CARDiac, and GAUSsian waveform types can now be selected.

What's New in Version 1.10

New command descriptions for Version 1.10 of the InfiniiVision 3000 X-Series oscilloscope software appear below.

New Commands

Command	Description
:SYSTem:PRESet (see page 824)	Now documented, this command is equivalent to the front panel **[Default Setup]** key which leaves some user settings, like preferences, unchanged. The *RST command is equivalent to a factory default setup where no user settings are left unchanged.

Version 1.00 at Introduction

The Agilent InfiniiVision 3000 X-Series oscilloscopes were introduced with version 1.00 of oscilloscope operating software.

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

Command Differences From 7000B Series Oscilloscopes

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

The main differences between the version 1.00 programming command set for the InfiniiVision 3000 X-Series oscilloscopes and the 6.10 programming command set for the InfiniiVision 7000B Series oscilloscopes are related to:

- Built-in waveform generator (with Option WGN license).
- Built-in demo signals (with Option EDU license that comes with the N6455A Education Kit).
- Reference waveforms (in place of trace memory).
- Multiple serial decode waveforms.
- Serial decode now available on 2-channel oscilloscopes.
- Enhanced set of trigger types.
- Additional measurements.
- Different path name format for internal and USB storage device locations.

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

New Commands

Command	Description
:DEMO Commands (see page 273)	Commands for using built-in demo signals (with the Option EDU license that comes with the N6455A Education Kit).
:HARDcopy:NETWork Commands (see page 341)	For accessing network printers.
:MEASure:AREA (see page 392)	Measures the area between the waveform and the ground level.
:MEASure:BWIDth (see page 393)	Measures the burst width from the first edge on screen to the last.
:MEASure:NEDGes (see page 404)	Counts the number of falling edges.
:MEASure:NPULses (see page 405)	Counts the number of negative pulses.
:MEASure:PEDGes (see page 409)	Counts the number of rising edges.

Command	Description
:MEASure:PPULses (see page 412)	Counts the number of positive pulses.
:MEASure:WINDow (see page 443)	When the zoomed time base in on, specifies whether the measurement window is the zoomed time base or the main time base.
:MTESt:ALL (see page 472)	Specifies whether all channels are included in the mask test.
:RECall:WMEMory<r>[:STARt] (see page 572)	Recalls reference waveforms.
:SAVE:WMEMory:SOURce (see page 593)	Selects the source for saving a reference waveform.
:SAVE:WMEMory[:STARt] (see page 594)	Saves reference waveforms.
:SBUS<n>:CAN Commands (see page 618)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:CAN subsystem.
:SBUS<n>:I2S Commands (see page 654)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:I2S subsystem.
:SBUS<n>:IIC Commands (see page 673)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:IIC subsystem.
:SBUS<n>:LIN Commands (see page 683)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:LIN subsystem.
:SBUS<n>:SPI Commands (see page 618)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:SPI subsystem.
:SBUS<n>:UART Commands (see page 721)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:UART subsystem.
:SEARch:EDGE Commands (see page 748)	Commands for searching edge events.
:SEARch:GLITch Commands (see page 751)	Commands for searching glitch events.
:SEARch:RUNT Commands (see page 758)	Commands for searching runt events.
:SEARch:TRANsition Commands (see page 748)	Commands for searching edge transition events.
:TRIGger:LEVel:HIGH (see page 849)	Sets runt and transition (rise/fall time) trigger high level.
:TRIGger:LEVel:LOW (see page 850)	Sets runt and transition (rise/fall time) trigger low level.
:TRIGger:PATTern Commands (see page 883)	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:DURation subsystem.

Command	Description
:TRIGger:RUNT Commands (see page 892)	Commands for triggering on runt pulses.
:TRIGger:SHOLd Commands (see page 897)	Commands for triggering on setup and hold time violations.
:TRIGger:TRANsition Commands (see page 903)	Commands for triggering on edge transitions.
:WGEN Commands (see page 959)	Commands for controlling the built-in waveform generator (with Option WGN license).
:WMEMory<r> Commands (see page 985)	Commands for reference waveforms.

Changed Commands

Command	Differences From InfiniiVision 7000B Series Oscilloscopes
:ACQuire:MODE (see page 223)	There is no ETIMe parameter with the 3000 X-Series oscilloscopes.
:CALibrate:OUTPut (see page 247)	The TRIG OUT signal can be a trigger output, mask test failure, or waveform generator sync pulse.
:DISPlay:DATA (see page 296)	Monochrome TIFF images of the graticule cannot be saved or restored.
:DISPlay:LABList (see page 298)	The label list contains up to 77, 10-character labels (instead of 75).
:DISPlay:VECTors (see page 300)	Always ON with 3000 X-Series oscilloscopes.
:MARKer Commands (see page 363)	Can select reference waveforms as marker source.
:MEASure Commands (see page 379)	Can select reference waveforms as the source for many measurements.
General :SBUS<n> Commands (see page 597)	With multiple serial decode waveforms, "SBUS" is now "SBUS1" or "SBUS2".
:SAVE:IMAGe[:STARt] (see page 578)	Cannot save images to internal locations.
:SEARch:MODE (see page 746)	Can select EDGE, GLITch, RUNT, and TRANsition modes. Also, SERial is now SERial{1 | 2}.
:SEARch:SERial:IIC:MODE (see page 793)	ANACknowledge parameter is now ANACk.
:TRIGger:PATTern (see page 884)	Takes <string> parameter instead of <value>,<mask> parameters.

Command	Differences From InfiniiVision 7000B Series Oscilloscopes
:WAVeform:SOURce (see page 945)	Can select reference waveforms as the source.
:VIEW (see page 217)	PMEMory (pixel memory) locations are not present.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences

Discontinued Commands

Command	Description
:ACQuire:RSIGnal	The 3000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.
:CALibrate:SWITch?	Replaced by :CALibrate:PROTected? (see page 248). The oscilloscope has a protection button instead of a switch.
:DISPlay:SOURce	PMEMory (pixel memory) locations are not present.
:EXTernal:IMPedance	External TRIG IN connector is now fixed at 1 MOhm.
:EXTernal:PROBe:ID	Not supported on external TRIG IN connector.
:EXTernal:PROBe:STYPe	Not supported on external TRIG IN connector.
:EXTernal:PROTection	Not supported on external TRIG IN connector.
:HARDcopy:DEVice, :HARDcopy:FORMat	Use the :SAVE:IMAGe:FORMat, :SAVE:WAVeform:FORMat, and :HARDcopy:APRinter commands instead.
:MERGe	Waveform traces have been replaced by reference waveforms.
:RECall:IMAGe[:STARt]	Waveform traces have been replaced by reference waveforms.
:SYSTem:PRECision	The 3000 X-Series oscilloscopes' measurement record is 62,500 points, and there is no need for a special precision mode.
:TIMebase:REFClock	The 3000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.

2 Setting Up

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

Step 1. Install Agilent IO Libraries Suite software

1 Download the Agilent IO Libraries Suite software from the Agilent web site at:

- "http://www.agilent.com/find/iolib"

2 Run the setup file, and follow its installation instructions.

Step 2. Connect and set up the oscilloscope

The 3000 X-Series oscilloscope has three different interfaces you can use for programming:

- USB (device port).
- LAN, when the LAN/VGA option module is installed. To configure the LAN interface, press the **[Utility]** key on the front panel, then press the **I/O** softkey, then press the **Configure** softkey.
- GPIB, when the GPIB option module is installed.

When installed, these interfaces are always active.

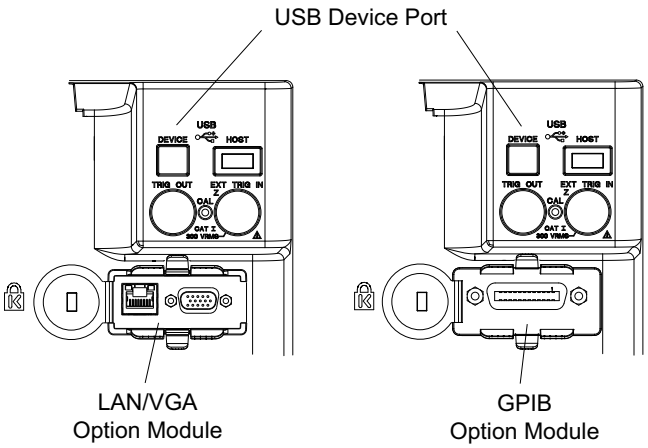


Figure 1 Control Connectors on Rear Panel

Using the USB (Device) Interface

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

This is a USB 2.0 high-speed port.

Using the LAN Interface

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

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

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

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

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

4 Configure the oscilloscope's LAN interface:

   a Press the **Configure** softkey until "LAN" is selected.

   b Press the **LAN Settings** softkey.

   c Press the **Config** softkey, and enable all the configuration options supported by your network.

   d If automatic configuration is not supported, press the **Addresses** softkey.

   Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the IP Address, Subnet Mask, Gateway IP, and DNS IP values.

   When you are done, press the **[Back up]** key.

   e Press the **Host name** softkey. Use the softkeys and the Entry knob to enter the Host name.

   When you are done, press the **[Back up]** key.

Using the GPIB Interface

1 Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the GPIB option module.

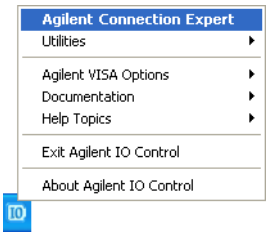
2 Configure the oscilloscope's GPIB interface:

   a Press the **Configure** softkey until "GPIB" is selected.

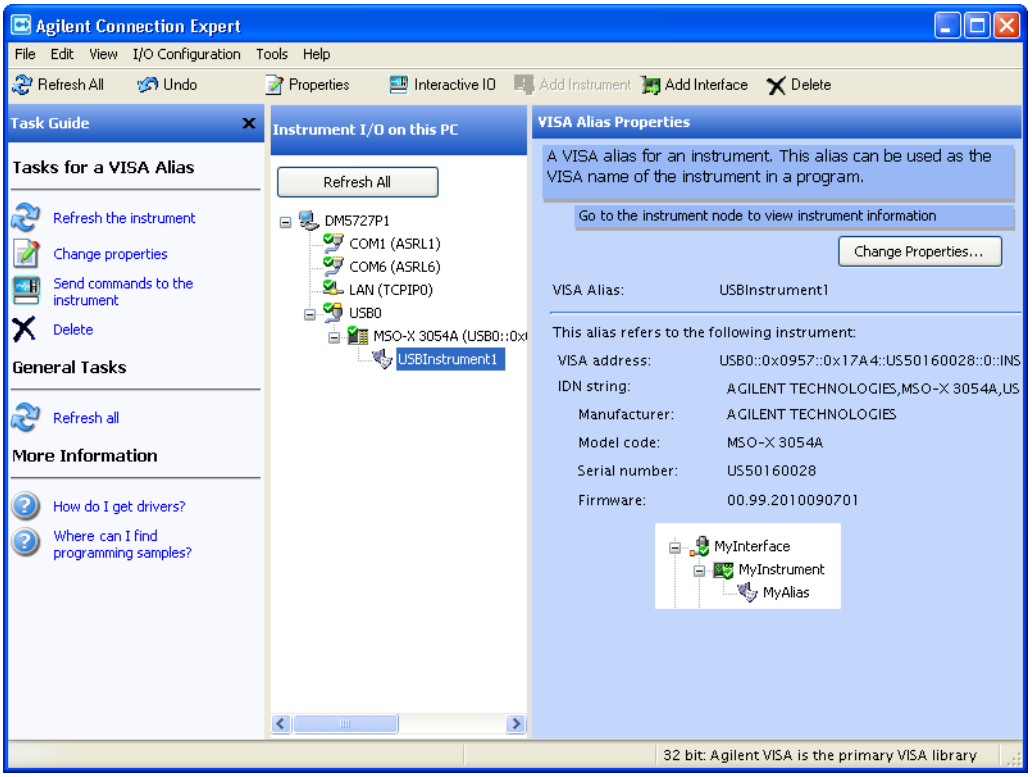
   b Use the Entry knob to select the **Address** value.

Step 3. Verify the oscilloscope connection

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

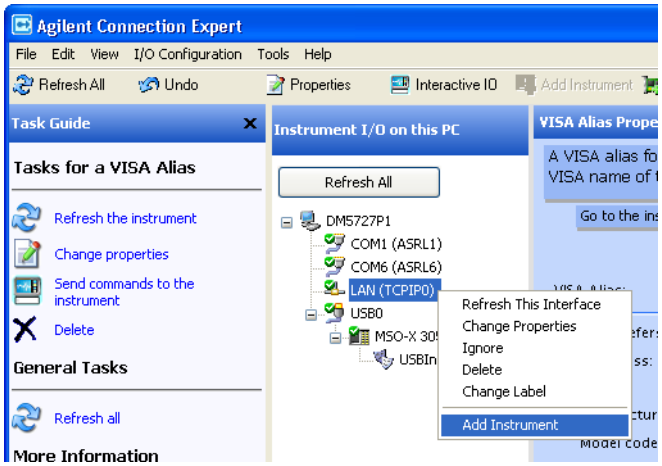


2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)

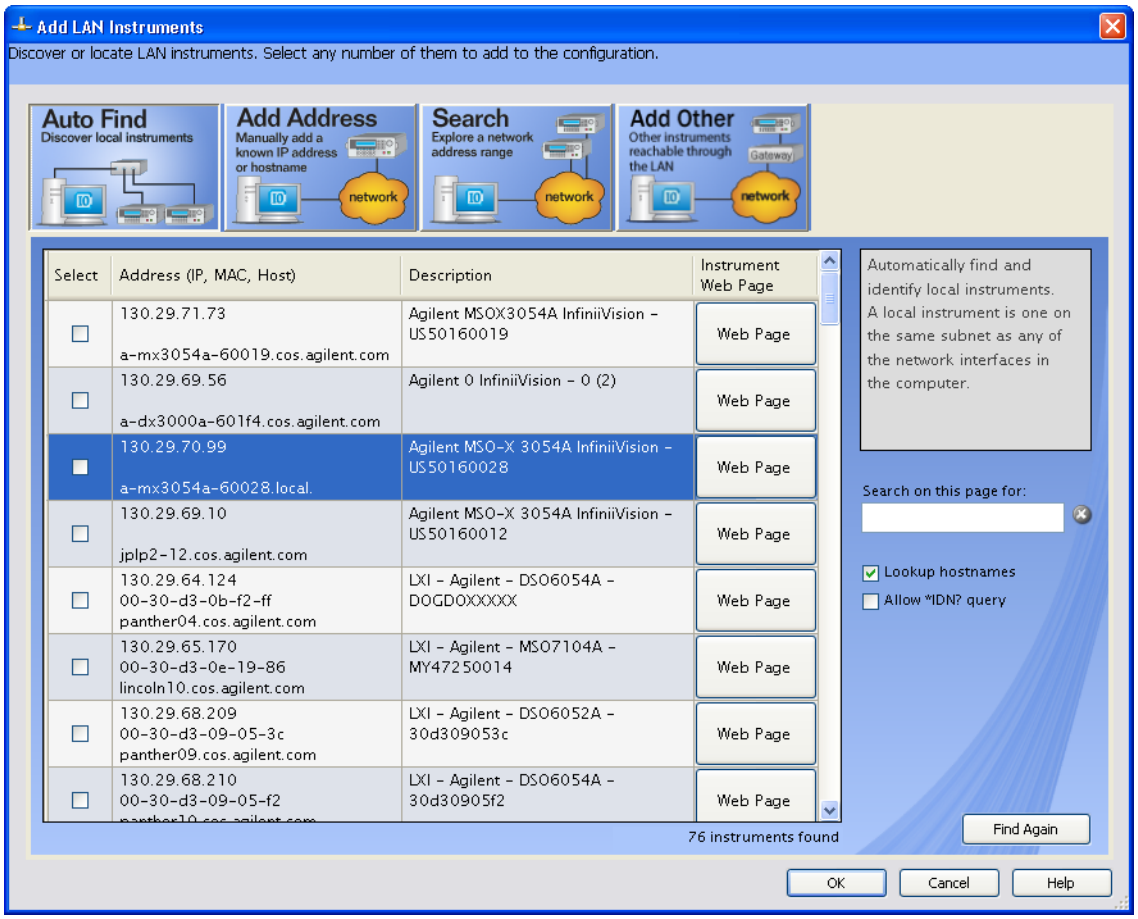


You must manually add instruments on LAN interfaces:

a Right-click on the LAN interface, choose **Add Instrument** from the popup menu



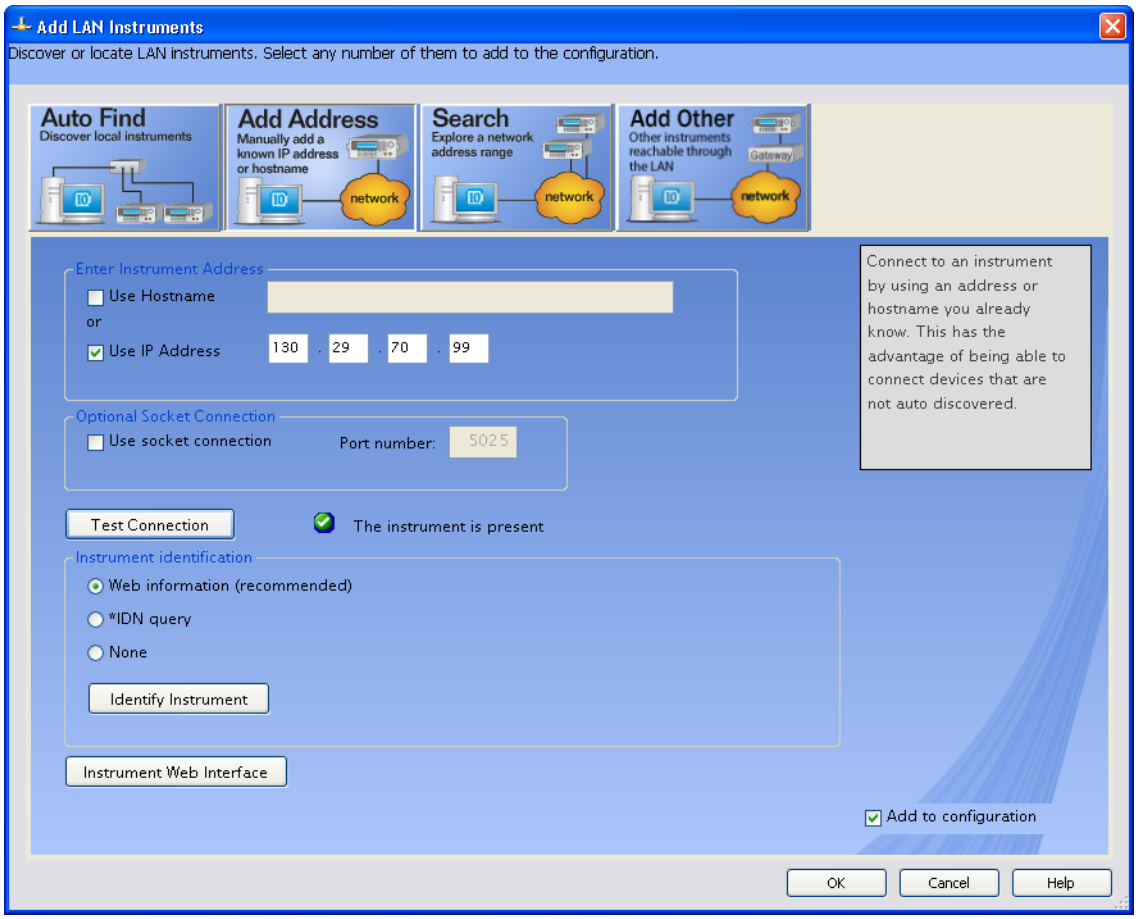
b If the oscilloscope is on the same subnet, select it, and click **OK**.



Otherwise, if the instrument is not on the same subnet, click **Add Address**.

i In the next dialog, select either **Hostname** or **IP address**, and enter the oscilloscope's hostname or IP address.

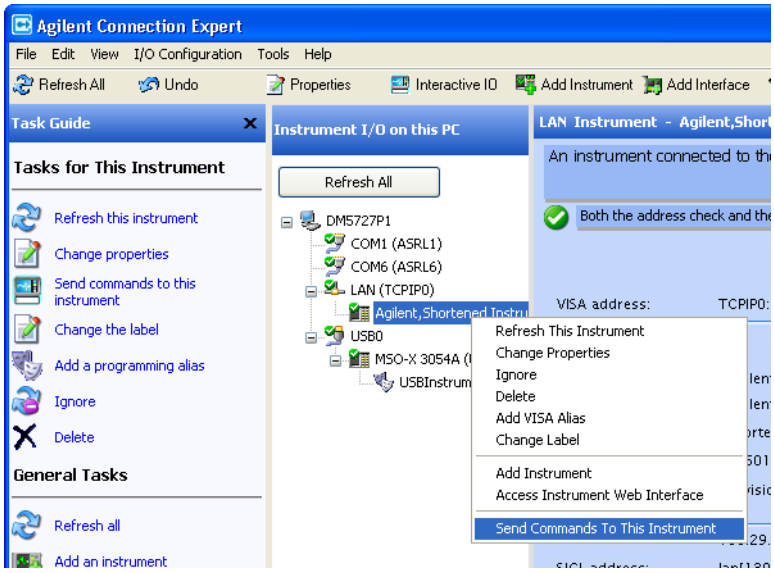
ii Click **Test Connection**.



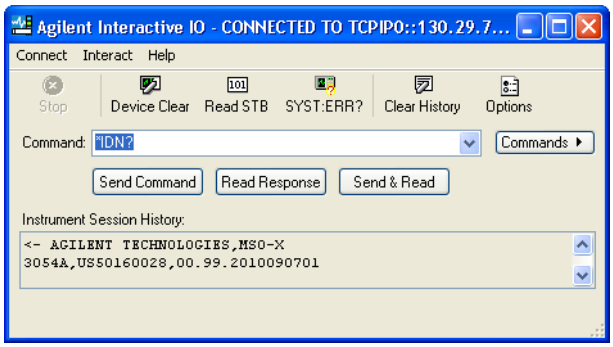
iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

3 Test some commands on the instrument:

a Right-click on the instrument and choose **Send Commands To This Instrument** from the popup menu.



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



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

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

3 Getting Started

This chapter gives you an overview of programming the 3000 X-Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

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

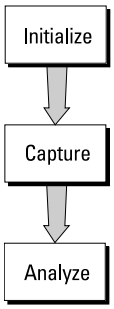
NOTE

Language for Program Examples

The programming examples in this guide are written in Visual Basic using the Agilent 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. Agilent 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

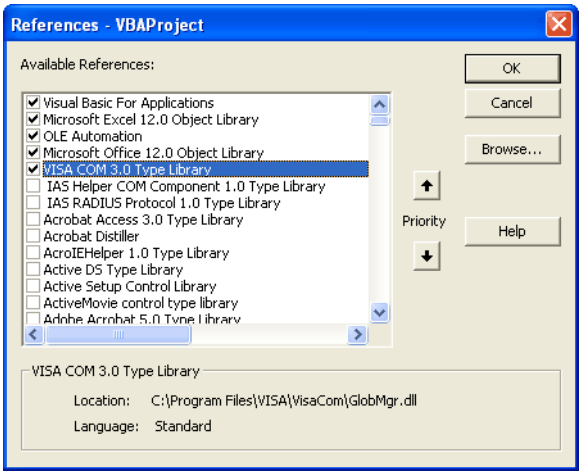
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

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

To reference the Agilent 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 3.0 Type Library".



3 Click **OK**.

To reference the Agilent 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 3.0 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 Agilent 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
' Agilent Connection Expert (installed with Agilent 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 1089.

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 Agilent 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
' Agilent Connection Expert (installed with Agilent 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 5, "Common (*) Commands," starting on page 153.

Refer to the Agilent 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 31, ":WAVeform Commands," starting on page 923.

NOTE

Aborting a Digitize Operation Over the Programming Interface

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

Reading Query Responses from the Oscilloscope

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

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

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

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

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

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

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

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

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

Reading Query Results into String Variables

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

NOTE

Express String Variables Using Exact Syntax

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

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

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

After running this program, the controller displays:

Range (string): +40.0E+00

Reading Query Results into Numeric Variables

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

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

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

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

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

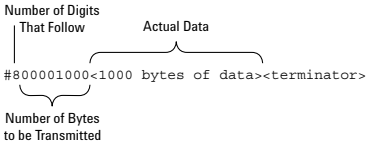


Figure 2 Definite-length block response data

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

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

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

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

Sending Multiple Queries and Reading Results

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

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

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

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

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

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

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

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

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

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 36, “Status Reporting,” starting on page 1055 which explains how to check the status of the instrument.

Other Ways of Sending Commands

Standard Commands for Programmable Instrumentation (SCPI) can also be sent via a Telnet socket or through the Browser Web Control:

Telnet Sockets

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

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

```
telnet <hostname> 5024
```

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

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

```
telnet <hostname> 5025
```

Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the *InfiniiVision 3000 X-Series Oscilloscopes User's Guide*. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the **Browser Web Control** tab, then select the **Remote Programming** link.

4 Commands Quick Reference

Command Summary

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 157)	n/a	n/a
*ESE <mask> (see page 158)	*ESE? (see page 158)	<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 160)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 160)	AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 163)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 164)	*OPC? (see page 164)	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 165)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <reserved>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <reserved>, <Power Measurements>, <RS-232/UART Serial>, <Segmented Memory>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <I2S Serial>, <reserved>, <reserved>, <Waveform Generator>, <reserved>, <reserved> <All field> ::= {0 | All} <reserved> ::= 0 <MSO> ::= {0 | MSO} <Memory> ::= {0 | MEMUP} <Low Speed Serial> ::= {0 | EMBD} <Automotive Serial> ::= {0 | AUTO} <Power Measurements> ::= {0 | PWR} <RS-232/UART Serial> ::= {0 | COMP} <Segmented Memory> ::= {0 | SGM} <Mask Test> ::= {0 | MASK} <Bandwidth> ::= {0 | BW20 | BW50} <I2S Serial> ::= {0 | AUDIO} <Waveform Generator> ::= {0 | WAVEGEN}
*RCL <value> (see page 167)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 168)	n/a	See *RST (Reset) (see page 168)
*SAV <value> (see page 171)	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 172)	*SRE? (see page 173)	<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 174)	<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 176)	n/a	n/a
n/a	*TST? (see page 177)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 178)	n/a	n/a

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 183)	:ACTivity? (see page 183)	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER? (see page 184)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 185)	n/a	<source> ::= CHANnel<n> for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD1 | POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODE <value> (see page 187)	:AUToscale:AMODE? (see page 187)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 188)	:AUToscale:CHANnels? (see page 188)	<value> ::= {ALL | DISPlayed}}
:AUToscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 189)	:AUToscale:FDEBug? (see page 189)	{0 | 1}
:BLANk [<source>] (see page 190)	n/a	<source> ::= {CHANnel<n>} | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,..,<source>]] (see page 191)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:MTEenable <n> (see page 193)	:MTEenable? (see page 193)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENt]? (see page 195)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 197)	:OPEE? (see page 198)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDition? (see page 199)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister[:EVENt]? (see page 201)	<n> ::= 15-bit integer in NR1 format
:OVLenable <mask> (see page 203)	:OVLenable? (see page 204)	<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 205)	<value> ::= integer in NR1 format. See OVLenable for <value>

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:PRINt [<options>] (see page 207)	n/a	<options> ::= [<print option>][,..,<print option>] <print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG | NOFactors | FACTors} <print option> can be repeated up to 5 times.
:PWRenable <n> (see page 208)	:PWRenable? (see page 208)	<n> ::= 16-bit integer in NR1 format
n/a	:PWRRegister[:EVENt]? (see page 210)	<n> ::= 16-bit integer in NR1 format
:RUN (see page 211)	n/a	n/a
n/a	:SERial (see page 212)	<return value> ::= unquoted string containing serial number
:SINGle (see page 213)	n/a	n/a
n/a	:STATus? <display> (see page 214)	{0 | 1} <display> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:STOP (see page 215)	n/a	n/a
n/a	:TER? (see page 216)	{0 | 1}
:VIEW <source> (see page 217)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMPlete <complete> (see page 221)	:ACQuire:COMPlete? (see page 221)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 222)	:ACQuire:COUNt? (see page 222)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 223)	:ACQuire:MODE? (see page 223)	<mode> ::= {RTIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 224)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMented:ANALyze (see page 225)	n/a	n/a (with Option SGM)
:ACQuire:SEGMented:COUNt <count> (see page 226)	:ACQuire:SEGMented:COUNt? (see page 226)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
:ACQuire:SEGMented:INDex <index> (see page 227)	:ACQuire:SEGMented:INDex? (see page 227)	<index> ::= an integer from 1 to 1000 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATe? (see page 230)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 231)	:ACQuire:TYPE? (see page 231)	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Table 5 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}} (see page 235)	:BUS<n>:BIT<m>? (see page 235)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 | OFF} | {1 | ON}} (see page 236)	:BUS<n>:BITS? (see page 236)	<channel_list>, {0 | 1} <channel_list> ::= (@<m>,<m>:<m>...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format

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

Command	Query	Options and Query Returns
:BUS<n>:CLEar (see page 238)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 239)	:BUS<n>:DISPlay? (see page 239)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:LABel <string> (see page 240)	:BUS<n>:LABel? (see page 240)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 241)	:BUS<n>:MASK? (see page 241)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

Table 6 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 245)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 246)	:CALibrate:LABel? (see page 246)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 247)	:CALibrate:OUTPut? (see page 247)	<signal> ::= {TRIGgers | MASK | WAVEgen}
n/a	:CALibrate:PROTected? (see page 248)	{PROTected | UNPRotected}
:CALibrate:STARt (see page 249)	n/a	n/a

Table 6 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:STATus? (see page 250)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see page 251)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 252)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 7 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 256)	:CHANnel<n>:BWLimit? (see page 256)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 257)	:CHANnel<n>:COUPling? (see page 257)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 258)	:CHANnel<n>:DISPlay? (see page 258)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 259)	:CHANnel<n>:IMPedance? (see page 259)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 260)	:CHANnel<n>:INVert? (see page 260)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 261)	:CHANnel<n>:LABel? (see page 261)	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 262)	:CHANnel<n>:OFFSet? (see page 262)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format

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

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe <attenuation> (see page 263)	:CHANnel<n>:PROBe? (see page 263)	<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 264)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 264)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 265)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 266)	:CHANnel<n>:PROBe:SKEW? (see page 266)	<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 267)	:CHANnel<n>:PROBe:STYPe? (see page 267)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 268)	:CHANnel<n>:PROTection? (see page 268)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 269)	:CHANnel<n>:RANGe? (see page 269)	<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 270)	:CHANnel<n>:SCALe? (see page 270)	<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 271)	:CHANnel<n>:UNITs? (see page 271)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 272)	:CHANnel<n>:VERNier? (see page 272)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format

Table 8 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCtion <signal> (see page 274)	:DEMO:FUNCtion? (see page 276)	<signal> ::= {SINusoid | NOISy | PHASe | RINGing | SINGle | AM | CLK | GLITch | BURSt | MSO | RUNT | TRANsition | RFBurst | SHOLd | LFSine | FMBurst | ETE | CAN | LIN | UART | I2C | SPI | I2S | CANLin | ARINc | FLEXray | MIL | MIL2}
:DEMO:FUNCtion:PHASe:PHASe <angle> (see page 278)	:DEMO:FUNCtion:PHASe:PHASe? (see page 278)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 | OFF} | {1 | ON}} (see page 279)	:DEMO:OUTPut? (see page 279)	{0 | 1}

Table 9 :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital<d>:DISPlay {{0 | OFF} | {1 | ON}} (see page 283)	:DIGital<d>:DISPlay? (see page 283)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 | 1}
:DIGital<d>:LABel <string> (see page 284)	:DIGital<d>:LABel? (see page 284)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGital<d>:POSition <position> (see page 285)	:DIGital<d>:POSition? (see page 285)	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.

Table 9 :DIGital<d> Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGital<d>:SIZE <value> (see page 286)	:DIGital<d>:SIZE? (see page 286)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl | MEDium | LARGe}
:DIGital<d>:THReshold <value>[suffix] (see page 287)	:DIGital<d>:THReshold? (see page 287)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS | ECL | TTL | <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V | mV | uV}

Table 10 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation {{0 | OFF} | {1 | ON}} (see page 291)	:DISPlay:ANNotation? (see page 291)	{0 | 1}
:DISPlay:ANNotation:BACKground <mode> (see page 292)	:DISPlay:ANNotation:BACKground? (see page 292)	<mode> ::= {OPAQue | INVerted | TRANsparent}
:DISPlay:ANNotation:COLor <color> (see page 293)	:DISPlay:ANNotation:COLor? (see page 293)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED}
:DISPlay:ANNotation:TEXT <string> (see page 294)	:DISPlay:ANNotation:TEXT? (see page 294)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 295)	n/a	n/a
n/a	:DISPlay:DATA? [<format>][,][<palette>] (see page 296)	<format> ::= {BMP | BMP8bit | PNG} <palette> ::= {COLor | GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 297)	:DISPlay:LABel? (see page 297)	{0 | 1}
:DISPlay:LABList <binary block> (see page 298)	:DISPlay:LABList? (see page 298)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters

Table 10 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:PERSistence <value> (see page 299)	:DISPlay:PERSistence? (see page 299)	<value> ::= {MINimum | INFinite | <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1 | ON} (see page 300)	:DISPlay:VECTors? (see page 300)	1

Table 11 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 302)	:EXTernal:BWLimit? (see page 302)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 303)	:EXTernal:PROBe? (see page 303)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 304)	:EXTernal:RANGe? (see page 304)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 305)	:EXTernal:UNITs? (see page 305)	<units> ::= {VOLT | AMPere}

Table 12 :FUNCtion Commands Summary

Command	Query	Options and Query Returns
:FUNCtion:BUS:CLOCk <source> (see page 312)	:FUNCtion:BUS:CLOCk? (see page 312)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:FUNCtion:BUS:SLOPe <slope> (see page 313)	:FUNCtion:BUS:SLOPe? (see page 313)	<slope> ::= {NEGative | POSitive | EITHer}
:FUNCtion:BUS:YINCrement <value> (see page 314)	:FUNCtion:BUS:YINCrement? (see page 314)	<value> ::= value per bus code, in NR3 format

Table 12 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:BUS:YORigin <value> (see page 315)	:FUNCtion:BUS:YORigin? (see page 315)	<value> ::= value at bus code = 0, in NR3 format
:FUNCtion:BUS:YUNits <units> (see page 316)	:FUNCtion:BUS:YUNits? (see page 316)	<units> ::= {VOLT | AMPere | NONE}
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}} (see page 317)	:FUNCtion:DISPlay? (see page 317)	{0 | 1}
:FUNCtion[:FFT]:CENTer <frequency> (see page 318)	:FUNCtion[:FFT]:CENTer? (see page 318)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCtion[:FFT]:SPAN (see page 319)	:FUNCtion[:FFT]:SPAN? (see page 319)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCtion[:FFT]:VTYPe <units> (see page 320)	:FUNCtion[:FFT]:VTYPe? (see page 320)	<units> ::= {DECibel | VRMS}
:FUNCtion[:FFT]:WINDow <window> (see page 321)	:FUNCtion[:FFT]:WINDow? (see page 321)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris}
:FUNCtion:FREQuency:HIGHpass <3dB_freq> (see page 322)	:FUNCtion:FREQuency:HIGHpass? (see page 322)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format
:FUNCtion:FREQuency:LOWPass <3dB_freq> (see page 323)	:FUNCtion:FREQuency:LOWPass? (see page 323)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format
:FUNCtion:GOFT:OPERation <operation> (see page 324)	:FUNCtion:GOFT:OPERation? (see page 324)	<operation> ::= {ADD | SUBTract | MULTiply}
:FUNCtion:GOFT:SOURce1 <source> (see page 325)	:FUNCtion:GOFT:SOURce1? (see page 325)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:FUNCtion:GOFT:SOURce2 <source> (see page 326)	:FUNCtion:GOFT:SOURce2? (see page 326)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models

Table 12 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:INTegrate:IOFFset <input_offset> (see page 327)	:FUNCtion:INTegrate:IOFFset? (see page 3?7)	<input_offset> ::= DC offset correction in NR3 format.
:FUNCtion:LINear:GAIN <value> (see page 328)	:FUNCtion:LINear:GAIN? (see page 328)	<value> ::= 'A' in Ax + B, value in NR3 format
:FUNCtion:LINear:OFFSet <value> (see page 329)	:FUNCtion:LINear:OFFSet? (see page 329)	<value> ::= 'B' in Ax + B, value in NR3 format
:FUNCtion:OFFSet <offset> (see page 330)	:FUNCtion:OFFSet? (see page 330)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:OPERation <operation> (see page 331)	:FUNCtion:OPERation? (see page 332)	<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFF | FFT | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | DIVide | LINear | TRENd | BTIMing | BSTate}
:FUNCtion:RANGe <range> (see page 333)	:FUNCtion:RANGe? (see page 333)	<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.
:FUNCtion:REFerence <level> (see page 334)	:FUNCtion:REFerence? (see page 334)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:SCALe <scale value>[<suffix>] (see page 335)	:FUNCtion:SCALe? (see page 335)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB}

Table 12 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:SOURce1 <source> (see page 336)	:FUNCtion:SOURce1? (see page 336)	<source> ::= {CHANnel<n> | GOFT | BUS<m>} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models <m> ::= {1 | 2} GOFT is only for FFT, INTegrate, DIFF, and SQRT operations.
:FUNCtion:SOURce2 <source> (see page 338)	:FUNCtion:SOURce2? (see page 338)	<source> ::= {CHANnel<n> | NONE} <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:TRENd:MEASurement <type> (see page 339)	:FUNCtion:TRENd:MEASurement? (see page 339)	<type> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDth | NWIDth | DUTYcycle | RISetime | FALLtime}

Table 13 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 343)	:HARDcopy:AREA? (see page 343)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 344)	:HARDcopy:APRinter? (see page 344)	<active_printer> ::= {<index> | <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}} (see page 345)	:HARDcopy:FACTors? (see page 345)	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}} (see page 346)	:HARDcopy:FFEed? (see page 346)	{0 | 1}
:HARDcopy:INKSaver {{0 | OFF} | {1 | ON}} (see page 347)	:HARDcopy:INKSaver? (see page 347)	{0 | 1}
:HARDcopy:LAYout <layout> (see page 348)	:HARDcopy:LAYout? (see page 348)	<layout> ::= {LANDscape | PORTrait}

Table 13 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:NETWork:ADDRess <address> (see page 349)	:HARDcopy:NETWork:ADDRess? (see page 349)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APPLy (see page 350)	n/a	n/a
:HARDcopy:NETWork:DOMain <domain> (see page 351)	:HARDcopy:NETWork:DOMain? (see page 351)	<domain> ::= quoted ASCII string
:HARDcopy:NETWork:PASSword <password> (see page 352)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLOT <slot> (see page 353)	:HARDcopy:NETWork:SLOT? (see page 353)	<slot> ::= {NET0 | NET1}
:HARDcopy:NETWork:USERname <username> (see page 354)	:HARDcopy:NETWork:USERname? (see page 354)	<username> ::= quoted ASCII string
:HARDcopy:PALette <palette> (see page 355)	:HARDcopy:PALette? (see page 355)	<palette> ::= {COLor | GRAYscale | NONE}
n/a	:HARDcopy:PRINter:LIST? (see page 356)	<list> ::= [<printer_spec>] ... [printer_spec>] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y | N} <name> ::= name of printer
:HARDcopy:STARt (see page 357)	n/a	n/a

Table 14 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 360)	<binary_block> ::= comma-separated data with newlines at the end of each row

Table 14 :LISTer Commands Summary (continued)

Command	Query	Options and Query Returns
:LISTer:DISPlay {{OFF | 0} | {SBUS1 | ON | 1} | {SBUS2 | 2} | ALL} (see page 361)	:LISTer:DISPlay? (see page 361)	{OFF | SBUS1 | SBUS2 | ALL}
:LISTer:REFerence <time_ref> (see page 362)	:LISTer:REFerence? (see page 362)	<time_ref> ::= {TRIGger | PREVious}

Table 15 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 365)	:MARKer:MODE? (see page 365)	<mode> ::= {OFF | MEASurement | MANual | WAVeform}
:MARKer:X1Position <position>[suffix] (see page 366)	:MARKer:X1Position? (see page 366)	<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 367)	:MARKer:X1Y1source? (see page 367)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 368)	:MARKer:X2Position? (see page 368)	<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 369)	:MARKer:X2Y2source? (see page 369)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 370)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 371)	:MARKer:XUNits? (see page 371)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}

Table 15 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:XUNits:USE (see page 372)	n/a	n/a
:MARKer:Y1Position <position>[suffix] (see page 373)	:MARKer:Y1Position? (see page 373)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 374)	:MARKer:Y2Position? (see page 374)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 375)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 376)	:MARKer:YUNits? (see page 376)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 377)	n/a	n/a

Table 16 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 391)	n/a	n/a
:MEASure:AREa [<interval>][,][<source>] (see page 392)	:MEASure:AREa? [<interval>][,][<source>] (see page 392)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BWIDth [<source>] (see page 393)	:MEASure:BWIDth? [<source>] (see page 393)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 394)	n/a	n/a

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:COUNter [<source>] (see page 395)	:MEASure:COUNter? [<source>] (see page 395)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 396)	:MEASure:DEFine? DELay (see page 397)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 396)	:MEASure:DEFine? THResholds (see page 397)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent | ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 399)	:MEASure:DELay? [<source1>] [,<source2>] (see page 399)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 401)	:MEASure:DUTYcycle? [<source>] (see page 401)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 402)	:MEASure:FALLtime? [<source>] (see page 402)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 403)	:MEASure:FREQuency? [<source>] (see page 403)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NEDGes [<source>] (see page 404)	:MEASure:NEDGes? [<source>] (see page 404)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the falling edge count in NR3 format
:MEASure:NPULses [<source>] (see page 405)	:MEASure:NPULses? [<source>] (see page 405)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the falling pulse count in NR3 format
:MEASure:NWIDth [<source>] (see page 406)	:MEASure:NWIDth? [<source>] (see page 406)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 407)	:MEASure:OVERshoot? [<source>] (see page 407)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PEDGes [<source>] (see page 409)	:MEASure:PEDGes? [<source>] (see page 409)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the rising edge count in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PERiod [<source>] (see page 410)	:MEASure:PERiod? [<source>] (see page 410)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 411)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 411)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PPULses [<source>] (see page 412)	:MEASure:PPULses? [<source>] (see page 412)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the rising pulse count in NR3 format
:MEASure:PREShoot [<source>] (see page 413)	:MEASure:PREShoot? [<source>] (see page 413)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PWIDth [<source>] (see page 414)	:MEASure:PWIDth? [<source>] (see page 414)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 415)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 418)	:MEASure:RISetime? [<source>] (see page 418)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 419)	:MEASure:SDEViation? [<source>] (see page 419)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1 | ON} (see page 420)	:MEASure:SHOW? (see page 420)	{1}

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SOURce <source1> [,<source2>] (see page 421)	:MEASure:SOURce? (see page 421)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r> | EXTernal} for DSO models <source1,2> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r> | EXTernal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 423)	:MEASure:STATistics? (see page 423)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 424)	:MEASure:STATistics:DISPlay? (see page 424)	{0 | 1}
:MEASure:STATistics:INCRement (see page 425)	n/a	n/a
:MEASure:STATistics:MCOunt <setting> (see page 426)	:MEASure:STATistics:MCOunt? (see page 426)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:RESet (see page 427)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 428)	:MEASure:STATistics:RSDeviation? (see page 428)	{0 | 1}

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TEDGe? <slope><occurrence>[,<source>] (see page 429)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>,[<slope>]<occurrence>[,<source>] (see page 431)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VAMPlitude [<source>] (see page 433)	:MEASure:VAMPlitude? [<source>] (see page 433)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<interval>][,][<source>] (see page 434)	:MEASure:VAVerage? [<interval>][,][<source>] (see page 434)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 435)	:MEASure:VBASe? [<source>] (see page 435)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 436)	:MEASure:VMAX? [<source>] (see page 436)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 437)	:MEASure:VMIN? [<source>] (see page 437)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VPP [<source>] (see page 438)	:MEASure:VPP? [<source>] (see page 438)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<interval>][,][<source1>] [,<source2>] (see page 439)	:MEASure:VRATio? [<interval>][,][<source1>] [,<source2>] (see page 439)	<interval> ::= {CYCLe | DISPlay} <source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 440)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 440)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 441)	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

Table 16 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 442)	:MEASure:VTOP? [<source>] (see page 442)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 443)	:MEASure:WINDow? (see page 443)	<type> ::= {MAIN | ZOOM | AUTO}
:MEASure:XMAX [<source>] (see page 444)	:MEASure:XMAX? [<source>] (see page 444)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 445)	:MEASure:XMIN? [<source>] (see page 445)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Table 17 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 472)	:MTESt:ALL? (see page 472)	{0 | 1}
:MTESt:AMASk:CREate (see page 473)	n/a	n/a
:MTESt:AMASk:SOURce <source> (see page 474)	:MTESt:AMASk:SOURce? (see page 474)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 475)	:MTESt:AMASk:UNITs? (see page 475)	<units> ::= {CURRent | DIVisions}

Table 17 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:AMASk:XDELta <value> (see page 476)	:MTESt:AMASk:XDELta? (see page 476)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 477)	:MTESt:AMASk:YDELta? (see page 477)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 478)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 479)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 480)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 481)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 482)	:MTESt:DATA? (see page 482)	<mask> ::= data in IEEE 488.2 # format.
:MTESt:DELete (see page 483)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 484)	:MTESt:ENABle? (see page 484)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 485)	:MTESt:LOCK? (see page 485)	{0 | 1}
:MTESt:RMODe <rmode> (see page 486)	:MTESt:RMODe? (see page 486)	<rmode> ::= {FOREver | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 487)	:MTESt:RMODe:FACTion:MEASure? (see page 487)	{0 | 1}
:MTESt:RMODe:FACTion:PRINt {{0 | OFF} | {1 | ON}} (see page 488)	:MTESt:RMODe:FACTion:PRINt? (see page 488)	{0 | 1}
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 489)	:MTESt:RMODe:FACTion:SAVE? (see page 489)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 490)	:MTESt:RMODe:FACTion:STOP? (see page 490)	{0 | 1}

Table 17 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:RMODe:SIGMa <level> (see page 491)	:MTESt:RMODe:SIGMa? (see page 491)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see page 492)	:MTESt:RMODe:TIME? (see page 492)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 493)	:MTESt:RMODe:WAVeforms? (see page 493)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 494)	:MTESt:SCALe:BIND? (see page 494)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 495)	:MTESt:SCALe:X1? (see page 495)	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALe:XDELta <xdelta_value> (see page 496)	:MTESt:SCALe:XDELta? (see page 496)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 497)	:MTESt:SCALe:Y1? (see page 497)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 498)	:MTESt:SCALe:Y2? (see page 498)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 499)	:MTESt:SOURce? (see page 499)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 500)	<title> ::= a string of up to 128 ASCII characters

Table 18 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 503)	:POD<n>:DISPlay? (see page 503)	{0 | 1} <n> ::= 1-2 in NR1 format

Table 18 :POD<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:POD<n>:SIZE <value> (see page 504)	:POD<n>:SIZE? (see page 504)	<value> ::= {SMALl | MEDium | LARGe}
:POD<n>:THReshold <type>[suffix] (see page 505)	:POD<n>:THReshold? (see page 505)	<n> ::= 1-2 in NR1 format <type> ::= {CMOS | ECL | TTL | <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V | mV | uV }

Table 19 :POWer Commands Summary

Command	Query	Options and Query Returns
:POWer:DESKew (see page 511)	n/a	n/a
:POWer:EFFiciency:APPLy (see page 512)	n/a	n/a
:POWer:ENABle {{0 | OFF} | {1 | ON}} (see page 513)	:POWer:ENABle? (see page 513)	{0 | 1}
:POWer:HARMonics:APPLy (see page 514)	n/a	n/a
n/a	:POWer:HARMonics:DATA? (see page 515)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISPlay <display> (see page 516)	:POWer:HARMonics:DISPlay? (see page 516)	<display> ::= {TABLe | BAR | OFF}
n/a	:POWer:HARMonics:FAILcount? (see page 517)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 518)	:POWer:HARMonics:LINE? (see page 518)	<frequency> ::= {F50 | F60 | F400}
n/a	:POWer:HARMonics:POWerfactor? (see page 519)	<value> ::= Class C power factor in NR3 format
n/a	:POWer:HARMonics:RUNCount? (see page 520)	<count> ::= integer in NR1 format
:POWer:HARMonics:STANdard <class> (see page 521)	:POWer:HARMonics:STANdard? (see page 521)	<class> ::= {A | B | C | D}

Table 19 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:POWer:HARMonics:STATus? (see page 522)	<status> ::= {PASS | FAIL | UNTested}
n/a	:POWer:HARMonics:THD? (see page 523)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLy (see page 524)	n/a	n/a
:POWer:INRush:EXIT (see page 525)	n/a	n/a
:POWer:INRush:NEXT (see page 526)	n/a	n/a
:POWer:MODulation:APPLy (see page 527)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 528)	:POWer:MODulation:SOURce? (see page 528)	<source> ::= {V | I}
:POWer:MODulation:TYPE <modulation> (see page 529)	:POWer:MODulation:TYPE? (see page 529)	<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDith | NWIDth | DUTYcycle | RISetime | FALLtime}
:POWer:ONOFf:APPLy (see page 530)	n/a	n/a
:POWer:ONOFf:EXIT (see page 531)	n/a	n/a
:POWer:ONOFf:NEXT (see page 532)	n/a	n/a
:POWer:ONOFf:TEST {{0 | OFF} | {1 | ON}} (see page 533)	:POWer:ONOFf:TEST? (see page 533)	{0 | 1}
:POWer:PSRR:APPLy (see page 534)	n/a	n/a
:POWer:PSRR:FREQuency:MAXimum <value>[suffix] (see page 535)	:POWer:PSRR:FREQuency:MAXimum? (see page 535)	<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:FREQuency:MINimum <value>[suffix] (see page 536)	:POWer:PSRR:FREQuency:MINimum? (see page 536)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}

Table 19 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:RMAXimum <value> (see page 537)	:POWer:PSRR:RMAXimum? (see page 537)	<value> ::= Maximum ratio value in NR1 format
:POWer:QUALity:APPLy (see page 538)	n/a	n/a
:POWer:QUALity:TYPE <quality> (see page 539)	:POWer:QUALity:TYPE? (see page 539)	<quality> ::= {FACTor | REAL | APParent | REACtive | CRESt | ANGLe}
:POWer:RIPPle:APPLy (see page 540)	n/a	n/a
:POWer:SIGNals:AUTosetup <analysis> (see page 542)	n/a	<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity | SLEW | SWITch}
:POWer:SIGNals:CYCLes <count> (see page 542)	:POWer:SIGNals:CYCLes? (see page 542)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURation <value>[suffix] (see page 543)	:POWer:SIGNals:DURation? (see page 543)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:IEXPected <value>[suffix] (see page 544)	:POWer:SIGNals:IEXPected? (see page 544)	<value> ::= Expected current value in NR3 format [suffix] ::= {A | mA}
:POWer:SIGNals:OVERshoot <percent> (see page 545)	:POWer:SIGNals:OVERshoot? (see page 545)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V | mV}}
:POWer:SIGNals:VMAXimum <value>[suffix] (see page 546)	:POWer:SIGNals:VMAXimum? (see page 546)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady <value>[suffix] (see page 547)	:POWer:SIGNals:VSTeady? (see page 547)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:SOURce:CURRent<i> <source> (see page 548)	:POWer:SIGNals:SOURce:CURRent<i>? (see page 548)	<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 549)	:POWer:SIGNals:SOURce:VOLTage<i>? (see page 549)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 19 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SLEW:APPLy (see page 550)	n/a	n/a
:POWer:SLEW:SOURce <source> (see page 551)	:POWer:SLEW:SOURce? (see page 551)	<source> ::= {V | I}
n/a	:POWer:SLEW:VALue? (see page 552)	<value> ::= slew rate in NR3 format
:POWer:SWITch:APPLy (see page 553)	n/a	n/a
:POWer:SWITch:CONDuction <conduction> (see page 554)	:POWer:SWITch:CONDuction? (see page 554)	<conduction> ::= {WAVeform | RDS | VCE}
:POWer:SWITch:IREFerence <percent> (see page 555)	:POWer:SWITch:IREFerence? (see page 555)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 556)	:POWer:SWITch:RDS? (see page 556)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM | mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 557)	:POWer:SWITch:VCE? (see page 557)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V | mV}
:POWer:SWITch:VREFerence <percent> (see page 558)	:POWer:SWITch:VREFerence? (see page 558)	<percent> ::= percent in NR1 format
:POWer:TRANsient:APPLy (see page 559)	n/a	n/a
:POWer:TRANsient:EXIT (see page 560)	n/a	n/a
:POWer:TRANsient:IINitial <value>[suffix] (see page 561)	:POWer:TRANsient:IINitial? (see page 561)	<value> ::= Initial current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:INEW <value>[suffix] (see page 562)	:POWer:TRANsient:INEW? (see page 562)	<value> ::= New current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:NEXT (see page 563)	n/a	n/a

Table 20 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[STARt] [<file_spec>][,<column>] (see page 567)	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
:RECall:FILename <base_name> (see page 568)	:RECall:FILename? (see page 568)	<base_name> ::= quoted ASCII string
:RECall:MASK[:STARt] [<file_spec>] (see page 569)	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 570)	:RECall:PWD? (see page 570)	<path_name> ::= quoted ASCII string
:RECall:SETup[:STARt] [<file_spec>] (see page 571)	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>] (see page 572)	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 21 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [<file_spec>] (see page 576)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:FILename <base_name> (see page 577)	:SAVE:FILename? (see page 577)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 578)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 579)	:SAVE:IMAGe:FACTors? (see page 579)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 580)	:SAVE:IMAGe:FORMat? (see page 580)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 581)	:SAVE:IMAGe:INKSaver? (see page 581)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 582)	:SAVE:IMAGe:PALette? (see page 582)	<palette> ::= {COLor | GRAYscale}
:SAVE:LISTer[:STARt] [<file_name>] (see page 583)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:STARt] [<file_spec>] (see page 584)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:POWer[:STARt] [<file_name>] (see page 585)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 586)	:SAVE:PWD? (see page 586)	<path_name> ::= quoted ASCII string

Table 21 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETup[:STARt] [<file_spec>] (see page 587)	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 588)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 589)	:SAVE:WAVeform:FORMat? (see page 589)	<format> ::= {ALB | ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 590)	:SAVE:WAVeform:LENGth? (see page 590)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 591)	:SAVE:WAVeform:LENGth:MAX? (see page 591)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 592)	:SAVE:WAVeform:SEGMented? (see page 592)	<option> ::= {ALL | CURRent}
:SAVE:WMEMory:SOURce <source> (see page 593)	:SAVE:WMEMory:SOURce? (see page 593)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 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 594)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 22 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 598)	:SBUS<n>:DISPlay? (see page 598)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 599)	:SBUS<n>:MODE? (see page 599)	<mode> ::= {A429 | CAN | FLEXray | I2S | IIC | LIN | M1553 | SPI | UART}

Table 23 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUTosetup (see page 602)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 603)	:SBUS<n>:A429:BASE? (see page 603)	<base> ::= {BINary | HEX}
n/a	:SBUS<n>:A429:COUNt:ERRor? (see page 604)	<error_count> ::= integer in NR1 format
:SBUS<n>:A429:COUNt:RESet (see page 605)	n/a	n/a
n/a	:SBUS<n>:A429:COUNt:WORD? (see page 606)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMat <format> (see page 607)	:SBUS<n>:A429:FORMat? (see page 607)	<format> ::= {LDSDi | LDSSm | LDATa}
:SBUS<n>:A429:SIGNal <signal> (see page 608)	:SBUS<n>:A429:SIGNal? (see page 608)	<signal> ::= {A | B | DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 609)	:SBUS<n>:A429:SOURce? (see page 609)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEed <speed> (see page 610)	:SBUS<n>:A429:SPEed? (see page 610)	<speed> ::= {LOW | HIGH}

Table 23 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :LABel <value> (see page 611)	:SBUS<n>:A429:TRIGger :LABel? (see page 611)	<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 612)	:SBUS<n>:A429:TRIGger :PATTern:DATA? (see page 612)	<string> ::= "nn...n" where n ::= {0 | 1 | X}, length depends on FORMat
:SBUS<n>:A429:TRIGger :PATTern:SDI <string> (see page 613)	:SBUS<n>:A429:TRIGger :PATTern:SDI? (see page 613)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :PATTern:SSM <string> (see page 614)	:SBUS<n>:A429:TRIGger :PATTern:SSM? (see page 614)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :RANGe <min>,<max> (see page 615)	:SBUS<n>:A429:TRIGger :RANGe? (see page 615)	<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 616)	:SBUS<n>:A429:TRIGger :TYPE? (see page 616)	<condition> ::= {WSTArt | WSTOp | LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors | LRANge | ABITs | AOBits | AZBits}

Table 24 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 620)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 621)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 622)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 623)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 624)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:SAMPlepoint <value> (see page 625)	:SBUS<n>:CAN:SAMPlepoint? (see page 625)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 626)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 626)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 627)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 627)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SOURce <source> (see page 628)	:SBUS<n>:CAN:SOURce? (see page 628)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> |} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 629)	:SBUS<n>:CAN:TRIGger? (see page 630)	<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote | ALLerrors | OVERload | ACKerror}
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 631)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 631)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 632)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 632)	<length> ::= integer from 1 to 8 in NR1 format

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

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 633)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 633)	<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 634)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 634)	<value> ::= {STANdard | EXTended}

Table 25 :SBUS<n>:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:AUTosetup (see page 637)	n/a	n/a
:SBUS<n>:FLEXray:BAUDrate <baudrate> (see page 638)	:SBUS<n>:FLEXray:BAUDrate? (see page 638)	<baudrate> ::= {2500000 | 5000000 | 10000000}
:SBUS<n>:FLEXray:CHANnel <channel> (see page 639)	:SBUS<n>:FLEXray:CHANnel? (see page 639)	<channel> ::= {A | B}
n/a	:SBUS<n>:FLEXray:COUNt:NULL? (see page 640)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:COUNt:RESet (see page 641)	n/a	n/a
n/a	:SBUS<n>:FLEXray:COUNt:SYNC? (see page 642)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:FLEXray:COUNt:TOTal? (see page 643)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:SOURce <source> (see page 644)	:SBUS<n>:FLEXray:SOURce? (see page 644)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format
:SBUS<n>:FLEXray:TRIGger <condition> (see page 645)	:SBUS<n>:FLEXray:TRIGger? (see page 645)	<condition> ::= {FRAMe | ERRor | EVENt}

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

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE <error_type> (see page 646)	:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE? (see page 646)	<error_type> ::= {ALL | HCRC | FCRC}
:SBUS<n>:FLEXray:TRIGger:EVENt:AUToset (see page 647)	n/a	n/a
:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID <frame_id> (see page 648)	:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID? (see page 648)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE <event> (see page 649)	:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE? (see page 649)	<event> ::= {WAKeup | TSS | {FES | DTS} | BSS}
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase <cycle_count_base> (see page 650)	:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase? (see page 650)	<cycle_count_base> ::= integer from 0-63
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition <cycle_count_repetition> (see page 651)	:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition? (see page 651)	<cycle_count_repetition> ::= {ALL | <rep #>} <rep #> ::= integer values 2, 4, 8, 16, 32, or 64
:SBUS<n>:FLEXray:TRIGger:FRAMe:ID <frame_id> (see page 652)	:SBUS<n>:FLEXray:TRIGger:FRAMe:ID? (see page 652)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE <frame_type> (see page 653)	:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE? (see page 653)	<frame_type> ::= {NORMal | STARtup | NULL | SYNC | NSTArtup | NNULl | NSYNc | ALL}

Table 26 :SBUS<n>:I2S Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:I2S:ALIGnment <setting> (see page 656)	:SBUS<n>:I2S:ALIGnment? (see page 656)	<setting> ::= {I2S | LJ | RJ}
:SBUS<n>:I2S:BASE <base> (see page 657)	:SBUS<n>:I2S:BASE? (see page 657)	<base> ::= {DECimal | HEX}

Table 26 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:CLOCk:SLOPe <slope> (see page 658)	:SBUS<n>:I2S:CLOCk:SLOPe? (see page 658)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:I2S:RWIDth <receiver> (see page 659)	:SBUS<n>:I2S:RWIDth? (see page 659)	<receiver> ::= 4-32 in NR1 format
:SBUS<n>:I2S:SOURce:CLOCk <source> (see page 660)	:SBUS<n>:I2S:SOURce:CLOCk? (see page 660)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:I2S:SOURce:DATA <source> (see page 661)	:SBUS<n>:I2S:SOURce:DATA? (see page 661)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:I2S:SOURce:WSELect <source> (see page 662)	:SBUS<n>:I2S:SOURce:WSELect? (see page 662)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:I2S:TRIGger <operator> (see page 663)	:SBUS<n>:I2S:TRIGger? (see page 663)	<operator> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange | OUTRange | INCReasing | DECReasing}
:SBUS<n>:I2S:TRIGger:AUDio <audio_ch> (see page 665)	:SBUS<n>:I2S:TRIGger:AUDio? (see page 665)	<audio_ch> ::= {RIGHt | LEFT | EITHer}

Table 26 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:TRIGger:PATTern:DATA <string> (see page 666)	:SBUS<n>:I2S:TRIGger:PATTern:DATA? (see page 667)	<string> ::= "n" where n ::= 32-bit integer in signed 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>:I2S:TRIGger:PATTern:FORMat <base> (see page 668)	:SBUS<n>:I2S:TRIGger:PATTern:FORMat? (see page 668)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:I2S:TRIGger:RANGe <lower>,<upper> (see page 669)	:SBUS<n>:I2S:TRIGger:RANGe? (see page 669)	<lower> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SBUS<n>:I2S:TWIDth <word_size> (see page 671)	:SBUS<n>:I2S:TWIDth? (see page 671)	<word_size> ::= 4-32 in NR1 format
:SBUS<n>:I2S:WSLow <low_def> (see page 672)	:SBUS<n>:I2S:WSLow? (see page 672)	<low_def> ::= {LEFT | RIGHt}

Table 27 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 674)	:SBUS<n>:IIC:ASIZe? (see page 674)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce]:CLOCk <source> (see page 675)	:SBUS<n>:IIC[:SOURce]:CLOCk? (see page 675)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC[:SOURce]:DATA <source> (see page 676)	:SBUS<n>:IIC[:SOURce]:DATA? (see page 676)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC:TRIGger:PATTern:ADDRess <value> (see page 677)	:SBUS<n>:IIC:TRIGger:PATTern:ADDRess? (see page 677)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATA <value> (see page 678)	:SBUS<n>:IIC:TRIGger:PATTern:DATA? (see page 678)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:PATTern:DATa2 <value> (see page 679)	:SBUS<n>:IIC:TRIGger:PATTern:DATa2? (see page 679)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger:QUALifier <value> (see page 680)	:SBUS<n>:IIC:TRIGger:QUALifier? (see page 680)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 681)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 681)	<type> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10 | NACKnowledge | ANACk | R7Data2 | W7Data2 | RESTart}

Table 28 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 685)	:SBUS<n>:LIN:PARity? (see page 685)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 686)	:SBUS<n>:LIN:SAMPlepoint? (see page 686)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 687)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 687)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 688)	:SBUS<n>:LIN:SOURce? (see page 688)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 689)	:SBUS<n>:LIN:STANdard? (see page 689)	<std> ::= {LIN13 | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 690)	:SBUS<n>:LIN:SYNCbreak? (see page 690)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 691)	:SBUS<n>:LIN:TRIGger? (see page 691)	<condition> ::= {SYNCbreak | ID | DATA}
:SBUS<n>:LIN:TRIGger:ID <value> (see page 692)	:SBUS<n>:LIN:TRIGger:ID? (see page 692)	<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 28 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 693)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 693)	<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 695)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 695)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 696)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 696)	<base> ::= {BINary | HEX | DECimal}

Table 29 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTosetup (see page 698)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 699)	:SBUS<n>:M1553:BASE? (see page 699)	<base> ::= {BINary | HEX}
:SBUS<n>:M1553:SOURce <source> (see page 700)	:SBUS<n>:M1553:SOURce? (see page 700)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGger:PATTern:DATA <string> (see page 701)	:SBUS<n>:M1553:TRIGger:PATTern:DATA? (see page 701)	<string> ::= "nn...n" where n ::= {0 | 1 | X}
:SBUS<n>:M1553:TRIGger:RTA <value> (see page 702)	:SBUS<n>:M1553:TRIGger:RTA? (see page 702)	<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 703)	:SBUS<n>:M1553:TRIGger:TYPE? (see page 703)	<type> ::= {DSTArt | DSTOp | CSTArt | CSTOp | RTA | PERRor | SERRor | MERRor | RTA11}

Table 30 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 706)	:SBUS<n>:SPI:BITorder? (see page 706)	<order> ::= {LSBFirst | MSBFirst}
:SBUS<n>:SPI:CLOCk:SLOPe <slope> (see page 707)	:SBUS<n>:SPI:CLOCk:SLOPe? (see page 707)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:SPI:CLOCk:TIMeout <time_value> (see page 708)	:SBUS<n>:SPI:CLOCk:TIMeout? (see page 708)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 709)	:SBUS<n>:SPI:FRAMing? (see page 709)	<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
:SBUS<n>:SPI:SOURce:CLOCk <source> (see page 710)	:SBUS<n>:SPI:SOURce:CLOCk? (see page 710)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:DATA <source> (see page 711)	:SBUS<n>:SPI:SOURce:DATA? (see page 711)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:FRAMe <source> (see page 712)	:SBUS<n>:SPI:SOURce:FRAMe? (see page 712)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

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

Command	Query	Options and Query Returns
:SBUS<n>:SPI:SOURce:MISO <source> (see page 713)	:SBUS<n>:SPI:SOURce:MISO? (see page 713)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:MOSI <source> (see page 714)	:SBUS<n>:SPI:SOURce:MOSI? (see page 714)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string> (see page 715)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 715)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width> (see page 716)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? (see page 716)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string> (see page 717)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? (see page 717)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width> (see page 718)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? (see page 718)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 719)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 719)	<value> ::= {MOSI | MISO}
:SBUS<n>:SPI:WIDTh <word_width> (see page 720)	:SBUS<n>:SPI:WIDTh? (see page 720)	<word_width> ::= integer 4-16 in NR1 format

Table 31 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 724)	:SBUS<n>:UART:BASE? (see page 724)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 725)	:SBUS<n>:UART:BAUDrate? (see page 725)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorder <bitorder> (see page 726)	:SBUS<n>:UART:BITorder? (see page 726)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 727)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 728)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 729)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 730)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 731)	:SBUS<n>:UART:FRAMing? (see page 731)	<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 732)	:SBUS<n>:UART:PARity? (see page 732)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 733)	:SBUS<n>:UART:POLarity? (see page 733)	<polarity> ::= {HIGH | LOW}
:SBUS<n>:UART:SOURce:RX <source> (see page 734)	:SBUS<n>:UART:SOURce:RX? (see page 734)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

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

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce:TX <source> (see page 735)	:SBUS<n>:UART:SOURce:TX? (see page 735)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 736)	:SBUS<n>:UART:TRIGger:BASE? (see page 736)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 737)	:SBUS<n>:UART:TRIGger:BURSt? (see page 737)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger:DATA <value> (see page 738)	:SBUS<n>:UART:TRIGger:DATA? (see page 738)	<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 739)	:SBUS<n>:UART:TRIGger:IDLE? (see page 739)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger:QUALifier <value> (see page 740)	:SBUS<n>:UART:TRIGger:QUALifier? (see page 740)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:SBUS<n>:UART:TRIGger:TYPE <value> (see page 741)	:SBUS<n>:UART:TRIGger:TYPE? (see page 741)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 742)	:SBUS<n>:UART:WIDTh? (see page 742)	<width> ::= {5 | 6 | 7 | 8 | 9}

Table 32 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 745)	<count> ::= an integer count value
:SEARch:MODE <value> (see page 746)	:SEARch:MODE? (see page 746)	<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2}}
:SEARch:STATe <value> (see page 747)	:SEARch:STATe? (see page 747)	<value> ::= {{0 | OFF} | {1 | ON}}

Table 33 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 749)	:SEARch:EDGE:SLOPe? (see page 749)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 750)	:SEARch:EDGE:SOURce? (see page 750)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 34 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 752)	:SEARch:GLITch:GREaterthan? (see page 752)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 753)	:SEARch:GLITch:LESSthan? (see page 753)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 754)	:SEARch:GLITch:POLarity? (see page 754)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 755)	:SEARch:GLITch:QUALifier? (see page 755)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}

Table 34 :SEARch:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 756)	:SEARch:GLITch:RANGe? (see page 756)	<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 757)	:SEARch:GLITch:SOURce? (see page 757)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 35 :SEARch:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARch:RUNT:POLarity <polarity> (see page 759)	:SEARch:RUNT:POLarity? (see page 759)	<polarity> ::= {POSitive | NEGative | EITHer}
:SEARch:RUNT:QUALifier <qualifier> (see page 760)	:SEARch:RUNT:QUALifier? (see page 760)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:SEARch:RUNT:SOURce <source> (see page 761)	:SEARch:RUNT:SOURce? (see page 761)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:RUNT:TIME <time>[suffix] (see page 762)	:SEARch:RUNT:TIME? (see page 762)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 36 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 764)	:SEARch:TRANsition:QUALifier? (see page 764)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 765)	:SEARch:TRANsition:SLOPe? (see page 765)	<slope> ::= {NEGative | POSitive}

Table 36 :SEARch:TRANsition Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:TRANsition:SOURce <source> (see page 766)	:SEARch:TRANsition:SOURce? (see page 766)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 767)	:SEARch:TRANsition:TIME? (see page 767)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 37 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:LABel <value> (see page 769)	:SEARch:SERial:A429:LABel? (see page 769)	<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 770)	:SEARch:SERial:A429:MODE? (see page 770)	<condition> ::= {LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors}
:SEARch:SERial:A429:PATTern:DATA <string> (see page 771)	:SEARch:SERial:A429:PATTern:DATA? (see page 771)	<string> ::= "nn...n" where n ::= {0 | 1}, length depends on FORMat
:SEARch:SERial:A429:PATTern:SDI <string> (see page 772)	:SEARch:SERial:A429:PATTern:SDI? (see page 772)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
:SEARch:SERial:A429:PATTern:SSM <string> (see page 773)	:SEARch:SERial:A429:PATTern:SSM? (see page 773)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits

Table 38 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 775)	:SEARch:SERial:CAN:MODE? (see page 775)	<value> ::= {DATA | IDData | IDEither | IDRemote | ALLerrors | OVERload | ERRor}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 776)	:SEARch:SERial:CAN:PATTern:DATA? (see page 776)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 777)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 777)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 778)	:SEARch:SERial:CAN:PATTern:ID? (see page 778)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 779)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 779)	<value> ::= {STANdard | EXTended}

Table 39 :SEARch:SERial:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:FLEXray:CYCLe <cycle> (see page 781)	:SEARch:SERial:FLEXray:CYCLe? (see page 781)	<cycle> ::= {ALL | <cycle #>} <cycle #> ::= integer from 0-63
:SEARch:SERial:FLEXray:DATA <string> (see page 782)	:SEARch:SERial:FLEXray:DATA? (see page 782)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
:SEARch:SERial:FLEXray:DATA:LENGth <length> (see page 783)	:SEARch:SERial:FLEXray:DATA:LENGth? (see page 783)	<length> ::= integer from 1 to 12 in NR1 format
:SEARch:SERial:FLEXray:FRAMe <frame id> (see page 784)	:SEARch:SERial:FLEXray:FRAMe? (see page 784)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SEARch:SERial:FLEXray:MODE <value> (see page 785)	:SEARch:SERial:FLEXray:MODE? (see page 785)	<value> := {FRAMe | CYCLe | DATA | HERRor | FERRor | AERRor}

Table 40 :SEARch:SERial:I2S Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:I2S:AUDio <audio_ch> (see page 787)	:SEARch:SERial:I2S:AUDio? (see page 787)	<audio_ch> ::= {RIGHt | LEFT | EITHer}
:SEARch:SERial:I2S:MODE <value> (see page 788)	:SEARch:SERial:I2S:MODE? (see page 788)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange | OUTRange}
:SEARch:SERial:I2S:PATTern:DATA <string> (see page 789)	:SEARch:SERial:I2S:PATTern:DATA? (see page 789)	<string> ::= "n" where n ::= 32-bit integer in signed 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
:SEARch:SERial:I2S:PATTern:FORMat <base> (see page 790)	:SEARch:SERial:I2S:PATTern:FORMat? (see page 790)	<base> ::= {BINary | HEX | DECimal}
:SEARch:SERial:I2S:RANGe <lower>, <upper> (see page 791)	:SEARch:SERial:I2S:RANGe? (see page 791)	<lower> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal

Table 41 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 793)	:SEARch:SERial:IIC:MODE? (see page 793)	<value> ::= { READ7 | WRITE7 | NACKnowledge | ANACk | R7Data2 | W7Data2 | RESTart | READEprom}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 795)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 795)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

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

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 796)	:SEARch:SERial:IIC:PATTern:DATA? (see page 796)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 797)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 797)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:QUALifier <value> (see page 798)	:SEARch:SERial:IIC:QUALifier? (see page 798)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

Table 42 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 800)	:SEARch:SERial:LIN:ID? (see page 800)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SEARch:SERial:LIN:MODE <value> (see page 801)	:SEARch:SERial:LIN:MODE? (see page 801)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 802)	:SEARch:SERial:LIN:PATTern:DATA? (see page 802)	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}

Table 42 :SEARch:SERial:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:PATTern:DATA:LENGth <length> (see page 803)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 803)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 804)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 804)	<base> ::= {HEX | DECimal}

Table 43 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553:MODE <value> (see page 806)	:SEARch:SERial:M1553:MODE? (see page 806)	<value> ::= {DSTArt | CSTArt | RTA | RTA11 | PERRor | SERRor | MERRor}
:SEARch:SERial:M1553:PATTern:DATA <string> (see page 807)	:SEARch:SERial:M1553:PATTern:DATA? (see page 807)	<string> ::= "nn...n" where n ::= {0 | 1}
:SEARch:SERial:M1553:RTA <value> (see page 808)	:SEARch:SERial:M1553:RTA? (see page 808)	<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 44 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MODE <value> (see page 810)	:SEARch:SERial:SPI:MODE? (see page 810)	<value> ::= {MOSI | MISO}
:SEARch:SERial:SPI:PATTern:DATA <string> (see page 811)	:SEARch:SERial:SPI:PATTern:DATA? (see page 811)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:SPI:PATTern:WIDTh <width> (see page 812)	:SEARch:SERial:SPI:PATTern:WIDTh? (see page 812)	<width> ::= integer from 1 to 10

Table 45 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 814)	:SEARch:SERial:UART:DATA? (see page 814)	<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 815)	:SEARch:SERial:UART:MODE? (see page 815)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | PARityerror | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 816)	:SEARch:SERial:UART:QUALifier? (see page 816)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

Table 46 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 819)	:SYSTem:DATE? (see page 819)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy | AUGust | SEPtember | OCTober | NOVember | DECember} <day> ::= {1,..31}
:SYSTem:DSP <string> (see page 820)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 821)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1047).
:SYSTem:LOCK <value> (see page 822)	:SYSTem:LOCK? (see page 822)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:MENU <menu> (see page 823)	n/a	<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}

Table 46 :SYSTem Commands Summary (continued)

Command	Query	Options and Query Returns
:SYSTem:PRESet (see page 824)	n/a	See :SYSTem:PRESet (see page 824)
:SYSTem:PROTection:LOCK <value> (see page 827)	:SYSTem:PROTection:LOCK? (see page 827)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:SETup <setup_data> (see page 828)	:SYSTem:SETup? (see page 828)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 830)	:SYSTem:TIME? (see page 830)	<time> ::= hours,minutes,seconds in NR1 format

Table 47 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 833)	:TIMebase:MODE? (see page 833)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 834)	:TIMebase:POSition? (see page 834)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 835)	:TIMebase:RANGe? (see page 835)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase:REFerence {LEFT | CENTer | RIGHt} (see page 836)	:TIMebase:REFerence? (see page 836)	<return_value> ::= {LEFT | CENTer | RIGHt}
:TIMebase:SCALe <scale_value> (see page 837)	:TIMebase:SCALe? (see page 837)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 838)	:TIMebase:VERNier? (see page 838)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 839)	:TIMebase:WINDow:POSition? (see page 839)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format

Table 47 :TIMebase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMebase:WINDow:RANGe <range_value> (see page 840)	:TIMebase:WINDow:RANGe? (see page 840)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 841)	:TIMebase:WINDow:SCALe? (see page 841)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Table 48 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 846)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 847)	:TRIGger:HFReject? (see page 847)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 848)	:TRIGger:HOLDoff? (see page 848)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:HIGH <level>, <source> (see page 849)	:TRIGger:LEVel:HIGH? <source> (see page 849)	<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 850)	:TRIGger:LEVel:LOW? <source> (see page 850)	<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 851)	:TRIGger:MODE? (see page 851)	<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2} | USB} <return_value> ::= {<mode> | <none>} <none> ::= query returns "NONE" if the :TIMebase:MODE is ROLL or XY

Table 48 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 852)	:TRIGger:NREJect? (see page 852)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 853)	:TRIGger:SWEep? (see page 853)	<sweep> ::= {AUTO | NORMal}

Table 49 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SLOPe <slope> (see page 855)	:TRIGger:DELay:ARM:SLOPe? (see page 855)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:ARM:SOURce <source> (see page 856)	:TRIGger:DELay:ARM:SOURce? (see page 856)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:DELay:TDELay:TIME <time_value> (see page 857)	:TRIGger:DELay:TDELay:TIME? (see page 857)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGger:COUNt <count> (see page 858)	:TRIGger:DELay:TRIGger:COUNt? (see page 858)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGger:SLOPe <slope> (see page 859)	:TRIGger:DELay:TRIGger:SLOPe? (see page 859)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:TRIGger:SOURce <source> (see page 860)	:TRIGger:DELay:TRIGger:SOURce? (see page 860)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 50 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNt <count> (see page 862)	:TRIGger:EBURst:COUNt? (see page 862)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 863)	:TRIGger:EBURst:IDLE? (see page 863)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 864)	:TRIGger:EBURst:SLOPe? (see page 864)	<slope> ::= {NEGative | POSitive}
:TRIGger:EBURst:SOURce <source> (see page 865)	:TRIGger:EBURst:SOURce? (see page 865)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 51 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 867)	:TRIGger[:EDGE]:COUPling? (see page 867)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 868)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 868)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 51 :TRIGger[:EDGE] Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 869)	:TRIGger[:EDGE]:REJect? (see page 869)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 870)	:TRIGger[:EDGE]:SLOPe? (see page 870)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 871)	:TRIGger[:EDGE]:SOURce? (see page 871)	<source> ::= {CHANnel<n> | EXTernal | LINE | WGEN} for the DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 52 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 874)	:TRIGger:GLITch:GREaterthan? (see page 874)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 875)	:TRIGger:GLITch:LESSthan? (see page 875)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 52 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:LEVel <level> [<source>] (see page 876)	:TRIGger:GLITch:LEVel? (see page 876)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 877)	:TRIGger:GLITch:POLarity? (see page 877)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 878)	:TRIGger:GLITch:QUALifier? (see page 878)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 879)	:TRIGger:GLITch:RANGe? (see page 879)	<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 880)	:TRIGger:GLITch:SOURce? (see page 880)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 53 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 882)	:TRIGger:OR? (see page 882)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog or digital channel as shown on the front panel display.

Table 54 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 884)	:TRIGger:PATTern? (see page 885)	<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} for DSO models <edge_source> ::= {CHANnel<n> | DIGital<d> | NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 886)	:TRIGger:PATTern:FORMat? (see page 886)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 887)	:TRIGger:PATTern:GREaterthan? (see page 887)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 888)	:TRIGger:PATTern:LESSthan? (see page 888)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 54 :TRIGger:PATTern Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:PATTern:QUALifier <qualifier> (see page 889)	:TRIGger:PATTern:QUALifier? (see page 889)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 891)	:TRIGger:PATTern:RANGe? (see page 891)	<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 55 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 893)	:TRIGger:RUNT:POLarity? (see page 893)	<polarity> ::= {POSitive | NEGative | EITHer}
:TRIGger:RUNT:QUALifier <qualifier> (see page 894)	:TRIGger:RUNT:QUALifier? (see page 894)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 895)	:TRIGger:RUNT:SOURce? (see page 895)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 896)	:TRIGger:RUNT:TIME? (see page 896)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 56 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 898)	:TRIGger:SHOLd:SLOPe? (see page 898)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce:CLOCk <source> (see page 899)	:TRIGger:SHOLd:SOURce:CLOCk? (see page 899)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 56 :TRIGger:SHOLd Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SOURce :DATA <source> (see page 900)	:TRIGger:SHOLd:SOURce :DATA? (see page 900)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 901)	:TRIGger:SHOLd:TIME:HOLD? (see page 901)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 902)	:TRIGger:SHOLd:TIME:SETup? (see page 902)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 57 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 904)	:TRIGger:TRANsition:QUALifier? (see page 904)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 905)	:TRIGger:TRANsition:SLOPe? (see page 905)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 906)	:TRIGger:TRANsition:SOURce? (see page 906)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 907)	:TRIGger:TRANsition:TIME? (see page 907)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 58 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 909)	:TRIGger:TV:LINE? (see page 909)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 910)	:TRIGger:TV:MODE? (see page 910)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | LFIeld1 | LFIeld2 | LALTernate}

Table 58 :TRIGger:TV Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:TV:POLarity <polarity> (see page 911)	:TRIGger:TV:POLarity? (see page 911)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 912)	:TRIGger:TV:SOURce? (see page 912)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see page 913)	:TRIGger:TV:STANdard? (see page 913)	<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 914)	:TRIGger:TV:UDTV:ENUMber? (see page 914)	<count> ::= edge number in NR1 format
:TRIGger:TV:UDTV:HSYNc {{0 | OFF} | {1 | ON}} (see page 915)	:TRIGger:TV:UDTV:HSYNc? (see page 915)	{0 | 1}
:TRIGger:TV:UDTV:HTIMe <time> (see page 916)	:TRIGger:TV:UDTV:HTIMe? (see page 916)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTHan <min_time> (see page 917)	:TRIGger:TV:UDTV:PGTHan? (see page 917)	<min_time> ::= seconds in NR3 format

Table 59 :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:DMINus <source> (see page 919)	:TRIGger:USB:SOURce:DMINus? (see page 919)	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:USB:SOURce:DPLus <source> (see page 920)	:TRIGger:USB:SOURce:DPLus? (see page 920)	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:USB:SPEed <value> (see page 921)	:TRIGger:USB:SPEed? (see page 921)	<value> ::= {LOW | FULL}
:TRIGger:USB:TRIGger <value> (see page 922)	:TRIGger:USB:TRIGger? (see page 922)	<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}

Table 60 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 931)	:WAVeform:BYTeorder? (see page 931)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 932)	<count> ::= an integer from 1 to 65536 in NR1 format

Table 60 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:DATA? (see page 933)	<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 935)	:WAVeform:FORMat? (see page 935)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <# points> (see page 936)	:WAVeform:POINts? (see page 936)	<# 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 938)	:WAVeform:POINts:MODE? (see page 938)	<points_mode> ::= {NORMal | MAXimum | RAW}

Table 60 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:PREamble? (see page 940)	<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 943)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGMented:TTAG? (see page 944)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVeform:SOURce <source> (see page 945)	:WAVeform:SOURce? (see page 945)	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models <n> ::= 1 to (# analog channels) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 949)	:WAVeform:SOURce:SUBSource? (see page 949)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
n/a	:WAVeform:TYPE? (see page 950)	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 951)	:WAVeform:UNSigned? (see page 951)	{0 | 1}
:WAVeform:VIEW <view> (see page 952)	:WAVeform:VIEW? (see page 952)	<view> ::= {MAIN}

Table 60 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:XINCrement? (see page 953)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 954)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 955)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 956)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 957)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 958)	<return_value> ::= y-reference value in the current preamble in NR1 format

Table 61 :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:ARBitrary:BYTeorder <order> (see page 962)	:WGEN:ARBitrary:BYTeorder? (see page 962)	<order> ::= {MSBFirst | LSBFirst}
:WGEN:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 963)	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
n/a	:WGEN:ARBitrary:DATA:ATTRibute:POINts? (see page 964)	<points> ::= number of points in NR1 format
:WGEN:ARBitrary:DATA:CLEar (see page 965)	n/a	n/a

Table 61 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 966)	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
:WGEN:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 967)	:WGEN:ARBitrary:INTerpolate? (see page 967)	{0 | 1}
:WGEN:ARBitrary:STORe <source> (see page 968)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format
:WGEN:FREQuency <frequency> (see page 969)	:WGEN:FREQuency? (see page 969)	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNCtion <signal> (see page 970)	:WGEN:FUNCtion? (see page 972)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary}
:WGEN:FUNCtion:PULSe:WIDTh <width> (see page 973)	:WGEN:FUNCtion:PULSe:WIDTh? (see page 973)	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNCtion:RAMP:SYMMetry <percent> (see page 974)	:WGEN:FUNCtion:RAMP:SYMMetry? (see page 974)	<percent> ::= symmetry percentage from 0% to 100% in NR3 format
:WGEN:FUNCtion:SQUare:DCYCle <percent> (see page 975)	:WGEN:FUNCtion:SQUare:DCYCle? (see page 975)	<percent> ::= duty cycle percentage from 20% to 80% in NR3 format
:WGEN:MODulation:NOISe <percent> (see page 976)	:WGEN:MODulation:NOISe? (see page 976)	<percent> ::= 0 to 100
:WGEN:OUTPut {{0 | OFF} | {1 | ON}} (see page 977)	:WGEN:OUTPut? (see page 977)	{0 | 1}
:WGEN:OUTPut:LOAD <impedance> (see page 978)	:WGEN:OUTPut:LOAD? (see page 978)	<impedance> ::= {ONEMeg | FIFTy}

Table 61 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:PERiod <period> (see page 979)	:WGEN:PERiod? (see page 979)	<period> ::= period in seconds in NR3 format
:WGEN:RST (see page 980)	n/a	n/a
:WGEN:VOLTage <amplitude> (see page 981)	:WGEN:VOLTage? (see page 981)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTage:HIGH <high> (see page 982)	:WGEN:VOLTage:HIGH? (see page 982)	<high> ::= high-level voltage in volts, in NR3 format
:WGEN:VOLTage:LOW <low> (see page 983)	:WGEN:VOLTage:LOW? (see page 983)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTage:OFFSet <offset> (see page 984)	:WGEN:VOLTage:OFFSet? (see page 984)	<offset> ::= offset in volts in NR3 format

Table 62 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 987)	n/a	<r> ::= 1-2 in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 988)	:WMEMory<r>:DISPlay? (see page 988)	<r> ::= 1-2 in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 989)	:WMEMory<r>:LABel? (see page 989)	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 990)	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 991)	:WMEMory<r>:SKEW? (see page 991)	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format

Table 62 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YOFFset <offset>[suffix] (see page 992)	:WMEMory<r>:YOFFset? (see page 992)	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YRANge <range>[suffix] (see page 993)	:WMEMory<r>:YRANge? (see page 993)	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YSCale <scale>[suffix] (see page 994)	:WMEMory<r>:YSCale? (see page 994)	<r> ::= 1-2 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 Agilent VISA COM library in Visual Basic, the command:

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

has a quoted ASCII string of:

```
'One'
```

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

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

5 Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See "Introduction to Common (*) Commands" on page 155.

Table 63 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 157)	n/a	n/a
*ESE <mask> (see page 158)	*ESE? (see page 158)	<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 160)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 160)	AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 163)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 164)	*OPC? (see page 164)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 63 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 165)	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <reserved>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <reserved>, <Power Measurements>, <RS-232/UART Serial>, <Segmented Memory>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <I2S Serial>, <reserved>, <reserved>, <Waveform Generator>, <reserved>, <reserved> <All field> ::= {0 | All} <reserved> ::= 0 <MSO> ::= {0 | MSO} <Memory> ::= {0 | MEMUP} <Low Speed Serial> ::= {0 | EMBD} <Automotive Serial> ::= {0 | AUTO} <Power Measurements> ::= {0 | PWR} <RS-232/UART Serial> ::= {0 | COMP} <Segmented Memory> ::= {0 | SGM} <Mask Test> ::= {0 | MASK} <Bandwidth> ::= {0 | BW20 | BW50} <I2S Serial> ::= {0 | AUDIO} <Waveform Generator> ::= {0 | WAVEGEN}
*RCL <value> (see page 167)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 168)	n/a	See *RST (Reset) (see page 168)
*SAV <value> (see page 171)	n/a	<value> ::= {0 | 1 | 4 | 5 | 6 | 7 | 8 | 9}

Table 63 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
*SRE <mask> (see page 172)	*SRE? (see page 173)	<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 174)	<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 176)	n/a	n/a
n/a	*TST? (see page 177)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 178)	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)

C (see page 1088)

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 155
- "*STB (Read Status Byte)" on page 174
- "*ESE (Standard Event Status Enable)" on page 158
- "*ESR (Standard Event Status Register)" on page 160
- "*SRE (Service Request Enable)" on page 172
- ":SYSTem:ERRor" on page 821

*ESE (Standard Event Status Enable)

C (see page 1088)

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.

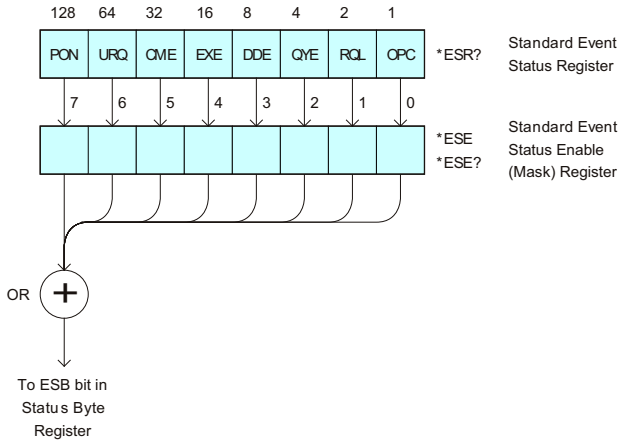


Table 64 Standard Event Status Enable (ESE)

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 155
- "*ESR (Standard Event Status Register)" on page 160
- "*OPC (Operation Complete)" on page 164
- "*CLS (Clear Status)" on page 157

*ESR (Standard Event Status Register)

C (see page 1088)

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.

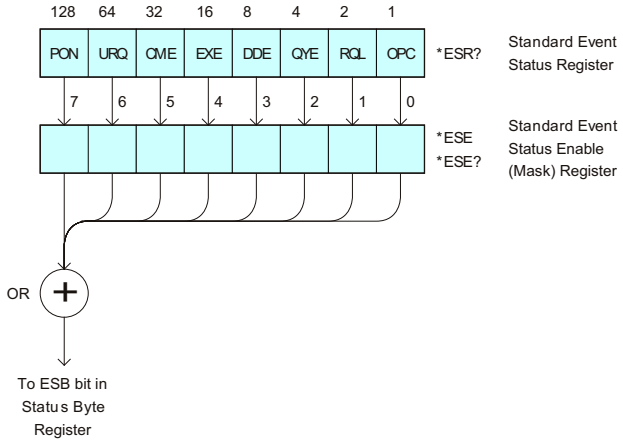


Table 65 Standard Event Status Register (ESR)

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 155
- "*ESE (Standard Event Status Enable)" on page 158
- "*OPC (Operation Complete)" on page 164
- "*CLS (Clear Status)" on page 157
- ":SYSTem:ERRor" on page 821

*IDN (Identification Number)

C (see page 1088)

Query Syntax `*IDN?`

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

Return Format

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

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

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

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

See Also

- "Introduction to Common (*) Commands" on page 155
- "*OPT (Option Identification)" on page 165

*LRN (Learn Device Setup)

C (see page 1088)

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 828) 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 Agilent oscilloscopes to match the IEEE 488.2 specification which says that the query result must contain ":SYST:SET " before the binary block data.

See Also

- "Introduction to Common (*) Commands" on page 155
- "*RCL (Recall)" on page 167
- "*SAV (Save)" on page 171
- ":SYSTem:SETup" on page 828

*OPC (Operation Complete)

C (see page 1088)

Command Syntax

```
*OPC
```

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

Query Syntax

```
*OPC?
```

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

Return Format

```
<complete><NL>

<complete> ::= 1
```

See Also

- "Introduction to Common (*) Commands" on page 155
- "*ESE (Standard Event Status Enable)" on page 158
- "*ESR (Standard Event Status Register)" on page 160
- "*CLS (Clear Status)" on page 157

*OPT (Option Identification)

C (see page 1088)

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>, <MSO>, <reserved>,
                   <Memory>, <Low Speed Serial>, <Automotive Serial>,
                   <reserved>, <FlexRay Serial>,
                   <Power Measurements>, <RS-232/UART Serial>,
                   <Segmented Memory>, <Mask Test>, <reserved>,
                   <Bandwidth>, <reserved>, <reserved>, <reserved>,
                   <I2S Serial>, <reserved>, <Educator's Kit>,
                   <Waveform Generator>, <MIL-1553/ARINC 429 Serial>,
                   <Extended Video>, <Advanced Math>, <reserved>,
                   <reserved>, <reserved>, <Digital Voltmeter>,
                   <reserved>

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

<reserved> ::= 0

<MSO> ::= {0 | MSO}

<Memory> ::= {0 | MEMUP}

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

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

<FlexRay Serial> ::= {0 | FLEX}

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

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

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

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

<Bandwidth> ::= {0 | BW20 | BW50}

<I2S Serial> ::= {0 | AUDIO}

<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> ::= {0 | DVM}
```

The <MSO> field indicates whether the unit is a mixed-signal oscilloscope.

The *OPT? query returns the following:

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

See Also

- "Introduction to Common (*) Commands" on page 155
- "*IDN (Identification Number)" on page 162

*RCL (Recall)

C (see page 1088)

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 155
- "*SAV (Save)" on page 171

*RST (Reset)

C (see page 1088)

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

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

Display Menu	
Persistence	Off
Grid	33%

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	60 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 155
- ":SYSTem:PRESet" on page 824

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 39, “Programming Examples,” starting on page 1097

*SAV (Save)

C (see page 1088)

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 155
- "*RCL (Recall)" on page 167

*SRE (Service Request Enable)

C (see page 1088)

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.

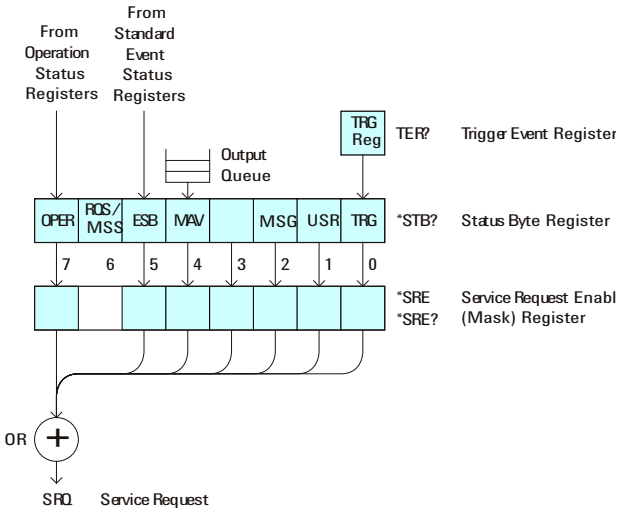


Table 66 Service Request Enable Register (SRE)

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 155
- "*STB (Read Status Byte)" on page 174
- "*CLS (Clear Status)" on page 157

*STB (Read Status Byte)

C (see page 1088)

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`

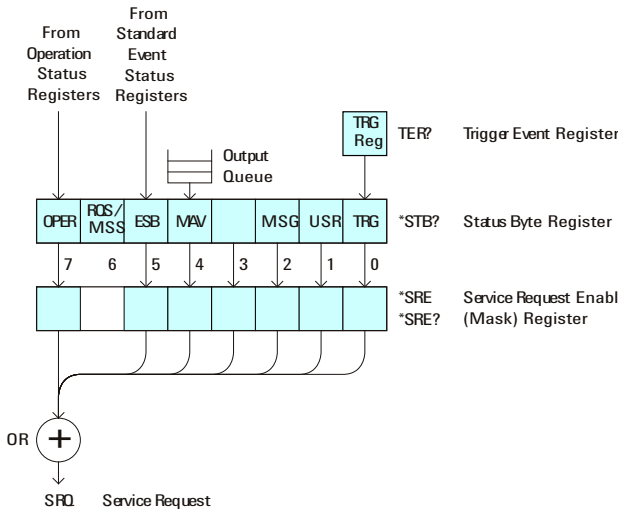


Table 67 Status Byte Register (STB)

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 155
- "*SRE (Service Request Enable)" on page 172

*TRG (Trigger)

C (see page 1088)

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 155
- ":DIGitize" on page 191
- ":RUN" on page 211
- ":STOP" on page 215

*TST (Self Test)

C (see page 1088)

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 155

*WAI (Wait To Continue)

C (see page 1088)

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 155

6 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 182.

Table 68 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 183)	:ACTivity? (see page 183)	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER? (see page 184)	{0 | 1}; an integer in NR1 format
:AUTOscale [<source>[,..,<source>]] (see page 185)	n/a	<source> ::= CHANnel<n> for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD1 | POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUTOscale:AMODE <value> (see page 187)	:AUTOscale:AMODE? (see page 187)	<value> ::= {NORMal | CURRent}}
:AUTOscale:CHANnels <value> (see page 188)	:AUTOscale:CHANnels? (see page 188)	<value> ::= {ALL | DISPlayed}}
:AUTOscale:FDEBug {{0 | OFF} | {1 | ON}} (see page 189)	:AUTOscale:FDEBug? (see page 189)	{0 | 1}

Table 68 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:BLANk [<source>] (see page 190)	n/a	<source> ::= {CHANnel<n>} | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:DIGitize [<source>[,..,<source>]] (see page 191)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:MTEenable <n> (see page 193)	:MTEenable? (see page 193)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENt]? (see page 195)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 197)	:OPEE? (see page 198)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDition? (see page 199)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERegister[:EVENt]? (see page 201)	<n> ::= 15-bit integer in NR1 format

Table 68 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:OVLenable <mask> (see page 203)	:OVLenable? (see page 204)	<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 205)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>] (see page 207)	n/a	<options> ::= [<print option>][,..,<print option>] <print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG | NOFactors | FACTors} <print option> can be repeated up to 5 times.
:PWRenable <n> (see page 208)	:PWRenable? (see page 208)	<n> ::= 16-bit integer in NR1 format
n/a	:PWRRegister[:EVENt]? (see page 210)	<n> ::= 16-bit integer in NR1 format
:RUN (see page 211)	n/a	n/a
n/a	:SERial (see page 212)	<return value> ::= unquoted string containing serial number
:SINGle (see page 213)	n/a	n/a
n/a	:STATus? <display> (see page 214)	{0 | 1} <display> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:STOP (see page 215)	n/a	n/a

Table 68 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 216)	{0 | 1}
:VIEW <source> (see page 217)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Introduction to Root (:) Commands

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

:ACTivity

N (see page 1088)

Command Syntax `:ACTivity`

The :ACTivity command clears the cumulative edge variables for the next activity query.

Query Syntax `:ACTivity?`

The :ACTivity? query returns whether there has been activity (edges) on the digital channels since the last query, and returns the current logic levels.

NOTE Because the :ACTivity? query returns edge activity since the last :ACTivity? query, you must send this query twice before the edge activity result is valid.

Return Format

```
<edges>,<levels><NL>

<edges> ::= presence of edges (16-bit integer in NR1 format).

<levels> ::= logical highs or lows (16-bit integer in NR1 format).

bit 0 ::= DIGital 0

bit 15 ::= DIGital 15
```

NOTE A bit = 0 (zero) in the <edges> result indicates that no edges were detected on that channel (across the specified threshold voltage) since the last query.

A bit = 1 (one) in the <edges> result indicates that edges have been detected on that channel (across the specified threshold voltage) since the last query.

(The threshold voltage must be set appropriately for the logic levels of the signals being probed.)

See Also

- "Introduction to Root (:) Commands" on page 182
- ":POD<n>:THReshold" on page 505
- ":DIGital<d>:THReshold" on page 287

:AER (Arm Event Register)

C (see page 1088)

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 182
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 199
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:AUToscale

C (see page 1088)

Command Syntax

```
:AUToscale

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

<source> ::= CHANnel<n> for the DSO models

<source> ::= {DIGital<d> | POD1 | POD2 | CHANnel<n>} for the
             MSO models

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

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

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

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

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see ":AUToscale:CHANnels" on page 188) is set to DISPlayed channels. Then, the autoscale is performed.

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

- Thresholds.
- Channels with activity around the trigger point are turned on, others are turned off.
- Channels are reordered on screen; analog channel 1 first, followed by the remaining analog channels, then the digital channels 0-15.
- Delay is set to 0 seconds.
- Time/Div.

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

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

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":AUToscale:CHANnels" on page 188
- ":AUToscale:AMODE" on page 187

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 39, “Programming Examples,” starting on page 1097

:AUToscale:AMODE

N (see page 1088)

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 182
- ":AUToscale" on page 185
- ":AUToscale:CHANnels" on page 188
- ":ACQuire:TYPE" on page 231
- ":ACQuire:MODE" on page 223

:AUToscale:CHANnels

N (see page 1088)

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 182
- ":AUToscale" on page 185
- ":AUToscale:AMODE" on page 187
- ":VIEW" on page 217
- ":BLANk" on page 190

:AUToscale:FDEBug

N (see page 1088)

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 182
- ":AUToscale" on page 185

:BLANk

N (see page 1088)

Command Syntax

```
:BLANk [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}}
             for the DSO models

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}}
             for the MSO models

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

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

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

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

NOTE MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 182
- ":DISPlay:CLEar" on page 295
- ":CHANnel<n>:DISPlay" on page 258
- ":DIGital<d>:DISPlay" on page 283
- ":FUNCtion:DISPlay" on page 317
- ":POD<n>:DISPlay" on page 503
- ":STATus" on page 214
- ":VIEW" on page 217

Example Code

- "Example Code" on page 217

:DIGitize

C (see page 1088)

Command Syntax

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

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}}
             for the DSO models

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}}
             for the MSO models

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

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

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

The :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 is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 182
- ":RUN" on page 211
- ":SINGle" on page 213
- ":STOP" on page 215
- ":TIMebase:MODE" on page 833
- Chapter 7, ":ACQuire Commands," starting on page 219
- Chapter 31, ":WAVeform Commands," starting on page 923

Example Code

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

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

:MTEenable (Mask Test Event Enable Register)

N (see page 1088)

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.

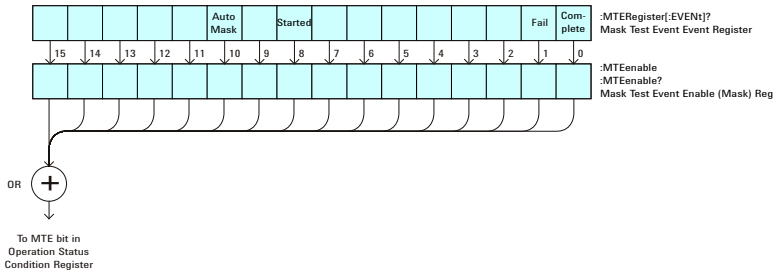


Table 69 Mask Test Event Enable Register (MTEenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	---	---	(Not used.)
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	Mask test failed.
0	Comple te	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 182
- ":AER (Arm Event Register)" on page 184

- ":CHANnel<n>:PROTection" on page 268
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

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

N (see page 1088)

Query Syntax `:MTERegister[:EVENt]?`

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

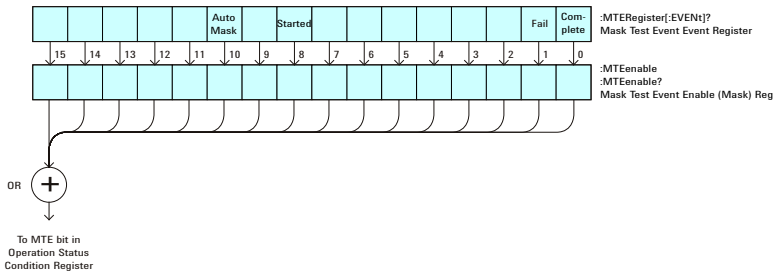


Table 70 Mask Test Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-11	---	---	(Not used.)
10	Auto Mask	Auto Mask Created	Auto mask creation completed.
9	---	---	(Not used.)
8	Started	Mask Testing Started	Mask testing started.
7-2	---	---	(Not used.)
1	Fail	Mask Test Fail	The mask test failed.
0	Comple te	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 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 199
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205

- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:OPEE (Operation Status Enable Register)

C (see page 1088)

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.

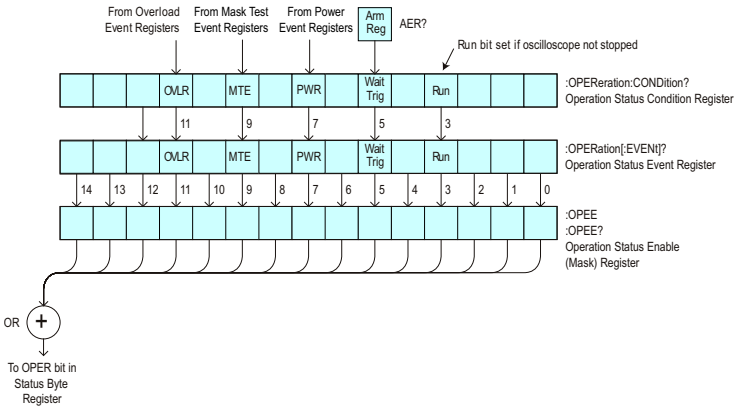


Table 71 Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
14-12	---	---	(Not used.)
11	OVLR	Overload	Event when 50Ω input overload occurs.
10	---	---	(Not used.)
9	MTE	Mask Test Event	Event when mask test event occurs.
8	---	---	(Not used.)
7	PWR	Power Event	A power measurements application event has occurred.
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 182
- ":AER (Arm Event Register)" on page 184
- ":CHANnel<n>:PROTection" on page 268
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:OPERegister:CONDition (Operation Status Condition Register)

C (see page 1088)

Query Syntax `:OPERegister:CONDition?`

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

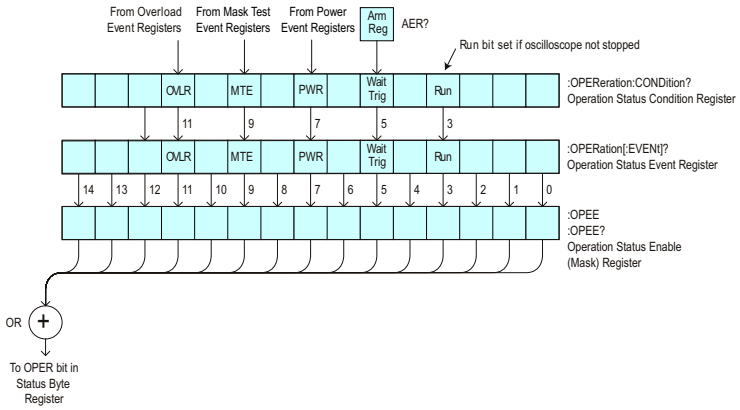


Table 72 Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14-12	---	---	(Not used.)
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)
9	MTE	Mask Test Event	A mask test event has occurred.
8	---	---	(Not used.)
7	PWR	Power Event	A power measurements application event has occurred.
6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172
- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 195
- ":MTEenable (Mask Test Event Enable Register)" on page 193

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

C (see page 1088)

Query Syntax `:OPERegister[:EVENt]?`

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

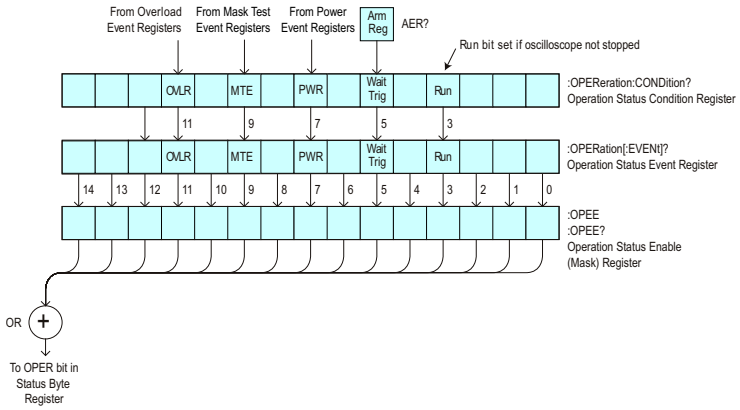


Table 73 Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14-12	---	---	(Not used.)
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)
9	MTE	Mask Test Event	A mask test event has occurred.
8	---	---	(Not used.)
7	PWR	Power Event	A power measurements application event has occurred.
6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope has gone from a stop state to a single or running state.
2-0	---	---	(Not used.)

Return Format `<value><NL>`

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 199
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172
- ":MTERegister[:EVENt] (Mask Test Event Event Register)" on page 195
- ":MTEenable (Mask Test Event Enable Register)" on page 193

:OVLenable (Overload Event Enable Register)

C (see page 1088)

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.

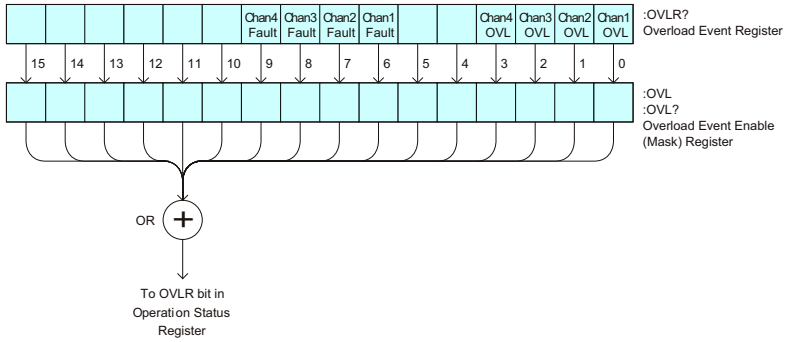


Table 74 Overload Event Enable Register (OVL)

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.
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.

Table 74 Overload Event Enable Register (OVL) (continued)

Bit	Description	When Set (1 = High = True), Enables:
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 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 199
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:OVLRegister (Overload Event Register)

C (see page 1088)

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.

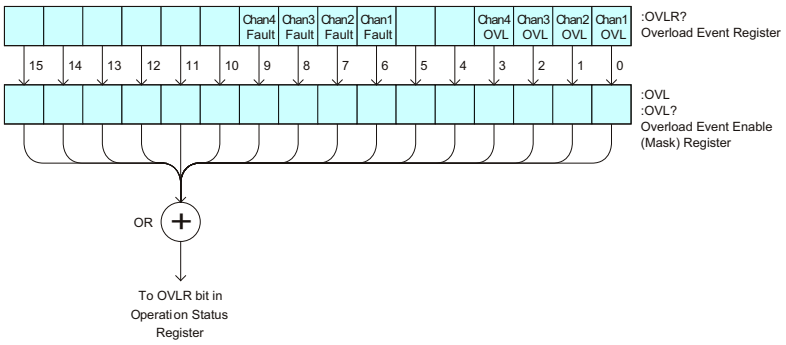


Table 75 Overload Event Register (OVLR)

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 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OVLenable (Overload Event Enable Register)" on page 203
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:PRINt

C (see page 1088)

Command Syntax

```
:PRINt [<options>]

<options> ::= [<print option>][,..,<print option>]

<print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG
                    | NOFactors | FACTors}

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

The PRINt command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used.

See Also

- "Introduction to Root (:) Commands" on page 182
- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:GRAYscale" on page 1014
- ":DISPlay:DATA" on page 296

:PWRenable (Power Event Enable Register)

N (see page 1088)

Command Syntax

```
:PWRenable <mask>

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

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

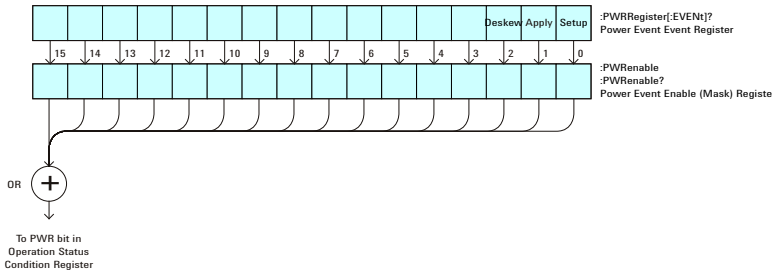


Table 76 Power Event Enable Register (PWRenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-3	---	---	(Not used.)
2	Deskew	Deskew Complete	Power analysis deskew is complete.
1	Apply	Apply Complete	Power analysis apply feature is complete.
0	Setup	Setup Complete	Power analysis auto setup feature is complete.

Query Syntax

```
:PWRenable?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":AER (Arm Event Register)" on page 184
- ":CHANnel<n>:PROTection" on page 268
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 201
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205

- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:PWRRegister[:EVENt] (Power Event Event Register)

N (see page 1088)

Query Syntax `:PWRRegister[:EVENt]?`

The :PWRRegister[:EVENt]? query returns the integer value contained in the Power Event Event Register and clears the register.

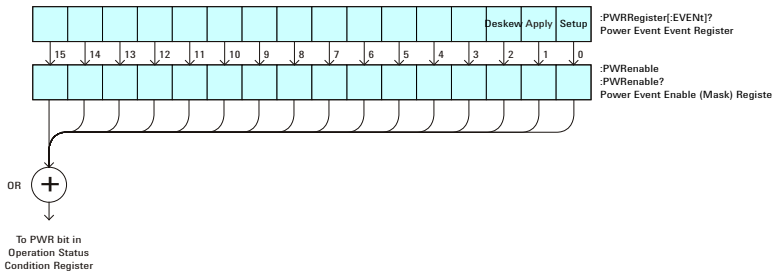


Table 77 Power Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-3	---	---	(Not used.)
2	Deskew	Deskew Complete	Power analysis deskew is complete.
1	Apply	Apply Complete	Power analysis apply feature is complete.
0	Setup	Setup Complete	Power analysis auto setup feature is complete.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":CHANnel<n>:PROTection" on page 268
- ":OPEE (Operation Status Enable Register)" on page 197
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 199
- ":OVLenable (Overload Event Enable Register)" on page 203
- ":OVLRegister (Overload Event Register)" on page 205
- "*STB (Read Status Byte)" on page 174
- "*SRE (Service Request Enable)" on page 172

:RUN

C (see page 1088)

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 182
- ":SINGle" on page 213
- ":STOP" on page 215

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 39, “Programming Examples,” starting on page 1097

:SERial

N (see page 1088)

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 182

:SINGle

C (see page 1088)

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 182
- ":RUN" on page 211
- ":STOP" on page 215

:STATus

N (see page 1088)

Query Syntax

```
:STATus? <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}}
             for the DSO models

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}}
             for the MSO models

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

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

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

NOTE MATH is an alias for FUNCtion.

Return Format

```
<value><NL>

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

See Also

- "Introduction to Root (:) Commands" on page 182
- ":BLANk" on page 190
- ":CHANnel<n>:DISPlay" on page 258
- ":DIGital<d>:DISPlay" on page 283
- ":FUNCtion:DISPlay" on page 317
- ":POD<n>:DISPlay" on page 503
- ":VIEW" on page 217

:STOP

C (see page 1088)

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 182
- ":RUN" on page 211
- ":SINGle" on page 213

Example Code

- "Example Code" on page 211

:TER (Trigger Event Register)

C (see page 1088)

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 182
- "*SRE (Service Request Enable)" on page 172
- "*STB (Read Status Byte)" on page 174

:VIEW

N (see page 1088)

Command Syntax

```
:VIEW <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS{1 | 2}}
             for DSO models

<source> ::= {CHANnel<n> | DIGital<d> | POD{1 | 2}
             | BUS{1 | 2} | FUNCtion | MATH | SBUS{1 | 2}}
             for MSO models

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

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

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

NOTE MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 182
- ":BLANk" on page 190
- ":CHANnel<n>:DISPlay" on page 258
- ":DIGital<d>:DISPlay" on page 283
- ":FUNCtion:DISPlay" on page 317
- ":POD<n>:DISPlay" on page 503
- ":STATus" on page 214

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 39, “Programming Examples,” starting on page 1097

7 :ACQuire Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 219.

Table 78 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMPlete <complete> (see page 221)	:ACQuire:COMPlete? (see page 221)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 222)	:ACQuire:COUNt? (see page 222)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 223)	:ACQuire:MODE? (see page 223)	<mode> ::= {RTIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 224)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMented:ANALyze (see page 225)	n/a	n/a (with Option SGM)
:ACQuire:SEGMented:COUNt <count> (see page 226)	:ACQuire:SEGMented:COUNt? (see page 226)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
:ACQuire:SEGMented:INDex <index> (see page 227)	:ACQuire:SEGMented:INDex? (see page 227)	<index> ::= an integer from 1 to 1000 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATe? (see page 230)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 231)	:ACQuire:TYPE? (see page 231)	<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:COMPlete

C (see page 1088)

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 219
- ":ACQuire:TYPE" on page 231
- ":DIGitize" on page 191
- ":WAVeform:POINts" on page 936

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 39, “Programming Examples,” starting on page 1097

:ACQuire:COUNt

C (see page 1088)

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 219
- ":ACQuire:TYPE" on page 231
- ":DIGitize" on page 191
- ":WAVeform:COUNt" on page 932

:ACQuire:MODE

C (see page 1088)

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 219
- ":ACQuire:TYPE" on page 231

:ACQuire:POINts

C (see page 1088)

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 219
- ":DIGitize" on page 191
- ":WAVeform:POINts" on page 936

:ACQuire:SEGMented:ANALyze

N (see page 1088)

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 223
- ":ACQuire:SEGMented:COUNt" on page 226
- "Introduction to :ACQuire Commands" on page 219

:ACQuire:SEGMented:COUNt

N (see page 1088)

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 (Option SGM) is enabled.

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 223
- ":DIGitize" on page 191
- ":SINGle" on page 213
- ":RUN" on page 211
- ":WAVeform:SEGMented:COUNt" on page 943
- ":ACQuire:SEGMented:ANALyze" on page 225
- "Introduction to :ACQuire Commands" on page 219

Example Code

- "Example Code" on page 227

:ACQuire:SEGMented:INDex

N (see page 1088)

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 (Option SGM) is enabled.

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 223
- ":ACQuire:SEGMented:COUNt" on page 226
- ":DIGitize" on page 191
- ":SINGle" on page 213
- ":RUN" on page 211
- ":WAVeform:SEGMented:COUNt" on page 943
- ":WAVeform:SEGMented:TTAG" on page 944
- ":ACQuire:SEGMented:ANALyze" on page 225
- "Introduction to :ACQuire Commands" on page 219

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("USB0::0x0957::0x17A6::US50210029::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

N (see page 1088)

Query Syntax `:ACQuire:SRATe?`

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

Return Format `<sample_rate><NL>`

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

See Also

- "Introduction to :ACQuire Commands" on page 219
- ":ACQuire:POINts" on page 224

:ACQuire:TYPE

C (see page 1088)

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 219
- ":ACQuire:COUNt" on page 222
- ":ACQuire:MODE" on page 223
- ":DIGitize" on page 191
- ":WAVeform:FORMat" on page 935
- ":WAVeform:TYPE" on page 950
- ":WAVeform:PREamble" on page 940

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 39, “Programming Examples,” starting on page 1097

8 :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "Introduction to :BUS<n> Commands" on page 234.

Table 79 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 | OFF} | {1 | ON}} (see page 235)	:BUS<n>:BIT<m>? (see page 235)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 | OFF} | {1 | ON}} (see page 236)	:BUS<n>:BITS? (see page 236)	<channel_list>, {0 | 1} <channel_list> ::= (@<m>,<m>:<m>...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:CLEar (see page 238)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 239)	:BUS<n>:DISPlay? (see page 239)	{0 | 1} <n> ::= 1 or 2; an integer in NR1 format

Table 79 :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 240)	:BUS<n>:LABel? (see page 240)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 241)	:BUS<n>:MASK? (see page 241)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

Introduction to :BUS<n> Commands

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

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

NOTE These commands are only valid for the MSO models.

Reporting the Setup

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

Return Format

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

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

:BUS<n>:BIT<m>

N (see page 1088)

Command Syntax

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

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

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

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

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BITS" on page 236
- ":BUS<n>:CLEar" on page 238
- ":BUS<n>:DISPlay" on page 239
- ":BUS<n>:LABel" on page 240
- ":BUS<n>:MASK" on page 241

Example Code

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

:BUS<n>:BITS

N (see page 1088)

Command Syntax

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

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

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

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

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

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

Return Format

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

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

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

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BIT<m>" on page 235
- ":BUS<n>:CLEar" on page 238
- ":BUS<n>:DISPlay" on page 239
- ":BUS<n>:LABel" on page 240
- ":BUS<n>:MASK" on page 241

Example Code

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

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

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

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

:BUS<n>:CLEar

N (see page 1088)

Command Syntax

```
:BUS<n>:CLEar

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

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

NOTE This command is only valid for the MSO models.

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BIT<m>" on page 235
- ":BUS<n>:BITS" on page 236
- ":BUS<n>:DISPlay" on page 239
- ":BUS<n>:LABel" on page 240
- ":BUS<n>:MASK" on page 241

:BUS<n>:DISPlay

N (see page 1088)

Command Syntax

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

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

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

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BIT<m>" on page 235
- ":BUS<n>:BITS" on page 236
- ":BUS<n>:CLEar" on page 238
- ":BUS<n>:LABel" on page 240
- ":BUS<n>:MASK" on page 241

:BUS<n>:LABel

N (see page 1088)

Command Syntax

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

<quoted_string> ::= any series of 10 or less characters as a
quoted ASCII string.

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

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

NOTE This command is only valid for the MSO models.

NOTE Label strings are 10 characters or less, and may contain any commonly used ASCII characters. Labels with more than 10 characters are truncated to 10 characters.

Query Syntax

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

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

Return Format

```
<quoted_string><NL>

<quoted_string> ::= any series of 10 or less characters as a
quoted ASCII string.
```

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BIT<m>" on page 235
- ":BUS<n>:BITS" on page 236
- ":BUS<n>:CLEar" on page 238
- ":BUS<n>:DISPlay" on page 239
- ":BUS<n>:MASK" on page 241
- ":CHANnel<n>:LABel" on page 261
- ":DISPlay:LABList" on page 298
- ":DIGital<d>:LABel" on page 284

Example Code

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

:BUS<n>:MASK

N (see page 1088)

Command Syntax

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

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

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

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

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

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

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

Return Format

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

See Also

- "Introduction to :BUS<n> Commands" on page 234
- ":BUS<n>:BIT<m>" on page 235
- ":BUS<n>:BITS" on page 236
- ":BUS<n>:CLEar" on page 238
- ":BUS<n>:DISPlay" on page 239
- ":BUS<n>:LABel" on page 240

9 :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 243.

Table 80 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 245)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see page 246)	:CALibrate:LABel? (see page 246)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 247)	:CALibrate:OUTPut? (see page 247)	<signal> ::= {TRIGgers | MASK | WAVEgen}
n/a	:CALibrate:PROTected? (see page 248)	{PROTected | UNPRotected}
:CALibrate:STARt (see page 249)	n/a	n/a
n/a	:CALibrate:STATus? (see page 250)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see page 251)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 252)	<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

N (see page 1088)

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 243

:CALibrate:LABel

N (see page 1088)

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 243

:CALibrate:OUTPut

N (see page 1088)

Command Syntax

```
:CALibrate:OUTPut <signal>

<signal> ::= {TRIGgers | MASK | WAVEgen}
```

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

- TRIGgers — pulse when a trigger event occurs.
- MASK — signal from mask test indicating a failure.
- WAVEgen — waveform generator sync output signal. This signal depends on the :WGEN: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

Query Syntax

```
:CALibrate:OUTPut?
```

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

Return Format

```
<signal><NL>

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

See Also

- "Introduction to :CALibrate Commands" on page 243
- ":WGEN:FUNCtion" on page 970

:CALibrate:PROTected

N (see page 1088)

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> ::= {PROT | UNPR}`

See Also

- "Introduction to :CALibrate Commands" on page 243

:CALibrate:STARt

N (see page 1088)

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 243
- ":CALibrate:PROTected" on page 248

:CALibrate:STATus

N (see page 1088)

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
```

See Also

- "Introduction to :CALibrate Commands" on page 243

:CALibrate:TEMPerature

N (see page 1088)

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 243

:CALibrate:TIME

N (see page 1088)

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 243

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 254.

Table 81 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 256)	:CHANnel<n>:BWLimit? (see page 256)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 257)	:CHANnel<n>:COUPling? (see page 257)	<coupling> ::= {AC | DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 258)	:CHANnel<n>:DISPlay? (see page 258)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 259)	:CHANnel<n>:IMPedance? (see page 259)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 260)	:CHANnel<n>:INVert? (see page 260)	{0 | 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 261)	:CHANnel<n>:LABel? (see page 261)	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 262)	:CHANnel<n>:OFFSet? (see page 262)	<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 263)	:CHANnel<n>:PROBe? (see page 263)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format

Table 81 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see page 264)	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see page 264)	<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6 | DIFF12 | DIFF20 | NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 265)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 266)	:CHANnel<n>:PROBe:SKEW? (see page 266)	<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 267)	:CHANnel<n>:PROBe:STYPe? (see page 267)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 268)	:CHANnel<n>:PROTection? (see page 268)	{NORM | TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 269)	:CHANnel<n>:RANGe? (see page 269)	<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 270)	:CHANnel<n>:SCALe? (see page 270)	<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 271)	:CHANnel<n>:UNITs? (see page 271)	<units> ::= {VOLT | AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 272)	:CHANnel<n>:VERNier? (see page 272)	{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

C (see page 1088)

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 254

:CHANnel<n>:COUPling

C (see page 1088)

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 254

:CHANnel<n>:DISPlay

C (see page 1088)

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 254
- ":VIEW" on page 217
- ":BLANk" on page 190
- ":STATus" on page 214
- ":POD<n>:DISPlay" on page 503
- ":DIGital<d>:DISPlay" on page 283

:CHANnel<n>:IMPedance

C (see page 1088)

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 254

:CHANnel<n>:INVert

N (see page 1088)

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 254

:CHANnel<n>:LABel

N (see page 1088)

Command Syntax

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

<string> ::= quoted ASCII string

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

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 :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 254
- ":DISPlay:LABel" on page 297
- ":DIGital<d>:LABel" on page 284
- ":DISPlay:LABList" on page 298
- ":BUS<n>:LABel" on page 240

Example Code

```
' LABEL - This command allows you to write a name (10 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 39, “Programming Examples,” starting on page 1097

:CHANnel<n>:OFFSet

C (see page 1088)

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 254
- ":CHANnel<n>:RANGe" on page 269
- ":CHANnel<n>:SCALe" on page 270
- ":CHANnel<n>:PROBe" on page 263

:CHANnel<n>:PROBe

C (see page 1088)

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 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

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

Query Syntax

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

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

Return Format

```
<attenuation><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 254
- ":CHANnel<n>:RANGe" on page 269
- ":CHANnel<n>:SCALe" on page 270
- ":CHANnel<n>:OFFSet" on page 262

Example Code

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
' channel.  The probe attenuation factor may be set from 0.1 to 1000.
myScope.WriteString ":CHANnel1:PROBe 10"    ' Set Probe to 10:1.
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

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

C (see page 1088)

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 | 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, 0dB.
- SEND6 — Single-ended, 6dB.
- SEND12 — Single-ended, 12dB.
- SEND20 — Single-ended, 20dB.
- DIFF0 — Differential, 0dB.
- DIFF6 — Differential, 6dB.
- DIFF12 — Differential, 12dB.
- DIFF20 — Differential, 20dB.

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 | NONE}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 254
- ":CHANnel<n>:PROBe" on page 263
- ":CHANnel<n>:PROBe:ID" on page 265
- ":CHANnel<n>:PROBe:SKEW" on page 266
- ":CHANnel<n>:PROBe:STYPe" on page 267

:CHANnel<n>:PROBe:ID

C (see page 1088)

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 254

:CHANnel<n>:PROBe:SKEW

C (see page 1088)

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 254

:CHANnel<n>:PROBe:STYPe

C (see page 1088)

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 254
- ":CHANnel<n>:OFFSet" on page 262

:CHANnel<n>:PROTection

N (see page 1088)

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 259) 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 254
- ":CHANnel<n>:COUPling" on page 257
- ":CHANnel<n>:IMPedance" on page 259
- ":CHANnel<n>:PROBe" on page 263

:CHANnel<n>:RANGe

C (see page 1088)

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 254
- ":CHANnel<n>:SCALe" on page 270
- ":CHANnel<n>:PROBe" on page 263

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 39, “Programming Examples,” starting on page 1097

:CHANnel<n>:SCALe

N (see page 1088)

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 254
- ":CHANnel<n>:RANGe" on page 269
- ":CHANnel<n>:PROBe" on page 263

:CHANnel<n>:UNITs

N (see page 1088)

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 254
- ":CHANnel<n>:RANGe" on page 269
- ":CHANnel<n>:PROBe" on page 263
- ":EXTernal:UNITs" on page 305

:CHANnel<n>:VERNier

N (see page 1088)

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.

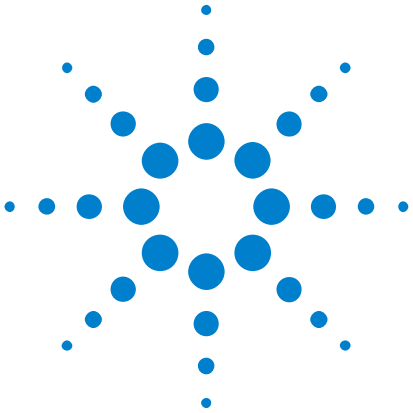
Return Format

```
<vernier value><NL>

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

See Also

- "Introduction to :CHANnel<n> Commands" on page 254



11 :DEMO Commands

When the education kit is licensed (Option EDU), you can output demonstration signals on the oscilloscope's Demo 1 and Demo 2 terminals. See "Introduction to :DEMO Commands" on page 273.

Table 82 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCtion <signal> (see page 274)	:DEMO:FUNCtion? (see page 276)	<signal> ::= {SINusoid | NOISy | PHASe | RINGing | SINGle | AM | CLK | GLITch | BURSt | MSO | RUNT | TRANsition | RFBurst | SHOLd | LFSine | FMBurst | ETE | CAN | LIN | UART | I2C | SPI | I2S | CANLin | ARINc | FLEXray | MIL | MIL2}
:DEMO:FUNCtion:PHASe:PHASe <angle> (see page 278)	:DEMO:FUNCtion:PHASe:PHASe? (see page 278)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 | OFF} | {1 | ON}} (see page 279)	:DEMO:OUTPut? (see page 279)	{0 | 1}

Introduction to :DEMO Commands

The :DEMO subsystem provides commands to output demonstration signals on the oscillosope's Demo 1 and Demo 2 terminals.

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 1088)

Command Syntax

```
:DEMO:FUNCtion <signal>

<signal> ::= {SINusoid | NOISy | PHASe | RINGing | SINGle | AM | CLK
             | GLITch | BURSt | MSO | RUNT | TRANsition | RFBurst
             | SHOLd | LFSine | FMBurst | ETE | CAN | LIN | UART
             | I2C | SPI | I2S | CANLin | ARINc | FLEXray | MIL
             | MIL2}
```

The :DEMO:FUNCtion command selects the type of demo signal:

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
SINusoid	5 MHz sine wave @ ~ 6 Vpp, 0 V offset	Off
NOISy	1 kHz sine wave @ ~ 2.4 Vpp, 0.0 V offset, with ~ 0.5 Vpp of random noise added	Off
PHASe	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset , phase shifted by the amount entered using the ":DEMO:FUNCtion:PHASe:PHASe" on page 278 command
RINGing	500 kHz digital pulse @ ~ 3 Vpp, 1.5 V offset, and ~500 ns pulse width with ringing	Off
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.	Off
AM	26 kHz sine wave, ~ 7 Vpp, 0 V offset	Amplitude modulated signal, ~ 3 Vpp, 0 V offset, with ~13 MHz carrier and sine envelope
CLK	3.6 MHz clock @ ~2 Vpp, 1 V offset, with infrequent glitch (1 glitch per 1,000,000 clocks)	Off
GLITch	Burst of 6 digital pulses (plus infrequent glitch) that occurs once every 80 µs @ ~3.6 Vpp, ~1.8 V offset	Off
BURSt	Burst of digital pulses that occur every 50 µs @ ~ 3.6 Vpp, ~1.5 V offset	Off

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
MSO	3.1 kHz stair-step sine wave output of DAC @ ~1.5 Vpp, 0.75 V offset DAC input signals are internally routed to digital channels D0 through D7	~3.1 kHz sine wave filtered from DAC output @ ~ 600 mVpp, 300 mV offset
RUNT	Digital pulse train with positive and negative runt pulses @ ~ 3.5 Vpp, 1.75 V offset	Off
TRANsition	Digital pulse train with two different edge speeds @ ~ 3.5 Vpp, 1.75 V offset	Off
RFBurst	5-cycle burst of a 10 MHz amplitude modulated sine wave @ ~ 2.6 Vpp, 0 V offset occurring once every 4 ms	Off
SHOLd	6.25 MHz digital clock @ ~ 3.5 Vpp, 1.75 V offset	Data signal @ ~3.5 Vpp, 1.75 V offset
LFSine	30 Hz sine wave @ ~2.7 Vpp, 0 V offset, with very narrow glitch near each positive peak	Off
FMBurst	FM burst, modulated from ~100 kHz to ~1 MHz, ~5.0 Vpp, ~600 mV offset.	Off
ETE	100 kHz pulse, 400 ns wide @ ~3.3 Vpp, 1.65 V offset	600 ns analog burst (@ ~3.3 Vpp, 0.7 V offset) followed by 3.6 μs digital burst @ ~3.3 Vpp, 1.65 V offset) at a 100 kHz repetitive rate
CAN	CAN_L, 125 kbps dominant-low, ~2.8 Vpp, ~1.4 V offset	Off
LIN	LIN, 19.2 kbs, ~2.8 Vpp, ~1.4 V offset	Off
UART	Receive data (RX) with odd parity, 19.2 kbps, 8-bit words, LSB out 1st, low idle @ ~2.8 Vpp, 1.4 V offset	Transmit data (TX) with odd parity, 19.2 kbps, 8-bit words, LSB out 1st, low idle @ ~ 2.8 Vpp, 1.4 V offset
I2C	I2C serial clock signal (SCL) @ ~2.8 Vpp, 1.4 V offset	I2C serial data signal (SDA) @ ~ 2.8 Vpp, 1.4 V offset

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
SPI	Off Signals are internally routed to digital channels D6 through D9: • D9 — MOSI, TTL level, with MSB out 1st (internally routed to digital input). • D8 — MISO, TTL level, with MSB out 1st (internally routed to digital input). • D7 — CLK, TTL level (internally routed to digital input). • D6 — ~CS, low-enable, TTL level (internally routed to digital input).	Off
I2S	Off Signals are internally routed to digital channels D7 through D9: • D9 — SDATA, TTL level, with "standard" alignment (internally routed to digital input). • D8 — SCLK, TTL level, (internally routed to digital input). • D7 — WS, TTL level, low for left channel, high for right channel (internally routed to digital input)	Off
CANLin	CAN_L, 250 kbps dominant-low, ~2.8 Vpp, ~1.4 V offset	LIN, 19.2 kbps, ~2.8 Vpp, ~1.4 V offset
ARINc	ARINC 429, 100 kbps, ~5 Vpp, ~0 V offset.	Off
FLEXray	FlexRay @ 10 Mbps, ~2.8 Vpp, ~0 V offset	Off
MIL	MIL-STD-1553 RT to RT transfer, received ~1.3 Vpp, transmitted ~4.8 Vpp, 0 V offset	Off
MIL2	MIL-STD-1553 RT to RT transfer, received ~1.3 Vpp, transmitted ~4.8 Vpp, 0 V offset	MIL-STD-1553 RT to BC transfer, received ~1.3 Vpp, transmitted ~4.8 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 | PHAS | RING | SINGl | AM | CLK | GLIT
             | BURS | MSO | RUNT | TRAN | RFB | SHOL | LFS | FMB
             | ETE | CAN | LIN | UART | I2C | SPI | I2S | CANL
             | ARIN | FLEX | MIL | MIL2}
```

See Also

- "Introduction to :DEMO Commands" on page 273

:DEMO:FUNCtion:PHASe:PHASe

N (see page 1088)

Command Syntax

```
:DEMO:FUNCtion:PHASe:PHASe <angle>

<angle> ::= angle in degrees from 0 to 360 in NR3 format
```

For the phase shifted sine demo signals, the :DEMO:FUNCtion:PHASe:PHASe command specifies the phase shift in the second sine waveform.

Query Syntax

```
:DEMO:FUNCtion:PHASe:PHASe?
```

The :DEMO:FUNCtion:PHASe:PHASe? query returns the currently set phase shift.

Return Format

```
<angle><NL>

<angle> ::= angle in degrees from 0 to 360 in NR3 format
```

See Also

- "Introduction to :DEMO Commands" on page 273
- ":DEMO:FUNCtion" on page 274

:DEMO:OUTPut

N (see page 1088)

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 273
- ":DEMO:FUNCtion" on page 274

12 :DIGital<d> Commands

Control all oscilloscope functions associated with individual digital channels. See "Introduction to :DIGital<d> Commands" on page 281.

Table 83 :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital<d>:DISPlay {{0 | OFF} | {1 | ON}} (see page 283)	:DIGital<d>:DISPlay? (see page 283)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 | 1}
:DIGital<d>:LABel <string> (see page 284)	:DIGital<d>:LABel? (see page 284)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGital<d>:POSition <position> (see page 285)	:DIGital<d>:POSition? (see page 285)	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.
:DIGital<d>:SIZE <value> (see page 286)	:DIGital<d>:SIZE? (see page 286)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl | MEDium | LARGe}
:DIGital<d>:THReshold <value>[suffix] (see page 287)	:DIGital<d>:THReshold? (see page 287)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS | ECL | TTL | <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V | mV | uV}

Introduction to :DIGital<d> Commands

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

The DIGital subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels, or *pods*.

NOTE These commands are only valid for the MSO models.

Reporting the Setup

Use :DIGital<d>? to query setup information for the DIGital subsystem.

Return Format

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

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

:DIGital<d>:DISPlay

N (see page 1088)

Command Syntax

```
:DIGital<d>:DISPlay <display>

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<d>:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :DIGital<d> Commands" on page 281
- ":POD<n>:DISPlay" on page 503
- ":CHANnel<n>:DISPlay" on page 258
- ":VIEW" on page 217
- ":BLANk" on page 190
- ":STATus" on page 214

:DIGital<d>:LABel

N (see page 1088)

Command Syntax

```
:DIGital<d>:LABel <string>

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

<string> ::= any series of 10 or less characters as quoted ASCII string.
```

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

NOTE This command is only valid for the MSO models.

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.

Query Syntax

```
:DIGital<d>:LABel?
```

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

Return Format

```
<label string><NL>

<label string> ::= any series of 10 or less characters as a quoted
                   ASCII string.
```

See Also

- "Introduction to :DIGital<d> Commands" on page 281
- ":CHANnel<n>:LABel" on page 261
- ":DISPlay:LABList" on page 298
- ":BUS<n>:LABel" on page 240

:DIGital<d>:POSition

N (see page 1088)

Command Syntax

```
:DIGital<d>:POSition <position>

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

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

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

The :DIGital<d>:POSition command sets the position of the specified channel. Note that bottom positions might not be valid depending on whether digital buses, serial decode waveforms, or the zoomed time base are displayed.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<d>:POSition?
```

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

If the returned value is "-1", this indicates there is no space to display the digital waveform (for example, when all serial lanes, digital buses, and the zoomed time base are displayed).

Return Format

```
<position><NL>

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

See Also

- "Introduction to :DIGital<d> Commands" on page 281

:DIGital<d>:SIZE

N (see page 1088)

Command Syntax

```
:DIGital<d>:SIZE <value>

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

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

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

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<d>:SIZE?
```

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

Return Format

```
<size_value><NL>

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

See Also

- "Introduction to :DIGital<d> Commands" on page 281
- ":POD<n>:SIZE" on page 504
- ":DIGital<d>:POSition" on page 285

:DIGital<d>:THReshold

N (see page 1088)

Command Syntax

```
:DIGital<d>:THReshold <value>

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

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

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

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

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

The :DIGital<d>:THReshold command sets the logic threshold value for all channels in the same *pod* as the specified channel. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE This command is only valid for the MSO models.

Query Syntax

```
:DIGital<d>:THReshold?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :DIGital<d> Commands" on page 281
- ":POD<n>:THReshold" on page 505
- ":TRIGger[:EDGE]:LEVel" on page 868

13 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 290.

Table 84 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation {{0 | OFF} | {1 | ON}} (see page 291)	:DISPlay:ANNotation? (see page 291)	{0 | 1}
:DISPlay:ANNotation:BACKground <mode> (see page 292)	:DISPlay:ANNotation:BACKground? (see page 292)	<mode> ::= {OPAQue | INVerted | TRANsparent}
:DISPlay:ANNotation:COLor <color> (see page 293)	:DISPlay:ANNotation:COLor? (see page 293)	<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe | RED}
:DISPlay:ANNotation:TEXT <string> (see page 294)	:DISPlay:ANNotation:TEXT? (see page 294)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 295)	n/a	n/a
n/a	:DISPlay:DATA? [<format>][,][<palette>] (see page 296)	<format> ::= {BMP | BMP8bit | PNG} <palette> ::= {COLor | GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 297)	:DISPlay:LABel? (see page 297)	{0 | 1}
:DISPlay:LABList <binary block> (see page 298)	:DISPlay:LABList? (see page 298)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters

Table 84 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:PERSistence <value> (see page 299)	:DISPlay:PERSistence? (see page 299)	<value> ::= {MINimum | INFinite | <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1 | ON} (see page 300)	:DISPlay:VECTors? (see page 300)	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 (see page 1088)

Command Syntax

```
:DISPlay:ANNotation <setting>

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

The :DISPlay:ANNotation 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?
```

The :DISPlay:ANNotation? query returns the annotation setting.

Return Format

```
<value><NL>

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

See Also

- ":DISPlay:ANNotation:TEXT" on page 294
- ":DISPlay:ANNotation:COLor" on page 293
- ":DISPlay:ANNotation:BACKground" on page 292
- "Introduction to :DISPlay Commands" on page 290

:DISPlay:ANNotation:BACKground

N (see page 1088)

Command Syntax

```
:DISPlay:ANNotation:BACKground <mode>

<mode> ::= {OPAQue | INVerted | TRANsparent}
```

The :DISPlay:ANNotation:BACKground command specifies the background of the annotation:

- OPAQue – the annotation has a solid background.
- INVerted – the annotation's foreground and background colors are switched.
- TRANsparent – the annotation has a transparent background.

Query Syntax

```
:DISPlay:ANNotation:BACKground?
```

The :DISPlay:ANNotation:BACKground? query returns the specified annotation background mode.

Return Format

```
<mode><NL>

<mode> ::= {OPAQ | INV | TRAN}
```

See Also

- ":DISPlay:ANNotation" on page 291
- ":DISPlay:ANNotation:TEXT" on page 294
- ":DISPlay:ANNotation:COLor" on page 293
- "Introduction to :DISPlay Commands" on page 290

:DISPlay:ANNotation:COLor

N (see page 1088)

Command Syntax

```
:DISPlay:ANNotation:COLor <color>

<color> ::= {CH1 | CH2 | CH3 | CH4 | DIG | MATH | REF | MARKer | WHITe
            | RED}
```

The :DISPlay:ANNotation: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:COLor?
```

The :DISPlay:ANNotation: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" on page 291
- ":DISPlay:ANNotation:TEXT" on page 294
- ":DISPlay:ANNotation:BACKground" on page 292
- "Introduction to :DISPlay Commands" on page 290

:DISPlay:ANNotation:TEXT

N (see page 1088)

Command Syntax

```
:DISPlay:ANNotation:TEXT <string>

<string> ::= quoted ASCII string (up to 254 characters)
```

The :DISPlay:ANNotation: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: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:TEXT?
```

The :DISPlay:ANNotation: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" on page 291
- ":DISPlay:ANNotation:COLor" on page 293
- ":DISPlay:ANNotation:BACKground" on page 292
- "Introduction to :DISPlay Commands" on page 290

:DISPlay:CLEar

N (see page 1088)

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 290

:DISPlay:DATA

N (see page 1088)

Query Syntax

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

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

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

The :DISPlay:DATA? query reads screen image data. You can choose 24-bit BMP, 8-bit BMP8bit, or 24-bit PNG formats in color or grayscale.

If no format or palette option is specified, the screen image is returned in BMP, COLor 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

- "Introduction to :DISPlay Commands" on page 290
- ":HARDcopy:INKSaver" on page 347
- ":PRINt" on page 207
- "*RCL (Recall)" on page 167
- "*SAV (Save)" on page 171
- ":VIEW" on page 217

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 39, “Programming Examples,” starting on page 1097

:DISPlay:LABel

N (see page 1088)

Command Syntax

```
:DISPlay:LABel <value>

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

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

Query Syntax

```
:DISPlay:LABel?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :DISPlay Commands" on page 290
- ":CHANnel<n>:LABel" on page 261

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 39, “Programming Examples,” starting on page 1097

:DISPlay:LABList

N (see page 1088)

Command Syntax

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

<binary block> ::= an ordered list of up to 75 labels, a maximum of 10
                   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 "A123456789", 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 10
                   characters each, separated by newline characters.
```

See Also

- "Introduction to :DISPlay Commands" on page 290
- ":DISPlay:LABel" on page 297
- ":CHANnel<n>:LABel" on page 261
- ":DIGital<d>:LABel" on page 284
- ":BUS<n>:LABel" on page 240

:DISPlay:PERSistence

N (see page 1088)

Command Syntax

```
:DISPlay:PERSistence <value>

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

<time> ::= seconds in in NR3 format from 100E-3 to 60E0
```

The :DISPlay:PERSistence command specifies the persistence setting:

- MINimum — indicates zero persistence.
- INFinite — indicates infinite persistence.
- <time> — for variable persistence, that is, you can specify how long acquisitions remain on the screen.

Use the :DISPlay: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>}
```

See Also

- "Introduction to :DISPlay Commands" on page 290
- ":DISPlay:CLEar" on page 295

:DISPlay:VECTors

N (see page 1088)

Command Syntax

```
:DISPlay:VECTors <vectors>

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

The only legal value for the :DISPlay:VECTors command is ON (or 1). This specifies that lines are drawn between acquired data points on the screen.

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 290

14 :EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXTernal Trigger Commands" on page 301.

Table 85 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 302)	:EXTernal:BWLimit? (see page 302)	<bwlimit> ::= {0 | OFF}
:EXTernal:PROBe <attenuation> (see page 303)	:EXTernal:PROBe? (see page 303)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range>[<suffix>] (see page 304)	:EXTernal:RANGe? (see page 304)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 305)	:EXTernal:UNITs? (see page 305)	<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 +8E+00;UNIT VOLT;PROB +1.000E+00
```

:EXTernal:BWLimit

C (see page 1088)

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 301
- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:HFReject" on page 847

:EXTernal:PROBe

C (see page 1088)

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 301
- ":EXTernal:RANGe" on page 304
- "Introduction to :TRIGger Commands" on page 843
- ":CHANnel<n>:PROBe" on page 263

:EXTernal:RANGe

C (see page 1088)

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 301
- ":EXTernal:PROBe" on page 303
- "Introduction to :TRIGger Commands" on page 843

:EXTernal:UNITs

N (see page 1088)

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 301
- "Introduction to :TRIGger Commands" on page 843
- ":EXTernal:RANGe" on page 304
- ":EXTernal:PROBe" on page 303
- ":CHANnel<n>:UNITs" on page 271

15 :FUNCtion Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCtion Commands" on page 310.

Table 86 :FUNCtion Commands Summary

Command	Query	Options and Query Returns
:FUNCtion:BUS:CLOCk <source> (see page 312)	:FUNCtion:BUS:CLOCk? (see page 312)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:FUNCtion:BUS:SLOPe <slope> (see page 313)	:FUNCtion:BUS:SLOPe? (see page 313)	<slope> ::= {NEGative | POSitive | EITHer}
:FUNCtion:BUS:YINCrement <value> (see page 314)	:FUNCtion:BUS:YINCrement? (see page 314)	<value> ::= value per bus code, in NR3 format
:FUNCtion:BUS:YORigin <value> (see page 315)	:FUNCtion:BUS:YORigin? (see page 315)	<value> ::= value at bus code = 0, in NR3 format
:FUNCtion:BUS:YUNits <units> (see page 316)	:FUNCtion:BUS:YUNits? (see page 316)	<units> ::= {VOLT | AMPere | NONE}
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}} (see page 317)	:FUNCtion:DISPlay? (see page 317)	{0 | 1}
:FUNCtion[:FFT]:CENTer <frequency> (see page 318)	:FUNCtion[:FFT]:CENTer? (see page 318)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCtion[:FFT]:SPAN (see page 319)	:FUNCtion[:FFT]:SPAN? (see page 319)	::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.

Table 86 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion[:FFT]:VTYPe <units> (see page 320)	:FUNCtion[:FFT]:VTYPe? (see page 320)	<units> ::= {DECibel | VRMS}
:FUNCtion[:FFT]:WINDow <window> (see page 321)	:FUNCtion[:FFT]:WINDow? (see page 321)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris}
:FUNCtion:FREQuency:HIGHpass <3dB_freq> (see page 322)	:FUNCtion:FREQuency:HIGHpass? (see page 322)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format
:FUNCtion:FREQuency:LOWPass <3dB_freq> (see page 323)	:FUNCtion:FREQuency:LOWPass? (see page 323)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format
:FUNCtion:GOFT:OPERation <operation> (see page 324)	:FUNCtion:GOFT:OPERation? (see page 324)	<operation> ::= {ADD | SUBTract | MULTiply}
:FUNCtion:GOFT:SOURce1 <source> (see page 325)	:FUNCtion:GOFT:SOURce1? (see page 325)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:FUNCtion:GOFT:SOURce2 <source> (see page 326)	:FUNCtion:GOFT:SOURce2? (see page 326)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:FUNCtion:INTegrate:IOFFset <input_offset> (see page 327)	:FUNCtion:INTegrate:IOFFset? (see page 327)	<input_offset> ::= DC offset correction in NR3 format.
:FUNCtion:LINear:GAIN <value> (see page 328)	:FUNCtion:LINear:GAIN? (see page 328)	<value> ::= 'A' in Ax + B, value in NR3 format
:FUNCtion:LINear:OFFSet <value> (see page 329)	:FUNCtion:LINear:OFFSet? (see page 329)	<value> ::= 'B' in Ax + B, value in NR3 format
:FUNCtion:OFFSet <offset> (see page 330)	:FUNCtion:OFFSet? (see page 330)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.

Table 86 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:OPERation <operation> (see page 331)	:FUNCtion:OPERation? (see page 332)	<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFF | FFT | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN | LOWPass | HIGHpass | DIVide | LINear | TRENd | BTIMing | BSTate}
:FUNCtion:RANGe <range> (see page 333)	:FUNCtion:RANGe? (see page 333)	<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.
:FUNCtion:REFerence <level> (see page 334)	:FUNCtion:REFerence? (see page 334)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:SCALe <scale value>[<suffix>] (see page 335)	:FUNCtion:SCALe? (see page 335)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB}
:FUNCtion:SOURce1 <source> (see page 336)	:FUNCtion:SOURce1? (see page 336)	<source> ::= {CHANnel<n> | GOFT | BUS<m>} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models <m> ::= {1 | 2} GOFT is only for FFT, INTegrate, DIFF, and SQRT operations.

Table 86 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:SOURce2 <source> (see page 338)	:FUNCtion:SOURce2? (see page 338)	<source> ::= {CHANnel<n> | NONE} <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:TRENd:MEASurement <type> (see page 339)	:FUNCtion:TRENd:MEASurement? (see page 339)	<type> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDth | NWIDth | DUTYcycle | RISetime | FALLtime}

Introduction to :FUNCtion Commands

The FUNCtion subsystem controls the math functions in the oscilloscope. As selected by the OPERation command, these math functions are available:

- Operators:
  - ADD
  - SUBTract
  - MULTiply

  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 – 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 decibel (dB).
  - SQRT – Square root

  Transforms operate on a single analog channel source or on a g(t) function that is the addition, subtraction, or multiplication of analog channel sources (specified by the GOFT commands).

With the DSOX3ADVMATH advanced math measurements license, these additional math functions are available:

- Operators:
  - DIVide
- Transforms:
  - LINear – Ax + B – The LINear commands set the gain (A) and offset (B) values for this function.

- SQUare
- ABSolute — Absolute Value
- LOG — Common Logarithm
- LN — Natural Logarithm
- EXP — Exponential (e^x)
- TEN — Base 10 exponential (10^x)

- 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.

  Filters operate on a single analog channel source or on a g(t) function that is the addition, subtraction, or multiplication of analog channel sources (specified by the GOFT commands).
- Visualizations:
  - MAGNify — Operates on a single analog channel source or on a g(t) function that is the addition, subtraction, or multiplication of analog channel sources (specified by the GOFT commands).
  - TRENd — Measurement trend — Operates on a single analog channel source. The TRENd:MEASurement command selects the measurement whose trend you want to measure.
  - BTIMing — Chart logic bus timing — Operates on a bus made up of digital channels. The BUS:YINcrement, BUS:YORigin, and BUS:YUNit commands specify function values.
  - BSTate — Chart logic bus state — Operates on a bus made up of digital channels. The BUS:YINcrement, BUS:YORigin, and BUS:YUNit commands specify function values. The BUS:CLOCk and BUS:SLOPe commands specify the clock source and edge.

The SOURce1, DISPlay, RANGe, and OFFSet (or REFerence) commands apply to any function.

Reporting the Setup

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

Return Format

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

```
:FUNC:OPER ADD;DISP 0;SOUR1 CHAN1;SOUR2 CHAN2;RANG +8.00E+00;OFFS
+0.0E+00;:FUNC:GOFT:OPER ADD;SOUR1 CHAN1;SOUR2 CHAN2
```

:FUNCtion:BUS:CLOCk

N (see page 1088)

Command Syntax

```
:FUNCtion:BUS:CLOCk <source>

<source> ::= {DIGital<d>}

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

The :FUNCtion:BUS:CLOCk command selects the clock signal source for the Chart Logic Bus State operation.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:BUS:CLOCk?
```

The :FUNCtion:BUS:CLOCk query returns the source selected for the clock signal.

Return Format

```
<source><NL>

<source> ::= {DIGital<d>}

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

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:BUS:SLOPe

N (see page 1088)

Command Syntax

```
:FUNCtion:BUS:SLOPe <slope>

<slope> ::= {NEGative | POSitive | EITHer}
```

The :FUNCtion:BUS:SLOPe command specifies the clock signal edge for the Chart Logic Bus State operation.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:BUS:SLOPe?
```

The :FUNCtion:BUS:SLOPe query returns the clock edge setting.

Return Format

```
<slope><NL>

<slope> ::= {NEGative | POSitive | EITHer}
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:BUS:YINCrement

N (see page 1088)

Command Syntax

```
:FUNCtion:BUS:YINCrement <value>

<value> ::= value per bus code, in NR3 format
```

The :FUNCtion:BUS:YINCrement command specifies the value associated with each increment in Chart Logic Bus data.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:BUS:YINCrement?
```

The :FUNCtion:BUS:YINCrement query returns the value associated with each increment in Chart Logic Bus data.

Return Format

```
<value><NL>

<value> ::= value per bus code, in NR3 format
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:BUS:YORigin

N (see page 1088)

Command Syntax

```
:FUNCtion:BUS:YORigin <value>

<value> ::= value at bus code = 0, in NR3 format
```

The :FUNCtion:BUS:YORigin command specifies the value associated with Chart Logic Bus data equal to zero.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:BUS:YORigin?
```

The :FUNCtion:BUS:YORigin query returns the value for associated with data equal to zero.

Return Format

```
<value><NL>

<value> ::= value at bus code = 0, in NR3 format
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:BUS:YUNits

N (see page 1088)

Command Syntax

```
:FUNCtion:BUS:YUNits <units>

<units> ::= {VOLT | AMPere | NONE}
```

The :FUNCtion:BUS:YUNits command specifies the vertical units for the Chart Logic Bus operations.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:BUS:YUNits?
```

The :FUNCtion:BUS:YUNits query returns the Chart Logic Bus vertical units.

Return Format

```
<units><NL>

<units> ::= {VOLT | AMP | NONE}
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:DISPlay

N (see page 1088)

Command Syntax

```
:FUNCtion:DISPlay <display>

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

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

Query Syntax

```
:FUNCtion:DISPlay?
```

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

Return Format

```
<display><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":VIEW" on page 217
- ":BLANk" on page 190
- ":STATus" on page 214

:FUNCtion[:FFT]:CENTer

N (see page 1088)

Command Syntax

```
:FUNCtion[:FFT]:CENTer <frequency>

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

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

Query Syntax

```
:FUNCtion[:FFT]:CENTer?
```

The :FUNCtion[: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 0 Hz to 25 GHz.
```

NOTE

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion[:FFT]:SPAN" on page 319
- ":TIMebase:RANGe" on page 835
- ":TIMebase:SCALe" on page 837

:FUNCtion[:FFT]:SPAN

N (see page 1088)

Command Syntax

```
:FUNCtion[:FFT]:SPAN <span>

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

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

The :FUNCtion[: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[:FFT]:SPAN?
```

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

NOTE

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

Return Format

```
<span><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion[:FFT]:CENTer" on page 318
- ":TIMebase:RANGe" on page 835
- ":TIMebase:SCALe" on page 837

:FUNCtion[:FFT]:VTYPe

N (see page 1088)

Command Syntax

```
:FUNCtion[:FFT]:VTYPe <units>

<units> ::= {DECibel | VRMS}
```

The :FUNCtion[:FFT]:VTYPe command specifies FFT vertical units as DECibel or VRMS.

Query Syntax

```
:FUNCtion[:FFT]:VTYPe?
```

The :FUNCtion[:FFT]:VTYPe? query returns the current FFT vertical units.

Return Format

```
<units><NL>

<units> ::= {DEC | VRMS}
```

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:OPERation" on page 331

:FUNCtion[:FFT]:WINDow

N (see page 1088)

Command Syntax

```
:FUNCtion[:FFT]:WINDow <window>

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

The :FUNCtion[:FFT]:WINDow command allows the selection of four 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).

Query Syntax

```
:FUNCtion[:FFT]:WINDow?
```

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

Return Format

```
<window><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310

:FUNCtion:FREQuency:HIGHpass

N (see page 1088)

Command Syntax

```
:FUNCtion:FREQuency:HIGHpass <3dB_freq>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion:FREQuency:HIGHpass command sets the high-pass filter's -3 dB cutoff frequency.

The high-pass filter is a single-pole high pass filter.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:FREQuency:HIGHpass?
```

The :FUNCtion: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:OPERation" on page 331

:FUNCtion:FREQuency:LOWPass

N (see page 1088)

Command Syntax

```
:FUNCtion:FREQuency:LOWPass <3dB_freq>

<3dB_freq> ::= -3dB cutoff frequency value in NR3 format
```

The :FUNCtion:FREQuency:LOWPass command sets the low-pass filter's -3 dB cutoff frequency.

The low-pass filter is a 4th order Bessel-Thompson filter.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:FREQuency:LOWPass?
```

The :FUNCtion: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:OPERation" on page 331

:FUNCtion:GOFT:OPERation

N (see page 1088)

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.

Query Syntax

```
:FUNCtion:GOFT:OPERation?
```

The :FUNCtion:GOFT:OPERation? query returns the current g(t) source operation setting.

Return Format

```
<operation><NL>

<operation> ::= {ADD | SUBT | MULT}
```

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:GOFT:SOURce1" on page 325
- ":FUNCtion:GOFT:SOURce2" on page 326
- ":FUNCtion:SOURce1" on page 336

:FUNCtion:GOFT:SOURce1

N (see page 1088)

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 Commands" on page 310
- ":FUNCtion:GOFT:SOURce2" on page 326
- ":FUNCtion:GOFT:OPERation" on page 324

:FUNCtion:GOFT:SOURce2

N (see page 1088)

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 Commands" on page 310
- ":FUNCtion:GOFT:SOURce1" on page 325
- ":FUNCtion:GOFT:OPERation" on page 324

:FUNCtion:INTegrate:IOFFset

N (see page 1088)

Command Syntax

```
:FUNCtion:INTegrate:IOFFset <input_offset>

<input_offset> ::= DC offset correction in NR3 format.
```

The :FUNCtion: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:INTegrate:IOFFset?
```

The :FUNCtion: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 Commands" on page 310
- ":FUNCtion:OPERation" on page 331

:FUNCtion:LINear:GAIN

N (see page 1088)

Command Syntax

```
:FUNCtion:LINear:GAIN <value>

<value> ::= 'A' in Ax + B, value in NR3 format
```

The :FUNCtion:LINear:GAIN command specifies the 'A' value in the Ax + B operation.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:LINear:GAIN?
```

The :FUNCtion:LINear:GAIN query returns the gain value.

Return Format

```
<value><NL>

<value> ::= 'A' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:LINear:OFFSet

N (see page 1088)

Command Syntax

```
:FUNCtion:LINear:OFFSet <value>

<value> ::= 'B' in Ax + B, value in NR3 format
```

The :FUNCtion:LINear:OFFSet command specifies the 'B' value in the Ax + B operation.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:LINear:OFFSet?
```

The :FUNCtion:LINear:OFFSet query returns the offset value.

Return Format

```
<value><NL>

<value> ::= 'B' in Ax + B, value in NR3 format
```

See Also

- ":FUNCtion:OPERation" on page 331

:FUNCtion:OFFSet

N (see page 1088)

Command Syntax

```
:FUNCtion:OFFSet <offset>

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

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

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

Query Syntax

```
:FUNCtion:OFFSet?
```

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

Return Format

```
<offset><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:RANGe" on page 333
- ":FUNCtion:REFerence" on page 334
- ":FUNCtion:SCALe" on page 335

:FUNCtion:OPERation

N (see page 1088)

Command Syntax

```
:FUNCtion:OPERation <operation>

<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFF | FFT
   | SQRT | MAGNify | ABSolute | SQUare | LN | LOG | EXP | TEN
   | LOWPass | HIGHpass | DIVide | LINear | TRENd | BTIMing | BSTate}
```

The :FUNCtion:OPERation command sets the desired waveform math operation:

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTiply – Source1 * source2.
- INTegrate – Integrate the selected waveform source.
- DIFF – Differentiate the selected waveform source.
- FFT – Fast Fourier Transform on the selected waveform source.
- SQRT – Square root on the selected waveform source.

With the DSOX3ADVMATH advanced math license, these additional operations are available:

- MAGNify – Magnify of the selected waveform source.
- ABSolute – Absolute value of the selected waveform source.
- SQUare – Square of the selected waveform source.
- LN – Natural logarithm on the selected waveform source.
- LOG – Common logarithm on the selected waveform source.
- EXP – Exponential (e^X) on the selected waveform source.
- TEN – Base 10 exponential (10^X) on the selected waveform source.
- LOWPass – Low-pass filter on the selected waveform source.
- HIGHpass – High-pass filter on the selected waveform source.
- DIVide – Divide operation on the selected waveform source.
- LINear – Ax + B operation on the selected waveform source.
- TRENd – Measurement Trend. The math waveform shows measurement values for each cycle of a selected waveform source.
- BTIMing – Chart Logic Bus Timing on the on the selected digital bus.
- BSTate – Chart Logic Bus State on the on the selected digital bus.

When the operation is ADD, SUBTract, MULTiply, or DIVide, the :FUNCtion:SOURce1 and :FUNCtion:SOURce2 commands are used to select source1 and source2. For all other operations, the :FUNCtion:SOURce1 command selects the waveform source.

Query Syntax

```
:FUNCtion:OPERation?
```

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

Return Format

```
<operation><NL>

<operation> ::= {ADD | SUBT | MULT | INT | DIFF | FFT | SQRT | MAGN
    | ABS | SQU | LN | LOG | EXP | TEN | LOWP | HIGH | DIV | LIN | TREN
    | BTIM | BST}
```

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:SOURce1" on page 336
- ":FUNCtion:SOURce2" on page 338

:FUNCtion:RANGe

N (see page 1088)

Command Syntax

```
:FUNCtion:RANGe <range>

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

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

Query Syntax

```
:FUNCtion:RANGe?
```

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

Return Format

```
<range><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:SCALe" on page 335

:FUNCtion:REFerence

N (see page 1088)

Command Syntax

```
:FUNCtion:REFerence <level>

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

The :FUNCtion: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:OFFSet command.

Query Syntax

```
:FUNCtion:REFerence?
```

The :FUNCtion: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 Commands" on page 310
- ":FUNCtion:OFFSet" on page 330
- ":FUNCtion:RANGe" on page 333
- ":FUNCtion:SCALe" on page 335

:FUNCtion:SCALe

N (see page 1088)

Command Syntax

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

<scale value> ::= integer in NR1 format

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

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

Query Syntax

```
:FUNCtion:SCALe?
```

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

Return Format

```
<scale value><NL>

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

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:RANGe" on page 333

:FUNCtion:SOURce1

N (see page 1088)

Command Syntax

```
:FUNCtion:SOURce1 <value>

<value> ::= {CHANnel<n> | GOFT | BUS<m>}

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models

<m> ::= {1 | 2}
```

The :FUNCtion:SOURce1 command is used for any :FUNCtion: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 GOFT parameter is only available for the transform functions, filter functions, and the magnify visualization function (see "Introduction to :FUNCtion Commands" on page 310). The GOFT parameter lets you specify, as the function input source, the addition, subtraction, or multiplication of two channels. When GOFT is used, the g(t) source is specified by the :FUNCtion:GOFT:OPERation, :FUNCtion:GOFT:SOURce1, and :FUNCtion:GOFT:SOURce2 commands.

The BUS<m> parameter is available for the bus charting visualization functions available with the DSOX3ADVMATH advanced math license.

NOTE Another shorthand notation for SOURce1 in this command/query (besides SOUR1) is SOUR.

Query Syntax

```
:FUNCtion:SOURce1?
```

The :FUNCtion:SOURce1? query returns the current source1 for function operations.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | GOFT | BUS<m>}

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models

<m> ::= {1 | 2}
```

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:OPERation" on page 331
- ":FUNCtion:GOFT:OPERation" on page 324
- ":FUNCtion:GOFT:SOURce1" on page 325

- ":FUNCtion:GOFT:SOURce2" on page 326

:FUNCtion:SOURce2

N (see page 1088)

Command Syntax

```
:FUNCtion:SOURce2 <value>

<value> ::= {CHANnel<n> | NONE}

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:SOURce2 command specifies the second source for math operator functions that have two sources. (The :FUNCtion:SOURce1 command specifies the first source.)

The :FUNCtion: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).

Query Syntax

```
:FUNCtion:SOURce2?
```

The :FUNCtion:SOURce2? query returns the currently specified second source for math operations.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | NONE}

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

See Also

- "Introduction to :FUNCtion Commands" on page 310
- ":FUNCtion:OPERation" on page 331
- ":FUNCtion:SOURce1" on page 336

:FUNCtion:TRENd:MEASurement

N (see page 1088)

Command Syntax

```
:FUNCtion:TRENd:MEASurement <type>

<type> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDth
           | NWIDth | DUTYcycle | RISetime | FALLtime}
```

The :FUNCtion:TRENd:MEASurement command selects the measurement whose trend is shown in the math waveform.

This command is available with the DSOX3ADVMATH advanced math license.

Query Syntax

```
:FUNCtion:TRENd:MEASurement?
```

The :FUNCtion:TRENd:MEASurement query returns the selected measurement.

Return Format

```
<type><NL>

<type> ::= {VAV | ACRM | VRAT | PER | FREQ | PWID | NWID | DUTY
           | RIS | FALL}
```

See Also

- ":FUNCtion:OPERation" on page 331

16 :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "Introduction to :HARDcopy Commands" on page 342.

Table 87 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 343)	:HARDcopy:AREA? (see page 343)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 344)	:HARDcopy:APRinter? (see page 344)	<active_printer> ::= {<index> | <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}} (see page 345)	:HARDcopy:FACTors? (see page 345)	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}} (see page 346)	:HARDcopy:FFEed? (see page 346)	{0 | 1}
:HARDcopy:INKSaver {{0 | OFF} | {1 | ON}} (see page 347)	:HARDcopy:INKSaver? (see page 347)	{0 | 1}
:HARDcopy:LAYout <layout> (see page 348)	:HARDcopy:LAYout? (see page 348)	<layout> ::= {LANDscape | PORTrait}
:HARDcopy:NETWork:ADDRess <address> (see page 349)	:HARDcopy:NETWork:ADDRess? (see page 349)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APPLy (see page 350)	n/a	n/a
:HARDcopy:NETWork:DOMain <domain> (see page 351)	:HARDcopy:NETWork:DOMain? (see page 351)	<domain> ::= quoted ASCII string

Table 87 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:NETWork:PASSword <password> (see page 352)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLOT <slot> (see page 353)	:HARDcopy:NETWork:SLOT? (see page 353)	<slot> ::= {NET0 | NET1}
:HARDcopy:NETWork:USERname <username> (see page 354)	:HARDcopy:NETWork:USERname? (see page 354)	<username> ::= quoted ASCII string
:HARDcopy:PALette <palette> (see page 355)	:HARDcopy:PALette? (see page 355)	<palette> ::= {COLor | GRAYscale | NONE}
n/a	:HARDcopy:PRINter:LIST? (see page 356)	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y | N} <name> ::= name of printer
:HARDcopy:STARt (see page 357)	n/a	n/a

Introduction to :HARDcopy Commands

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

:HARDC is an acceptable short form for :HARDcopy.

Reporting the Setup

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

Return Format

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

```
:HARD:APR "";AREA SCR;FACT 0;FFE 0;INKS 1;PAL NONE;LAY PORT
```

:HARDcopy:AREA

N (see page 1088)

Command Syntax

```
:HARDcopy:AREA <area>

<area> ::= SCReen
```

The :HARDcopy:AREA command controls what part of the display area is printed. Currently, the only legal choice is SCReen.

Query Syntax

```
:HARDcopy:AREA?
```

The :HARDcopy:AREA? query returns the selected display area.

Return Format

```
<area><NL>

<area> ::= SCR
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:APRinter" on page 344
- ":HARDcopy:PRINter:LIST" on page 356
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:INKSaver" on page 347
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:PALette" on page 355

:HARDcopy:APRinter

N (see page 1088)

Command Syntax

```
:HARDcopy:APRinter <active_printer>

<active_printer> ::= {<index> | <name>}

<index> ::= integer index of printer in list

<name> ::= name of printer in list
```

The :HARDcopy:APRinter command sets the active printer.

Query Syntax

```
:HARDcopy:APRinter?
```

The :HARDcopy:APRinter? query returns the name of the active printer.

Return Format

```
<name><NL>

<name> ::= name of printer in list
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:PRINter:LIST" on page 356
- ":HARDcopy:STARt" on page 357

:HARDcopy:FACTors

N (see page 1088)

Command Syntax

```
:HARDcopy:FACTors <factors>

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

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

Query Syntax

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns a flag indicating whether oscilloscope instrument settings are output on the hardcopy.

Return Format

```
<factors><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:INKSaver" on page 347
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:PALette" on page 355

:HARDcopy:FFEed

N (see page 1088)

Command Syntax

```
:HARDcopy:FFEed <ffeed>

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

The HARDcopy:FFEed command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

Query Syntax

```
:HARDcopy:FFEed?
```

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

Return Format

```
<ffeed><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:INKSaver" on page 347
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:PALette" on page 355

:HARDcopy:INKSaver

N (see page 1088)

Command Syntax

```
:HARDcopy:INKSaver <value>

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

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

Query Syntax

```
:HARDcopy:INKSaver?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:PALette" on page 355

:HARDcopy:LAYout

N (see page 1088)

Command Syntax

```
:HARDcopy:LAYout <layout>

<layout> ::= {LANDscape | PORTrait}
```

The :HARDcopy:LAYout command sets the hardcopy layout mode.

Query Syntax

```
:HARDcopy:LAYout?
```

The :HARDcopy:LAYout? query returns the selected hardcopy layout mode.

Return Format

```
<layout><NL>

<layout> ::= {LAND | PORT}
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:PALette" on page 355
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:INKSaver" on page 347

:HARDcopy:NETWork:ADDRess

N (see page 1088)

Command Syntax

```
:HARDcopy:NETWork:ADDRess <address>

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

The :HARDcopy:NETWork:ADDRess command sets the address for a network printer slot. The address is the server/computer name and the printer's share name in the \\server\share format.

The network printer slot is selected by the :HARDcopy:NETWork:SLOT command.

To apply the entered address, use the :HARDcopy:NETWork:APPLy command.

Query Syntax

```
:HARDcopy:NETWork:ADDRess?
```

The :HARDcopy:NETWork:ADDRess? query returns the specified address for the currently selected network printer slot.

Return Format

```
<address><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:SLOT" on page 353
- ":HARDcopy:NETWork:APPLy" on page 350
- ":HARDcopy:NETWork:DOMain" on page 351
- ":HARDcopy:NETWork:USERname" on page 354
- ":HARDcopy:NETWork:PASSword" on page 352

:HARDcopy:NETWork:APPLy

N (see page 1088)

Command Syntax `:HARDcopy:NETWork:APPLy`

The :HARDcopy:NETWork:APPLy command applies the network printer settings and makes the printer connection.

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:SLOT" on page 353
- ":HARDcopy:NETWork:ADDRess" on page 349
- ":HARDcopy:NETWork:DOMain" on page 351
- ":HARDcopy:NETWork:USERname" on page 354
- ":HARDcopy:NETWork:PASSword" on page 352

:HARDcopy:NETWork:DOMain

N (see page 1088)

Command Syntax

```
:HARDcopy:NETWork:DOMain <domain>

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

The :HARDcopy:NETWork:DOMain command sets the Windows network domain name.

The domain name setting is a common setting for both network printer slots.

Query Syntax

```
:HARDcopy:NETWork:DOMain?
```

The :HARDcopy:NETWork:DOMain? query returns the current Windows network domain name.

Return Format

```
<domain><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:SLOT" on page 353
- ":HARDcopy:NETWork:APPLy" on page 350
- ":HARDcopy:NETWork:ADDRess" on page 349
- ":HARDcopy:NETWork:USERname" on page 354
- ":HARDcopy:NETWork:PASSword" on page 352

:HARDcopy:NETWork:PASSword

N (see page 1088)

Command Syntax

```
:HARDcopy:NETWork:PASSword <password>

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

The :HARDcopy:NETWork:PASSword command sets the password for the specified Windows network domain and user name.

The password setting is a common setting for both network printer slots.

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:USERname" on page 354
- ":HARDcopy:NETWork:DOMain" on page 351
- ":HARDcopy:NETWork:SLOT" on page 353
- ":HARDcopy:NETWork:APPLy" on page 350
- ":HARDcopy:NETWork:ADDRess" on page 349

:HARDcopy:NETWork:SLOT

N (see page 1088)

Command Syntax

```
:HARDcopy:NETWork:SLOT <slot>

<slot> ::= {NET0 | NET1}
```

The :HARDcopy:NETWork:SLOT command selects the network printer slot used for the address and apply commands. There are two network printer slots to choose from.

Query Syntax

```
:HARDcopy:NETWork:SLOT?
```

The :HARDcopy:NETWork:SLOT? query returns the currently selected network printer slot.

Return Format

```
<slot><NL>

<slot> ::= {NET0 | NET1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:APPLy" on page 350
- ":HARDcopy:NETWork:ADDRess" on page 349
- ":HARDcopy:NETWork:DOMain" on page 351
- ":HARDcopy:NETWork:USERname" on page 354
- ":HARDcopy:NETWork:PASSword" on page 352

:HARDcopy:NETWork:USERname

N (see page 1088)

Command Syntax

```
:HARDcopy:NETWork:USERname <username>

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

The :HARDcopy:NETWork:USERname command sets the user name to use when connecting to the Windows network domain.

The user name setting is a common setting for both network printer slots.

Query Syntax

```
:HARDcopy:NETWork:USERname?
```

The :HARDcopy:NETWork:USERname? query returns the currently set user name.

Return Format

```
<username><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:NETWork:DOMain" on page 351
- ":HARDcopy:NETWork:PASSword" on page 352
- ":HARDcopy:NETWork:SLOT" on page 353
- ":HARDcopy:NETWork:APPLy" on page 350
- ":HARDcopy:NETWork:ADDRess" on page 349

:HARDcopy:PALette

N (see page 1088)

Command Syntax

```
:HARDcopy:PALette <palette>

<palette> ::= {COLor | GRAYscale | NONE}
```

The :HARDcopy:PALette command sets the hardcopy palette color.

The oscilloscope's print driver cannot print color images to color laser printers, so the COLor option is not available when connected to laser printers.

Query Syntax

```
:HARDcopy:PALette?
```

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

Return Format

```
<palette><NL>

<palette> ::= {COL | GRAY | NONE}
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:STARt" on page 357
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:INKSaver" on page 347

:HARDcopy:PRINter:LIST

N (see page 1088)

Query Syntax `:HARDcopy:PRINter:LIST?`

The :HARDcopy:PRINter:LIST? query returns a list of available printers. The list can be empty.

Return Format

```
<list><NL>

<list> ::= [<printer_spec>] ... [printer_spec>]

<printer_spec> ::= "<index>,<active>,<name>;"

<index> ::= integer index of printer

<active> ::= {Y | N}

<name> ::= name of printer (for example "DESKJET 950C")
```

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:APRinter" on page 344
- ":HARDcopy:STARt" on page 357

:HARDcopy:STARt

N (see page 1088)

Command Syntax `:HARDcopy:STARt`

The :HARDcopy:STARt command starts a print job.

See Also

- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:APRinter" on page 344
- ":HARDcopy:PRINter:LIST" on page 356
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:FFEed" on page 346
- ":HARDcopy:INKSaver" on page 347
- ":HARDcopy:LAYout" on page 348
- ":HARDcopy:PALette" on page 355

17 :LISTer Commands

Table 88 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 360)	<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 361)	:LISTer:DISPlay? (see page 361)	{OFF | SBUS1 | SBUS2 | ALL}
:LISTer:REFerence <time_ref> (see page 362)	:LISTer:REFerence? (see page 362)	<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

N (see page 1088)

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 359
- ":LISTer:DISPlay" on page 361
- "Definite-Length Block Response Data" on page 150

:LISTer:DISPlay

N (see page 1088)

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 359
- ":SBUS<n>:DISPlay" on page 598
- ":LISTer:DATA" on page 360

:LISTer:REFerence

N (see page 1088)

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 ot 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 359
- ":SBUS<n>:DISPlay" on page 598
- ":LISTer:DATA" on page 360
- ":LISTer:DISPlay" on page 361

18 :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 364.

Table 89 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 365)	:MARKer:MODE? (see page 365)	<mode> ::= {OFF | MEASurement | MANual | WAVeform}
:MARKer:X1Position <position>[suffix] (see page 366)	:MARKer:X1Position? (see page 366)	<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 367)	:MARKer:X1Y1source? (see page 367)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 368)	:MARKer:X2Position? (see page 368)	<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 369)	:MARKer:X2Y2source? (see page 369)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 370)	<return_value> ::= X cursors delta value in NR3 format

Table 89 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:XUNits <mode> (see page 371)	:MARKer:XUNits? (see page 371)	<units> ::= {SEConds | HERTz | DEGRees | PERCent}
:MARKer:XUNits:USE (see page 372)	n/a	n/a
:MARKer:Y1Position <position>[suffix] (see page 373)	:MARKer:Y1Position? (see page 373)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 374)	:MARKer:Y2Position? (see page 374)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 375)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 376)	:MARKer:YUNits? (see page 376)	<units> ::= {BASE | PERCent}
:MARKer:YUNits:USE (see page 377)	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:MODE

N (see page 1088)

Command Syntax

```
:MARKer:MODE <mode>

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

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.

Query Syntax

```
:MARKer:MODE?
```

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

Return Format

```
<mode><NL>

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

See Also

- "Introduction to :MARKer Commands" on page 364
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374

:MARKer:X1Position

N (see page 1088)

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 365).
- 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X2Position" on page 368
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MARKer:XUNits" on page 371
- ":MEASure:TSTArt" on page 1023

:MARKer:X1Y1source

N (see page 1088)

Command Syntax

```
:MARKer:X1Y1source <source>

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

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

<r> ::= {1 | 2}
```

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 365):

- 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421

:MARKer:X2Position

N (see page 1088)

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 365).
- 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Y2source" on page 369
- ":MARKer:XUNits" on page 371
- ":MEASure:TSTOp" on page 1024

:MARKer:X2Y2source

N (see page 1088)

Command Syntax

```
:MARKer:X2Y2source <source>

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

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

<r> ::= {1 | 2}
```

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 365):

- 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Y1source" on page 367
- ":MEASure:SOURce" on page 421

:MARKer:XDELta

N (see page 1088)

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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MARKer:XUNits" on page 371

:MARKer:XUNits

N (see page 1088)

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 364
- ":MARKer:XUNits:USE" on page 372
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368

:MARKer:XUNits:USE

N (see page 1088)

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 364
- ":MARKer:XUNits" on page 371
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:XDELta" on page 370

:MARKer:Y1Position

N (see page 1088)

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 365), 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MARKer:Y2Position" on page 374
- ":MARKer:YUNits" on page 376
- ":MEASure:VSTArt" on page 1029

:MARKer:Y2Position

N (see page 1088)

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 365), 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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MARKer:Y1Position" on page 373
- ":MARKer:YUNits" on page 376
- ":MEASure:VSTOp" on page 1030

:MARKer:YDELta

N (see page 1088)

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 364
- ":MARKer:MODE" on page 365
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YUNits" on page 376

:MARKer:YUNits

N (see page 1088)

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 364
- ":MARKer:YUNits:USE" on page 377
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YDELta" on page 375

:MARKer:YUNits:USE

N (see page 1088)

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 364
- ":MARKer:YUNits" on page 376
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YDELta" on page 375

19 :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 388.

Table 90 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 391)	n/a	n/a
:MEASure:AREa [<interval>][,][<source>] (see page 392)	:MEASure:AREa? [<interval>][,][<source>] (see page 392)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= area in volt-seconds, NR3 format
:MEASure:BWIDth [<source>] (see page 393)	:MEASure:BWIDth? [<source>] (see page 393)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= burst width in seconds, NR3 format
:MEASure:CLEar (see page 394)	n/a	n/a
:MEASure:COUNter [<source>] (see page 395)	:MEASure:COUNter? [<source>] (see page 395)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DEFine DELay, <delay spec> (see page 396)	:MEASure:DEFine? DELay (see page 397)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 396)	:MEASure:DEFine? THResholds (see page 397)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent | ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 399)	:MEASure:DELay? [<source1>] [,<source2>] (see page 399)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 401)	:MEASure:DUTYcycle? [<source>] (see page 401)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 402)	:MEASure:FALLtime? [<source>] (see page 402)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 403)	:MEASure:FREQuency? [<source>] (see page 403)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NEDGes [<source>] (see page 404)	:MEASure:NEDGes? [<source>] (see page 404)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the falling edge count in NR3 format
:MEASure:NPULses [<source>] (see page 405)	:MEASure:NPULses? [<source>] (see page 405)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the falling pulse count in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 406)	:MEASure:NWIDth? [<source>] (see page 406)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 407)	:MEASure:OVERshoot? [<source>] (see page 407)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PEDGes [<source>] (see page 409)	:MEASure:PEDGes? [<source>] (see page 409)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the rising edge count in NR3 format
:MEASure:PERiod [<source>] (see page 410)	:MEASure:PERiod? [<source>] (see page 410)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PHASe [<source1>] [,<source2>] (see page 411)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 411)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PPULses [<source>] (see page 412)	:MEASure:PPULses? [<source>] (see page 412)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the rising pulse count in NR3 format
:MEASure:PREShoot [<source>] (see page 413)	:MEASure:PREShoot? [<source>] (see page 413)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 414)	:MEASure:PWIDth? [<source>] (see page 414)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 415)	<result_list> ::= comma-separated list of measurement results

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:RISetime [<source>] (see page 418)	:MEASure:RISetime? [<source>] (see page 418)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 419)	:MEASure:SDEViation? [<source>] (see page 419)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1 | ON} (see page 420)	:MEASure:SHOW? (see page 420)	{1}
:MEASure:SOURce <source1> [,<source2>] (see page 421)	:MEASure:SOURce? (see page 421)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r> | EXTernal} for DSO models <source1,2> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r> | EXTernal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> | NONE}
:MEASure:STATistics <type> (see page 423)	:MEASure:STATistics? (see page 423)	<type> ::= {{ON | 1} | CURRent | MEAN | MINimum | MAXimum | STDDev | COUNt} ON ::= all statistics returned
:MEASure:STATistics:DISPlay {{0 | OFF} | {1 | ON}} (see page 424)	:MEASure:STATistics:DISPlay? (see page 424)	{0 | 1}
:MEASure:STATistics:INCRement (see page 425)	n/a	n/a

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:STATistics:MCOunt <setting> (see page 426)	:MEASure:STATistics:MCOunt? (see page 426)	<setting> ::= {INFinite | <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:RESet (see page 427)	n/a	n/a
:MEASure:STATistics:RSDeviation {{0 | OFF} | {1 | ON}} (see page 428)	:MEASure:STATistics:RSDeviation? (see page 428)	{0 | 1}
n/a	:MEASure:TEDGe? <slope><occurrence>[,<source>] (see page 429)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 431)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMPlitude [<source>] (see page 433)	:MEASure:VAMPlitude? [<source>] (see page 433)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<interval>][,][<source>] (see page 434)	:MEASure:VAVerage? [<interval>][,][<source>] (see page 434)	<interval> ::= {CYCLe | DISPlay} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 435)	:MEASure:VBASe? [<source>] (see page 435)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 436)	:MEASure:VMAX? [<source>] (see page 436)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 437)	:MEASure:VMIN? [<source>] (see page 437)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 438)	:MEASure:VPP? [<source>] (see page 438)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<interval>][,][<source1>] [,<source2>] (see page 439)	:MEASure:VRATio? [<interval>][,][<source1>] [,<source2>] (see page 439)	<interval> ::= {CYCLe | DISPlay} <source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 440)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 440)	<interval> ::= {CYCLe | DISPlay} <type> ::= {AC | DC} <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format

Table 90 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 441)	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | FUNCtion | MATH | WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format
:MEASure:VTOP [<source>] (see page 442)	:MEASure:VTOP? [<source>] (see page 442)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 443)	:MEASure:WINDow? (see page 443)	<type> ::= {MAIN | ZOOM | AUTO}
:MEASure:XMAX [<source>] (see page 444)	:MEASure:XMAX? [<source>] (see page 444)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 445)	:MEASure:XMIN? [<source>] (see page 445)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Introduction to :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the zoomed (delayed) time base mode (:TIMebase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the zoomed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on the digital channels or FFT (Fast Fourier Transform).

Reporting the Setup

Use the :MEASure? query to obtain setup information for the MEASure subsystem. (Currently, this is only :MEASure:SOURce.)

Return Format

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

```
:MEAS:SOUR CHAN1,CHAN2;STAT ON
```

:MEASure:ALL

N (see page 1088)

Command Syntax `:MEASure:ALL`

This command installs a Snapshot All measurement on the screen.

See Also

- "Introduction to :MEASure Commands" on page 388

:MEASure:AREa

N (see page 1088)

Command Syntax

```
:MEASure:AREa [<interval>][,][<source>]

<interval> ::= {CYCLe | DISPlay}

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:BWIDth

N (see page 1088)

Command Syntax

```
:MEASure:BWIDth [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:CLEar

N (see page 1088)

Command Syntax `:MEASure:CLEar`

This command clears all selected measurements and markers from the screen.

See Also

- "Introduction to :MEASure Commands" on page 388

:MEASure:COUNter

N (see page 1088)

Command Syntax

```
:MEASure:COUNter [<source>]

<source> ::= {<digital channels> | CHANnel<n> | EXTernal}

<digital channels> ::= DIGital<d> for the MSO models

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

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

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math may be selected for the source.

The counter measurement counts trigger level crossings at the selected trigger slope and displays the results in Hz. The gate time for the measurement is automatically adjusted to be 100 ms or twice the current time window, whichever is longer, up to 1 second. The counter measurement can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 X gate time).

The Y cursor shows the the edge threshold level used in the measurement.

Only one counter measurement may be displayed at a time.

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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:FREQuency" on page 403
- ":MEASure:CLEar" on page 394

:MEASure:DEFine

N (see page 1088)

Command Syntax

```
:MEASure:DEFine <meas_spec>

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DELay	THResholds
DUTYcycle		x
DELay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PREShoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

:MEASure:DEFine DELay Command Syntax

```
:MEASure:DEFine DELay,<delay spec>

<delay spec> ::= <edge_spec1>,<edge_spec2>

<edge_spec1> ::= [<slope>]<occurrence>

<edge_spec2> ::= [<slope>]<occurrence>

<slope> ::= {+ | -}

<occurrence> ::= integer
```

This command defines the behavior of the :MEASure:DELay? query by specifying the start and stop edge to be used. <edge_spec1> specifies the slope and edge number on source1. <edge_spec2> specifies the slope and edge number on source2. The measurement is taken as:

delay = t(<edge_spec2>) - t(<edge_spec1>)

NOTE

The :MEASure:DELay command and the front-panel delay measurement use an auto-edge selection method to determine the actual edge used for the measurement. The :MEASure:DEFine command has no effect on these delay measurements. The edges specified by the :MEASure:DEFine command only define the edges used by the :MEASure:DELay? query.

:MEASure:DEFine THResholds Command Syntax

```
:MEASure:DEFine THResholds,<threshold spec>

<threshold spec> ::= {STANdard}
                     | {<threshold mode>,<upper>,<middle>,<lower>}

<threshold mode> ::= {PERCent | ABSolute}
```

for <threshold mode> = PERCent:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <threshold mode> = ABSolute:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold absolute values in
                               NR3 format.
```

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.
- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGe or ":CHANnel<n>:SCALe" on page 270:CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

Query Syntax

```
:MEASure:DEFine? <meas_spec>

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

Return Format

for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <meas_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold voltages in NR3
                               format.
```

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:DELay" on page 399
- ":MEASure:SOURce" on page 421
- ":CHANnel<n>:RANGe" on page 269
- ":CHANnel<n>:SCALe" on page 270
- ":CHANnel<n>:PROBe" on page 263
- ":CHANnel<n>:UNITs" on page 271

:MEASure:DELay

N (see page 1088)

Command Syntax

```
:MEASure:DELay [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>}

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

<r> ::= 1-2 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 set by the :MEASure:DEFine command

NOTE

The :MEASure:DELay command and the front-panel delay measurement differ from the :MEASure:DELay? query.

The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DELay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

Query Syntax

```
:MEASure:DELay? [<source1>][,<source2>]
```

The :MEASure:DELay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also 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 388
- ":MEASure:DEFine" on page 396
- ":MEASure:PHASe" on page 411

:MEASure:DUTYcycle

C (see page 1088)

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

The :MEASure:DUTYcycle command installs a screen measurement and starts a duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

duty cycle = (+pulse width/period)*100

Return Format

```
<value><NL>

<value> ::= ratio of positive pulse width to period in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:PERiod" on page 410
- ":MEASure:PWIDth" on page 414
- ":MEASure:SOURce" on page 421

Example Code

- "Example Code" on page 422

:MEASure:FALLtime

C (see page 1088)

Command Syntax

```
:MEASure:FALLtime [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:RISetime" on page 418
- ":MEASure:SOURce" on page 421

:MEASure:FREQuency

C (see page 1088)

Command Syntax

```
:MEASure:FREQuency [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:PERiod" on page 410

Example Code

- "Example Code" on page 422

:MEASure:NEDGes

N (see page 1088)

Command Syntax

```
:MEASure:NEDGes [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:NPULses

N (see page 1088)

Command Syntax

```
:MEASure:NPULses [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:NWIDth

C (see page 1088)

Command Syntax

```
:MEASure:NWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:PWIDth" on page 414
- ":MEASure:PERiod" on page 410

:MEASure:OVERshoot

C (see page 1088)

Command Syntax

```
:MEASure:OVERshoot [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:PREShoot" on page 413
- ":MEASure:SOURce" on page 421
- ":MEASure:VMAX" on page 436

- ":MEASure:VTOP" on page 442
- ":MEASure:VBASe" on page 435
- ":MEASure:VMIN" on page 437

:MEASure:PEDGes

N (see page 1088)

Command Syntax

```
:MEASure:PEDGes [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:PERiod

C (see page 1088)

Command Syntax

```
:MEASure:PERiod [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:NWIDth" on page 406
- ":MEASure:PWIDth" on page 414
- ":MEASure:FREQuency" on page 403

Example Code

- "Example Code" on page 422

:MEASure:PHASe

N (see page 1088)

Command Syntax

```
:MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>}

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

<r> ::= 1-2 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 388
- ":MEASure:DELay" on page 399
- ":MEASure:PERiod" on page 410
- ":MEASure:SOURce" on page 421

:MEASure:PPULses

N (see page 1088)

Command Syntax

```
:MEASure:PPULses [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421

:MEASure:PREShoot

C (see page 1088)

Command Syntax

```
:MEASure:PREShoot [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VMIN" on page 437
- ":MEASure:VMAX" on page 436
- ":MEASure:VTOP" on page 442
- ":MEASure:VBASe" on page 435

:MEASure:PWIDth

C (see page 1088)

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:NWIDth" on page 406
- ":MEASure:PERiod" on page 410

:MEASure:RESults

N (see page 1088)

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 (left to right) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of four 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 388
- ":MEASure:STATistics" on page 423

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::130.29.70.228::inst0::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

C (see page 1088)

Command Syntax

```
:MEASure: RISetime [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:FALLtime" on page 402

:MEASure:SDEViation

N (see page 1088)

Command Syntax

```
:MEASure:SDEViation [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 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 388
- ":MEASure:VRMS" on page 440
- ":MEASure:SOURce" on page 421

:MEASure:SHOW

N (see page 1088)

Command Syntax

```
:MEASure:SHOW <show>

<show> ::= {1 | ON}
```

The :MEASure:SHOW command enables markers for tracking measurements on the display. This feature is always on.

Query Syntax

```
:MEASure:SHOW?
```

The :MEASure:SHOW? query returns the current state of the markers.

Return Format

```
<show><NL>

<show> ::= 1
```

See Also

- "Introduction to :MEASure Commands" on page 388

:MEASure:SOURce

C (see page 1088)

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNCtion
                         | MATH | WMEMory<r> | EXTernal}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

EXTernal is only a valid source for the counter measurement (and <source1>).

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELay and :MEASure:PHASe measurements.

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

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= {<digital channels> | CHAN<n> | FUNC | WMWM<r>
                         | EXT | NONE}
```

See Also:

- "Introduction to :MEASure Commands" on page 388
- ":MARKer:MODE" on page 365
- ":MARKer:X1Y1source" on page 367
- ":MARKer:X2Y2source" on page 369
- ":MEASure:DELay" on page 399

- ":MEASure:PHASe" on page 411

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 39, “Programming Examples,” starting on page 1097

:MEASure:STATistics

N (see page 1088)

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 388
- ":MEASure:RESults" on page 415
- ":MEASure:STATistics:DISPlay" on page 424
- ":MEASure:STATistics:RESet" on page 427
- ":MEASure:STATistics:INCRement" on page 425

Example Code

- "Example Code" on page 415

:MEASure:STATistics:DISPlay

N (see page 1088)

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 388
- ":MEASure:RESults" on page 415
- ":MEASure:STATistics" on page 423
- ":MEASure:STATistics:MCOunt" on page 426
- ":MEASure:STATistics:RESet" on page 427
- ":MEASure:STATistics:INCRement" on page 425
- ":MEASure:STATistics:RSDeviation" on page 428

:MEASure:STATistics:INCRement

N (see page 1088)

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 388
- ":MEASure:STATistics" on page 423
- ":MEASure:STATistics:DISPlay" on page 424
- ":MEASure:STATistics:RESet" on page 427
- ":MEASure:RESults" on page 415

:MEASure:STATistics:MCOunt

N (see page 1088)

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 388
- ":MEASure:RESults" on page 415
- ":MEASure:STATistics" on page 423
- ":MEASure:STATistics:DISPlay" on page 424
- ":MEASure:STATistics:RSDeviation" on page 428
- ":MEASure:STATistics:RESet" on page 427
- ":MEASure:STATistics:INCRement" on page 425

:MEASure:STATistics:RESet

N (see page 1088)

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 388
- ":MEASure:STATistics" on page 423
- ":MEASure:STATistics:DISPlay" on page 424
- ":MEASure:RESults" on page 415
- ":MEASure:STATistics:INCRement" on page 425

Example Code

- "Example Code" on page 415

:MEASure:STATistics:RSDeviation

N (see page 1088)

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 388
- ":MEASure:RESults" on page 415
- ":MEASure:STATistics" on page 423
- ":MEASure:STATistics:DISPlay" on page 424
- ":MEASure:STATistics:MCOunt" on page 426
- ":MEASure:STATistics:RESet" on page 427
- ":MEASure:STATistics:INCRement" on page 425

:MEASure:TEDGe

N (see page 1088)

Query Syntax

```
:MEASure:TEDGe? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a
            space or plus sign (+). A falling edge is indicated by a
            minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number
                 is one, the first crossing from the left screen edge is
                 reported.  If the number is two, the second crossing is
                 reported, etc.

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

<d> ::= 0 to (# digital channels - 1) 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 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 430.

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 39, “Programming Examples,” starting on page 1097

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:TVALue" on page 431
- ":MEASure:VTIMe" on page 441

:MEASure:TVALue

C (see page 1088)

Query Syntax

```
:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross.  The
            value can be volts or a math function value such as dB,
            Vs, or V/s.

<slope> ::= direction of the waveform.  A rising slope is indicated
            by a plus sign (+).  A falling edge is indicated by a
            minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

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

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

<r> ::= 1-2 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 388
- ":MEASure:TEDGe" on page 429
- ":MEASure:VTIMe" on page 441

:MEASure:VAMPlitude

C (see page 1088)

Command Syntax

```
:MEASure:VAMPlitude [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VBASe" on page 435
- ":MEASure:VTOP" on page 442
- ":MEASure:VPP" on page 438

:MEASure:VAVerage

C (see page 1088)

Command Syntax

```
:MEASure:VAVerage [<interval>][,][<source>]

<interval> ::= {CYCLe | DISPlay}

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 in NR1 format
```

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

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.

Query Syntax

```
:MEASure:VAVerage? [<source>]
```

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:SOURce" on page 421

:MEASure:VBASe

C (see page 1088)

Command Syntax

```
:MEASure:VBASe [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VTOP" on page 442
- ":MEASure:VAMPlitude" on page 433
- ":MEASure:VMIN" on page 437

:MEASure:VMAX

C (see page 1088)

Command Syntax

```
:MEASure:VMAX [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VMIN" on page 437
- ":MEASure:VPP" on page 438
- ":MEASure:VTOP" on page 442

:MEASure:VMIN

C (see page 1088)

Command Syntax

```
:MEASure:VMIN [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VBASe" on page 435
- ":MEASure:VMAX" on page 436
- ":MEASure:VPP" on page 438

:MEASure:VPP

C (see page 1088)

Command Syntax

```
:MEASure:VPP [<source>]

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

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

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VMAX" on page 436
- ":MEASure:VMIN" on page 437
- ":MEASure:VAMPlitude" on page 433

:MEASure:VRATio

N (see page 1088)

Command Syntax

```
:MEASure:VRATio [<interval>][,][<source1>][,<source2>]

<interval> ::= {CYCLe | DISPlay}

<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>}

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

<r> ::= 1-2 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 388
- ":MEASure:VRMS" on page 440
- ":MEASure:SOURce" on page 421

:MEASure:VRMS

C (see page 1088)

Command Syntax

```
:MEASure:VRMS [<interval>][,][<type>][,][<source>]

<interval> ::= {CYCLe | DISPlay}

<type> ::= {AC | DC}

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 in NR1 format
```

The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement. If the optional source parameter is specified, the current source is modified.

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.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRMS? [<source>]
```

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:SOURce" on page 421

:MEASure:VTIMe

N (see page 1088)

Query Syntax

```
:MEASure:VTIMe? <vtime_argument>[,<source>]

<vtime_argument> ::= time from trigger in seconds

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion | MATH
             | WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

The :MEASure:VTIMe? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>

<value> ::= value at the specified time in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:SOURce" on page 421
- ":MEASure:TEDGe" on page 429
- ":MEASure:TVALue" on page 431

:MEASure:VTOP

C (see page 1088)

Command Syntax

```
:MEASure:VTOP [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 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 388
- ":MEASure:SOURce" on page 421
- ":MEASure:VMAX" on page 436
- ":MEASure:VAMPlitude" on page 433
- ":MEASure:VBASe" on page 435

:MEASure:WINDow

N (see page 1088)

Command Syntax

```
:MEASure:WINDow <type>

<type> ::= {MAIN | ZOOM | AUTO}
```

When the zoomed time base is displayed, the :MEASure:WINDow command lets you specify the measurement window:

- MAIN — the measurement window is the upper, Main window.
- ZOOM — the measurement window is the lower, Zoom window.
- AUTO — the measurement is attempted in the lower, Zoom window; if it cannot be made there, the upper, Main window is used.

Query Syntax

```
:MEASure:WINDow?
```

The :MEASure:WINDow? query returns the current measurement window setting.

Return Format

```
<type><NL>

<type> ::= {MAIN | ZOOM | AUTO}
```

See Also

- "Introduction to :MEASure Commands" on page 388
- ":MEASure:SOURce" on page 421

:MEASure:XMAX

N (see page 1088)

Command Syntax

```
:MEASure:XMAX [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 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 388
- ":MEASure:XMIN" on page 445
- ":MEASure:TMAX" on page 1021

:MEASure:XMIN

N (see page 1088)

Command Syntax

```
:MEASure:XMIN [<source>]

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

<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format

<r> ::= 1-2 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 388
- ":MEASure:XMAX" on page 444
- ":MEASure:TMIN" on page 1022

20 :MEASure Power Commands

These :MEASure commands are available when the DSOX3PWR power measurements and analysis application is licensed and enabled.

Table 91 :MEASure Power Commands Summary

Command	Query	Options and Query Returns
:MEASure:ANGLe [<source1>][,<source2>] (see page 450)	:MEASure:ANGLe? [<source1>][,<source2>] (see page 450)	<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 451)	:MEASure:APParent? [<source1>][,<source2>] (see page 451)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the apparent power value in NR3 format
:MEASure:CRESt [<source>] (see page 452)	:MEASure:CRESt? [<source>] (see page 452)	<source> ::= {CHANnel<n>| FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the crest factor value in NR3 format
:MEASure:EFFiciency (see page 453)	:MEASure:EFFiciency? (see page 453)	<return_value> ::= percent value in NR3 format
:MEASure:ELOSs [<source>] (see page 454)	:MEASure:ELOSs? [<source>] (see page 454)	<source> ::= {CHANnel<n>| FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the energy loss value in NR3 format

Table 91 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FACTor [<source1>][,<source2>] (see page 455)	:MEASure:FACTor? [<source1>][,<source2>] (see page 455)	<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 456)	:MEASure:IPOWer? (see page 456)	<return_value> ::= the input power value in NR3 format
:MEASure:OFFTime [<source1>][,<source2>] (see page 457)	:MEASure:OFFTime? [<source1>][,<source2>] (see page 457)	<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 458)	:MEASure:ONTime? [<source1>][,<source2>] (see page 458)	<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 459)	:MEASure:OPOWer? (see page 459)	<return_value> ::= the output power value in NR3 format
:MEASure:PCURrent [<source>] (see page 460)	:MEASure:PCURrent? [<source>] (see page 460)	<source> ::= {CHANnel<n>| FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the peak current value in NR3 format
:MEASure:PLOSs [<source>] (see page 461)	:MEASure:PLOSs? [<source>] (see page 461)	<source> ::= {CHANnel<n>| FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the power loss value in NR3 format
:MEASure:REACtive [<source1>][,<source2>] (see page 462)	:MEASure:REACtive? [<source1>][,<source2>] (see page 462)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the reactive power value in NR3 format

Table 91 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:REAL [<source>] (see page 463)	:MEASure:REAL? [<source>] (see page 463)	<source> ::= {CHANnel<n>| FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the real power value in NR3 format
:MEASure:RIPPle [<source>] (see page 464)	:MEASure:RIPPle? [<source>] (see page 464)	<source> ::= {CHANnel<n>| FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the output ripple value in NR3 format
:MEASure:TRESponse [<source>] (see page 465)	:MEASure:TRESponse? [<source>] (see page 465)	<source> ::= {CHANnel<n>| FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= time in seconds for the overshoot to settle back into the band in NR3 format

:MEASure:ANGLe

N (see page 1088)

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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:APParent

N (see page 1088)

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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:CRESt

N (see page 1088)

Command Syntax

```
:MEASure:CRESt [<source>]

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

<n> ::= 1 to (# analog channels) 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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:EFFiciency

N (see page 1088)

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 549
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:EFFiciency:APPLy" on page 512

:MEASure:ELOSs

N (see page 1088)

Command Syntax

```
:MEASure:ELOSs [<source>]

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

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

<r> ::= 1-2 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 421
- ":POWer:SWITch:APPLy" on page 553

:MEASure:FACTor

N (see page 1088)

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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:IPOWer

N (see page 1088)

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 549
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:EFFiciency:APPLy" on page 512

:MEASure:OFFTime

N (see page 1088)

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 421
- ":POWer:ONOFf:TEST" on page 533
- ":POWer:ONOFf:APPLy" on page 530

:MEASure:ONTime

N (see page 1088)

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 421
- ":POWer:ONOFf:TEST" on page 533
- ":POWer:ONOFf:APPLy" on page 530

:MEASure:OPOWer

N (see page 1088)

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 549
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:EFFiciency:APPLy" on page 512

:MEASure:PCURrent

N (see page 1088)

Command Syntax

```
:MEASure:PCURrent [<source>]

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

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

<r> ::= 1-2 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 421
- ":POWer:INRush:APPLy" on page 524

:MEASure:PLOSs

N (see page 1088)

Command Syntax

```
:MEASure:PLOSs [<source>]

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

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

<r> ::= 1-2 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 421
- ":POWer:SWITch:APPLy" on page 553

:MEASure:REACtive

N (see page 1088)

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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:REAL

N (see page 1088)

Command Syntax

```
:MEASure:REAL [<source>]

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

<n> ::= 1 to (# analog channels) 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 421
- ":POWer:QUALity:TYPE" on page 539
- ":POWer:QUALity:APPLy" on page 538

:MEASure:RIPPle

N (see page 1088)

Command Syntax

```
:MEASure:RIPPle [<source>]

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

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

<r> ::= 1-2 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 421
- ":POWer:RIPPle:APPLy" on page 540

:MEASure:TRESponse

N (see page 1088)

Command Syntax

```
:MEASure:TRESponse [<source>]

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

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

<r> ::= 1-2 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 421
- ":POWer:TRANsient:APPLy" on page 559

21 :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "Introduction to :MTESt Commands" on page 469.

Table 92 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 | OFF} | {1 | ON}} (see page 472)	:MTESt:ALL? (see page 472)	{0 | 1}
:MTESt:AMASk:CREate (see page 473)	n/a	n/a
:MTESt:AMASk:SOURce <source> (see page 474)	:MTESt:AMASk:SOURce? (see page 474)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:MTESt:AMASk:UNITs <units> (see page 475)	:MTESt:AMASk:UNITs? (see page 475)	<units> ::= {CURRent | DIVisions}
:MTESt:AMASk:XDELta <value> (see page 476)	:MTESt:AMASk:XDELta? (see page 476)	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see page 477)	:MTESt:AMASk:YDELta? (see page 477)	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVeforms? [CHANnel<n>] (see page 478)	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see page 479)	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see page 480)	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVeforms? (see page 481)	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see page 482)	:MTESt:DATA? (see page 482)	<mask> ::= data in IEEE 488.2 # format.

Table 92 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:DELete (see page 483)	n/a	n/a
:MTESt:ENABle {{0 | OFF} | {1 | ON}} (see page 484)	:MTESt:ENABle? (see page 484)	{0 | 1}
:MTESt:LOCK {{0 | OFF} | {1 | ON}} (see page 485)	:MTESt:LOCK? (see page 485)	{0 | 1}
:MTESt:RMODe <rmode> (see page 486)	:MTESt:RMODe? (see page 486)	<rmode> ::= {FOREver | TIME | SIGMa | WAVeforms}
:MTESt:RMODe:FACTion:MEASure {{0 | OFF} | {1 | ON}} (see page 487)	:MTESt:RMODe:FACTion:MEASure? (see page 487)	{0 | 1}
:MTESt:RMODe:FACTion:PRINt {{0 | OFF} | {1 | ON}} (see page 488)	:MTESt:RMODe:FACTion:PRINt? (see page 488)	{0 | 1}
:MTESt:RMODe:FACTion:SAVE {{0 | OFF} | {1 | ON}} (see page 489)	:MTESt:RMODe:FACTion:SAVE? (see page 489)	{0 | 1}
:MTESt:RMODe:FACTion:STOP {{0 | OFF} | {1 | ON}} (see page 490)	:MTESt:RMODe:FACTion:STOP? (see page 490)	{0 | 1}
:MTESt:RMODe:SIGMa <level> (see page 491)	:MTESt:RMODe:SIGMa? (see page 491)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see page 492)	:MTESt:RMODe:TIME? (see page 492)	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVeforms <count> (see page 493)	:MTESt:RMODe:WAVeforms? (see page 493)	<count> ::= number of waveforms in NR1 format
:MTESt:SCALe:BIND {{0 | OFF} | {1 | ON}} (see page 494)	:MTESt:SCALe:BIND? (see page 494)	{0 | 1}
:MTESt:SCALe:X1 <x1_value> (see page 495)	:MTESt:SCALe:X1? (see page 495)	<x1_value> ::= X1 value in NR3 format

Table 92 :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:SCALe:XDELta <xdelta_value> (see page 496)	:MTESt:SCALe:XDELta? (see page 496)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 497)	:MTESt:SCALe:Y1? (see page 497)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 498)	:MTESt:SCALe:Y2? (see page 498)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 499)	:MTESt:SOURce? (see page 499)	<source> ::= {CHANnel<n> | NONE} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
n/a	:MTESt:TITLe? (see page 500)	<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("USB0::0x0957::0x17A6::US50210029::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

N (see page 1088)

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 469

:MTESt:AMASk:CREate

N (see page 1088)

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 469
- ":MTESt:AMASk:XDELta" on page 476
- ":MTESt:AMASk:YDELta" on page 477
- ":MTESt:AMASk:UNITs" on page 475
- ":MTESt:AMASk:SOURce" on page 474
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:AMASk:SOURce

N (see page 1088)

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 469
- ":MTESt:AMASk:XDELta" on page 476
- ":MTESt:AMASk:YDELta" on page 477
- ":MTESt:AMASk:UNITs" on page 475
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:AMASk:UNITs

N (see page 1088)

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 469
- ":MTESt:AMASk:XDELta" on page 476
- ":MTESt:AMASk:YDELta" on page 477
- ":CHANnel<n>:UNITs" on page 271
- ":MTESt:AMASk:SOURce" on page 474
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:AMASk:XDELta

N (see page 1088)

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 469
- ":MTESt:AMASk:UNITs" on page 475
- ":MTESt:AMASk:YDELta" on page 477
- ":MTESt:AMASk:SOURce" on page 474
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:AMASk:YDELta

N (see page 1088)

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 469
- ":MTESt:AMASk:UNITs" on page 475
- ":MTESt:AMASk:XDELta" on page 476
- ":MTESt:AMASk:SOURce" on page 474
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:COUNt:FWAVeforms

N (see page 1088)

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 469
- ":MTESt:COUNt:WAVeforms" on page 481
- ":MTESt:COUNt:TIME" on page 480
- ":MTESt:COUNt:RESet" on page 479
- ":MTESt:SOURce" on page 499

Example Code

- "Example Code" on page 469

:MTESt:COUNt:RESet

N (see page 1088)

Command Syntax `:MTESt:COUNt:RESet`

The :MTESt:COUNt:RESet command resets the mask statistics.

See Also

- "Introduction to :MTESt Commands" on page 469
- ":MTESt:COUNt:WAVeforms" on page 481
- ":MTESt:COUNt:FWAVeforms" on page 478
- ":MTESt:COUNt:TIME" on page 480

:MTESt:COUNt:TIME

N (see page 1088)

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 469
- ":MTESt:COUNt:WAVeforms" on page 481
- ":MTESt:COUNt:FWAVeforms" on page 478
- ":MTESt:COUNt:RESet" on page 479

Example Code

- "Example Code" on page 469

:MTESt:COUNt:WAVeforms

N (see page 1088)

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 469
- ":MTESt:COUNt:FWAVeforms" on page 478
- ":MTESt:COUNt:TIME" on page 480
- ":MTESt:COUNt:RESet" on page 479

Example Code

- "Example Code" on page 469

:MTESt:DATA

N (see page 1088)

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 584
- ":RECall:MASK[:STARt]" on page 569

:MTESt:DELete

N (see page 1088)

Command Syntax `:MTESt:DELete`

The :MTESt:DELete command clears the currently loaded mask.

See Also

- "Introduction to :MTESt Commands" on page 469
- ":MTESt:AMASk:CREate" on page 473

:MTESt:ENABle

N (see page 1088)

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 469

:MTESt:LOCK

N (see page 1088)

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 469
- ":MTESt:SOURce" on page 499

:MTESt:RMODe

N (see page 1088)

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 491 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 492 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 493 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 469
- ":MTESt:RMODe:SIGMa" on page 491
- ":MTESt:RMODe:TIME" on page 492
- ":MTESt:RMODe:WAVeforms" on page 493

Example Code

- "Example Code" on page 469

:MTESt:RMODe:FACTion:MEASure

N (see page 1088)

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 469
- ":MTESt:RMODe:FACTion:PRINt" on page 488
- ":MTESt:RMODe:FACTion:SAVE" on page 489
- ":MTESt:RMODe:FACTion:STOP" on page 490

:MTESt:RMODe:FACTion:PRINt

N (see page 1088)

Command Syntax

```
:MTESt:RMODe:FACTion:PRINt <on_off>

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

The :MTESt:RMODe:FACTion:PRINt command sets printing on mask failures on or off.

NOTE Setting :MTESt:RMODe:FACTion:PRINt ON automatically sets :MTESt:RMODe:FACTion:SAVE OFF.

See Chapter 16, ":HARDcopy Commands," starting on page 341 for more information on setting the hardcopy device and formatting options.

Query Syntax

```
:MTESt:RMODe:FACTion:PRINt?
```

The :MTESt:RMODe:FACTion:PRINt? query returns the current mask failure print setting.

Return Format

```
<on_off><NL>

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

See Also

- "Introduction to :MTESt Commands" on page 469
- ":MTESt:RMODe:FACTion:MEASure" on page 487
- ":MTESt:RMODe:FACTion:SAVE" on page 489
- ":MTESt:RMODe:FACTion:STOP" on page 490

:MTESt:RMODe:FACTion:SAVE

N (see page 1088)

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 25, ":SAVE Commands," starting on page 573 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 469
- ":MTESt:RMODe:FACTion:MEASure" on page 487
- ":MTESt:RMODe:FACTion:PRINt" on page 488
- ":MTESt:RMODe:FACTion:STOP" on page 490

:MTESt:RMODe:FACTion:STOP

N (see page 1088)

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 469
- ":MTESt:RMODe:FACTion:MEASure" on page 487
- ":MTESt:RMODe:FACTion:PRINt" on page 488
- ":MTESt:RMODe:FACTion:SAVE" on page 489

:MTESt:RMODe:SIGMa

N (see page 1088)

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 469
- ":MTESt:RMODe" on page 486

Example Code

- "Example Code" on page 469

:MTESt:RMODe:TIME

N (see page 1088)

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 469
- ":MTESt:RMODe" on page 486

:MTESt:RMODe:WAVeforms

N (see page 1088)

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 469
- ":MTESt:RMODe" on page 486

:MTESt:SCALe:BIND

N (see page 1088)

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 469
- ":MTESt:SCALe:X1" on page 495
- ":MTESt:SCALe:XDELta" on page 496
- ":MTESt:SCALe:Y1" on page 497
- ":MTESt:SCALe:Y2" on page 498

:MTESt:SCALe:X1

N (see page 1088)

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 * ΔX) + 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 469
- ":MTESt:SCALe:BIND" on page 494
- ":MTESt:SCALe:XDELta" on page 496
- ":MTESt:SCALe:Y1" on page 497
- ":MTESt:SCALe:Y2" on page 498

:MTESt:SCALe:XDELta

N (see page 1088)

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 469
- ":MTESt:SCALe:BIND" on page 494
- ":MTESt:SCALe:X1" on page 495
- ":MTESt:SCALe:Y1" on page 497
- ":MTESt:SCALe:Y2" on page 498

:MTESt:SCALe:Y1

N (see page 1088)

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 469
- ":MTESt:SCALe:BIND" on page 494
- ":MTESt:SCALe:X1" on page 495
- ":MTESt:SCALe:XDELta" on page 496
- ":MTESt:SCALe:Y2" on page 498

:MTESt:SCALe:Y2

N (see page 1088)

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 469
- ":MTESt:SCALe:BIND" on page 494
- ":MTESt:SCALe:X1" on page 495
- ":MTESt:SCALe:XDELta" on page 496
- ":MTESt:SCALe:Y1" on page 497

:MTESt:SOURce

N (see page 1088)

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 469
- ":MTESt:AMASk:SOURce" on page 474

:MTESt:TITLe

N (see page 1088)

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 469

22 :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "Introduction to :POD<n> Commands" on page 501.

Table 93 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 503)	:POD<n>:DISPlay? (see page 503)	{0 | 1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see page 504)	:POD<n>:SIZE? (see page 504)	<value> ::= {SMALl | MEDium | LARGe}
:POD<n>:THReshold <type>[suffix] (see page 505)	:POD<n>:THReshold? (see page 505)	<n> ::= 1-2 in NR1 format <type> ::= {CMOS | ECL | TTL | <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V | mV | uV }

Introduction to :POD<n> Commands

<n> ::= {1 | 2}

The POD subsystem commands control the viewing and threshold of groups of digital channels.

POD1 ::= D0-D7

POD2 ::= D8-D15

NOTE These commands are only valid for the MSO models.

Reporting the Setup

Use :POD1? or :POD2? to query setup information for the POD subsystem.

Return Format

The following is a sample response from the :POD1? query. In this case, the query was issued following a *RST command.

```
:POD1:DISP 0;THR +1.40E+00
```

:POD<n>:DISPlay

N (see page 1088)

Command Syntax

```
:POD<n>:DISPlay <display>

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

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

POD1 ::= D0-D7

POD2 ::= D8-D15
```

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:POD<n>:DISPlay?
```

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

Return Format

```
<display><NL>

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

See Also

- "Introduction to :POD<n> Commands" on page 501
- ":DIGital<d>:DISPlay" on page 283
- ":CHANnel<n>:DISPlay" on page 258
- ":VIEW" on page 217
- ":BLANk" on page 190
- ":STATus" on page 214

:POD<n>:SIZE

N (see page 1088)

Command Syntax

```
:POD<n>:SIZE <value>

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

POD1 ::= D0-D7

POD2 ::= D8-D15

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

The :POD<n>:SIZE command specifies the size of digital channels on the display. Sizes are set for all pods. Therefore, if you set the size on pod 1 (for example), the same size is set on pod 2 as well.

NOTE This command is only valid for the MSO models.

Query Syntax

```
:POD<n>:SIZE?
```

The :POD<n>:SIZE? query returns the digital channels size setting.

Return Format

```
<size_value><NL>

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

See Also

- "Introduction to :POD<n> Commands" on page 501
- ":DIGital<d>:SIZE" on page 286
- ":DIGital<d>:POSition" on page 285

:POD<n>:THReshold

N (see page 1088)

Command Syntax

```
:POD<n>:THReshold <type>[<suffix>]

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

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

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

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

POD1 ::= D0-D7

POD2 ::= D8-D15

TTL ::= 1.4V

CMOS ::= 2.5V

ECL ::= -1.3V
```

The :POD<n>:THReshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

NOTE This command is only valid for the MSO models.

Query Syntax

```
:POD<n>:THReshold?
```

The :POD<n>:THReshold? query returns the threshold value for the specified group of channels.

Return Format

```
<threshold><NL>

<threshold> ::= Floating point number in NR3 format
```

See Also

- "Introduction to :POD<n> Commands" on page 501
- ":DIGital<d>:THReshold" on page 287
- ":TRIGger[:EDGE]:LEVel" on page 868

Example Code

```
' THRESHOLD - This command is used to set the voltage threshold for
' the waveforms.  There are three preset values (TTL, CMOS, and ECL)
' and you can also set a user-defined threshold value between
' -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, then set channels
' 8-15 to a user-defined 2.0 volts, and then set the external trigger
' to TTL.  Of course, you only need to set the thresholds for the
' channels you will be using in your program.
```

```
' Set channels 0-7 to CMOS threshold.
myScope.WriteString ":POD1:THRESHOLD CMOS"

' Set channels 8-15 to 2.0 volts.
myScope.WriteString ":POD2:THRESHOLD 2.0"

' Set external channel to TTL threshold (short form).
myScope.WriteString ":TRIG:LEV TTL,EXT"
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

23 :POWer Commands

These :POWer commands are available when the DSOX3PWR power measurements and analysis application is licensed and enabled.

Table 94 :POWer Commands Summary

Command	Query	Options and Query Returns
:POWer:DESKew (see page 511)	n/a	n/a
:POWer:EFFiciency:APPLy (see page 512)	n/a	n/a
:POWer:ENABle {{0 | OFF} | {1 | ON}} (see page 513)	:POWer:ENABle? (see page 513)	{0 | 1}
:POWer:HARMonics:APPLy (see page 514)	n/a	n/a
n/a	:POWer:HARMonics:DATA? (see page 515)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISPlay <display> (see page 516)	:POWer:HARMonics:DISPlay? (see page 516)	<display> ::= {TABLe | BAR | OFF}
n/a	:POWer:HARMonics:FAILcount? (see page 517)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 518)	:POWer:HARMonics:LINE? (see page 518)	<frequency> ::= {F50 | F60 | F400}
n/a	:POWer:HARMonics:POWerfactor? (see page 519)	<value> ::= Class C power factor in NR3 format
n/a	:POWer:HARMonics:RUNCount? (see page 520)	<count> ::= integer in NR1 format
:POWer:HARMonics:STANdard <class> (see page 521)	:POWer:HARMonics:STANdard? (see page 521)	<class> ::= {A | B | C | D}

Table 94 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:POWer:HARMonics:STATus? (see page 522)	<status> ::= {PASS | FAIL | UNTested}
n/a	:POWer:HARMonics:THD? (see page 523)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLy (see page 524)	n/a	n/a
:POWer:INRush:EXIT (see page 525)	n/a	n/a
:POWer:INRush:NEXT (see page 526)	n/a	n/a
:POWer:MODulation:APPLy (see page 527)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 528)	:POWer:MODulation:SOURce? (see page 528)	<source> ::= {V | I}
:POWer:MODulation:TYPE <modulation> (see page 529)	:POWer:MODulation:TYPE? (see page 529)	<modulation> ::= {VAVerage | ACRMs | VRATio | PERiod | FREQuency | PWIDith | NWIDth | DUTYcycle | RISetime | FALLtime}
:POWer:ONOFf:APPLy (see page 530)	n/a	n/a
:POWer:ONOFf:EXIT (see page 531)	n/a	n/a
:POWer:ONOFf:NEXT (see page 532)	n/a	n/a
:POWer:ONOFf:TEST {{0 | OFF} | {1 | ON}} (see page 533)	:POWer:ONOFf:TEST? (see page 533)	{0 | 1}
:POWer:PSRR:APPLy (see page 534)	n/a	n/a
:POWer:PSRR:FREQuency:MAXimum <value>[suffix] (see page 535)	:POWer:PSRR:FREQuency:MAXimum? (see page 535)	<value> ::= {10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000 | 20000000} [suffix] ::= {Hz | kHz| MHz}
:POWer:PSRR:FREQuency:MINimum <value>[suffix] (see page 536)	:POWer:PSRR:FREQuency:MINimum? (see page 536)	<value> ::= {1 | 10 | 100 | 1000 | 10000 | 100000 | 1000000 | 10000000} [suffix] ::= {Hz | kHz| MHz}

Table 94 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:RMAXimum <value> (see page 537)	:POWer:PSRR:RMAXimum? (see page 537)	<value> ::= Maximum ratio value in NR1 format
:POWer:QUALity:APPLy (see page 538)	n/a	n/a
:POWer:QUALity:TYPE <quality> (see page 539)	:POWer:QUALity:TYPE? (see page 539)	<quality> ::= {FACTor | REAL | APParent | REACtive | CRESt | ANGLe}
:POWer:RIPPle:APPLy (see page 540)	n/a	n/a
:POWer:SIGNals:AUTosetup <analysis> (see page 542)	n/a	<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity | SLEW | SWITch}
:POWer:SIGNals:CYCLes <count> (see page 542)	:POWer:SIGNals:CYCLes? (see page 542)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURation <value>[suffix] (see page 543)	:POWer:SIGNals:DURation? (see page 543)	<value> ::= value in NR3 format [suffix] ::= {s | ms | us | ns}
:POWer:SIGNals:IEXPected <value>[suffix] (see page 544)	:POWer:SIGNals:IEXPected? (see page 544)	<value> ::= Expected current value in NR3 format [suffix] ::= {A | mA}
:POWer:SIGNals:OVERshoot <percent> (see page 545)	:POWer:SIGNals:OVERshoot? (see page 545)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V | mV}}
:POWer:SIGNals:VMAXimum <value>[suffix] (see page 546)	:POWer:SIGNals:VMAXimum? (see page 546)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:VSTeady <value>[suffix] (see page 547)	:POWer:SIGNals:VSTeady? (see page 547)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V | mV}
:POWer:SIGNals:SOURce:CURRent<i> <source> (see page 548)	:POWer:SIGNals:SOURce:CURRent<i>? (see page 548)	<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 549)	:POWer:SIGNals:SOURce:VOLTage<i>? (see page 549)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 94 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SLEW:APPLy (see page 550)	n/a	n/a
:POWer:SLEW:SOURce <source> (see page 551)	:POWer:SLEW:SOURce? (see page 551)	<source> ::= {V | I}
n/a	:POWer:SLEW:VALue? (see page 552)	<value> ::= slew rate in NR3 format
:POWer:SWITch:APPLy (see page 553)	n/a	n/a
:POWer:SWITch:CONDuction <conduction> (see page 554)	:POWer:SWITch:CONDuction? (see page 554)	<conduction> ::= {WAVeform | RDS | VCE}
:POWer:SWITch:IREFerence <percent> (see page 555)	:POWer:SWITch:IREFerence? (see page 555)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 556)	:POWer:SWITch:RDS? (see page 556)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM | mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 557)	:POWer:SWITch:VCE? (see page 557)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V | mV}
:POWer:SWITch:VREFerence <percent> (see page 558)	:POWer:SWITch:VREFerence? (see page 558)	<percent> ::= percent in NR1 format
:POWer:TRANsient:APPLy (see page 559)	n/a	n/a
:POWer:TRANsient:EXIT (see page 560)	n/a	n/a
:POWer:TRANsient:IINitial <value>[suffix] (see page 561)	:POWer:TRANsient:IINitial? (see page 561)	<value> ::= Initial current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:INEW <value>[suffix] (see page 562)	:POWer:TRANsient:INEW? (see page 562)	<value> ::= New current value in NR3 format [suffix] ::= {A | mA}
:POWer:TRANsient:NEXT (see page 563)	n/a	n/a

:POWer:DESKew

N (see page 1088)

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 *DSOX3PWR 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 549
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548

:POWer:EFFiciency:APPLy

N (see page 1088)

Command Syntax `:POWer:EFFiciency:APPLy`

The :POWer:EFFiciency:APPLy command applies the effieciency 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

- ":MEASure:EFFiciency" on page 453
- ":MEASure:IPOWer" on page 456
- ":MEASure:OPOWer" on page 459

:POWer:ENABle

N (see page 1088)

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 20, ":MEASure Power Commands," starting on page 447

:POWer:HARMonics:APPLy

N (see page 1088)

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 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:DATA

N (see page 1088)

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 514
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:DISPlay

N (see page 1088)

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 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:FAILcount

N (see page 1088)

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 520
- ":POWer:HARMonics:APPLy" on page 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:LINE

N (see page 1088)

Command Syntax

```
:POWer:HARMonics:LINE <frequency>

<frequency> ::= {F50 | F60 | F400}
```

The :POWer:HARMonics:LINE command specifies the line frequency setting for the current carmonics analysis:

- F50 — 50 Hz.
- F60 — 60 Hz.
- F400 — 400 Hz.

Query Syntax

```
:POWer:HARMonics:LINE?
```

The :POWer:HARMonics:LINE query returns the line frequency setting.

Return Format

```
<frequency><NL>

<frequency> ::= {F50 | F60 | F400}
```

See Also

- ":POWer:HARMonics:APPLy" on page 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:POWerfactor

N (see page 1088)

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 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:RUNCount

N (see page 1088)

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 517
- ":POWer:HARMonics:APPLy" on page 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:STANdard

N (see page 1088)

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 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STATus" on page 522
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:STATus

N (see page 1088)

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 520
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:APPLy" on page 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:THD" on page 523

:POWer:HARMonics:THD

N (see page 1088)

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 514
- ":POWer:HARMonics:DATA" on page 515
- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:HARMonics:FAILcount" on page 517
- ":POWer:HARMonics:LINE" on page 518
- ":POWer:HARMonics:POWerfactor" on page 519
- ":POWer:HARMonics:RUNCount" on page 520
- ":POWer:HARMonics:STANdard" on page 521
- ":POWer:HARMonics:STATus" on page 522

:POWer:INRush:APPLy

N (see page 1088)

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

- ":MEASure:PCURrent" on page 460
- ":POWer:INRush:EXIT" on page 525
- ":POWer:INRush:NEXT" on page 526

:POWer:INRush:EXIT

N (see page 1088)

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 524
- ":POWer:INRush:NEXT" on page 526

:POWer:INRush:NEXT

N (see page 1088)

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 524
- ":POWer:INRush:EXIT" on page 525

:POWer:MODulation:APPLy

N (see page 1088)

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 528
- ":POWer:MODulation:TYPE" on page 529
- ":MEASure:VAVerage" on page 434
- ":MEASure:VRMS" on page 440
- ":MEASure:VRATio" on page 439
- ":MEASure:PERiod" on page 410
- ":MEASure:FREQuency" on page 403
- ":MEASure:PWIDth" on page 414
- ":MEASure:NWIDth" on page 406
- ":MEASure:DUTYcycle" on page 401
- ":MEASure:RISetime" on page 418
- ":MEASure:FALLtime" on page 402

:POWer:MODulation:SOURce

N (see page 1088)

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 527
- ":POWer:MODulation:TYPE" on page 529

:POWer:MODulation:TYPE

N (see page 1088)

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 528
- ":POWer:MODulation:APPLy" on page 527
- ":MEASure:VAVerage" on page 434
- ":MEASure:VRMS" on page 440
- ":MEASure:VRATio" on page 439
- ":MEASure:PERiod" on page 410
- ":MEASure:FREQuency" on page 403
- ":MEASure:PWIDth" on page 414
- ":MEASure:NWIDth" on page 406
- ":MEASure:DUTYcycle" on page 401
- ":MEASure:RISetime" on page 418
- ":MEASure:FALLtime" on page 402

:POWer:ONOFf:APPLy

N (see page 1088)

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" on page 547
- ":MEASure:ONTime" on page 458
- ":MEASure:OFFTime" on page 457
- ":POWer:ONOFf:TEST" on page 533
- ":POWer:ONOFf:EXIT" on page 531
- ":POWer:ONOFf:NEXT" on page 532

:POWer:ONOFf:EXIT

N (see page 1088)

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:APPLy" on page 530
- ":POWer:ONOFf:NEXT" on page 532
- ":POWer:ONOFf:TEST" on page 533

:POWer:ONOFf:NEXT

N (see page 1088)

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:APPLy" on page 530
- ":POWer:ONOFf:EXIT" on page 531
- ":POWer:ONOFf:TEST" on page 533

:POWer:ONOFf:TEST

N (see page 1088)

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:APPLy" on page 530
- ":POWer:ONOFf:EXIT" on page 531
- ":POWer:ONOFf:NEXT" on page 532

:POWer:PSRR:APPLy

N (see page 1088)

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.

See Also

- ":POWer:PSRR:FREQuency:MAXimum" on page 535
- ":POWer:PSRR:FREQuency:MINimum" on page 536
- ":POWer:PSRR:RMAXimum" on page 537

:POWer:PSRR:FREQuency:MAXimum

N (see page 1088)

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:APPLy" on page 534
- ":POWer:PSRR:FREQuency:MINimum" on page 536
- ":POWer:PSRR:RMAXimum" on page 537

:POWer:PSRR:FREQuency:MINimum

N (see page 1088)

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:APPLy" on page 534
- ":POWer:PSRR:FREQuency:MAXimum" on page 535
- ":POWer:PSRR:RMAXimum" on page 537

:POWer:PSRR:RMAXimum

N (see page 1088)

Command Syntax

```
:POWer:PSRR:RMAXimum <value>

<value> ::= Maximum ratio value in NR1 format
```

The :POWer:PSRR:RMAXimum command specifies the vertical scale of the PSRR math waveform.

Query Syntax

```
:POWer:PSRR:RMAXimum?
```

The :POWer:PSRR:RMAXimum query returns the currently specified maximum ratio setting.

Return Format

```
<value><NL>

<value> ::= Maximum ratio value in NR1 format
```

See Also

- ":POWer:PSRR:RMAXimum" on page 537
- ":POWer:PSRR:FREQuency:MAXimum" on page 535
- ":POWer:PSRR:FREQuency:MINimum" on page 536

:POWer:QUALity:APPLy

N (see page 1088)

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

- ":POWer:QUALity:TYPE" on page 539
- ":MEASure:FACTor" on page 455
- ":MEASure:REAL" on page 463
- ":MEASure:APParent" on page 451
- ":MEASure:REACtive" on page 462
- ":MEASure:CRESt" on page 452
- ":MEASure:ANGLe" on page 450

:POWer:QUALity:TYPE

N (see page 1088)

Command Syntax

```
:POWer:QUALity:TYPE <quality>

<quality> ::= {FACTor | REAL | APParent | REACtive | CRESt | ANGLe}
```

The :POWer:QUALity:TYPE command selects the type of measurement to make in the power quality analysis:

- FACTor — Power Factor — Ratio of the actual power to the apparent power.
- REAL — Real (Actual) Power — The portion of power flow that, averaged over a complete cycle of the AC waveform, results in net transfer of energy in one direction.
- APParent — Apparent Power — The portion of power flow due to stored energy, which returns to the source in each cycle.
- REACtive — Reactive Power — The difference between apparent power and real power due to reactance.
- CRESt — Crest Factor — Crest factor is the ratio between the instantaneous peak current/voltage required by the load and the RMS current/voltage (RMS stands for Root Mean Square, which is a type of average).
- ANGLe — Phase Angle — In the *power triangle* (the right triangle where $\text{apparent\_power}^2 = \text{real\_power}^2 + \text{reactive\_power}^2$), phase angle is the angle between the apparent power and the real power, indicating the amount of reactive power.

Query Syntax

```
:POWer:QUALity:TYPE?
```

The :POWer:QUALity:TYPE query returns the selected power quality measurement type.

Return Format

```
<quality><NL>

<quality> ::= {FACT | REAL | APP | REAC | CRES | ANGL}
```

See Also

- ":MEASure:FACTor" on page 455
- ":MEASure:REAL" on page 463
- ":MEASure:APParent" on page 451
- ":MEASure:REACtive" on page 462
- ":MEASure:CRESt" on page 452
- ":MEASure:ANGLe" on page 450
- ":POWer:QUALity:APPLy" on page 538

:POWer:RIPPle:APPLy

N (see page 1088)

Command Syntax `:POWer:RIPPle:APPLy`

The :POWer:RIPPle:APPLy command applies the output ripple analysis.

See Also

- ":MEASure:RIPPle" on page 464

:POWer:SIGNals:AUTosetup

N (see page 1088)

Command Syntax

```
:POWer:SIGNals:AUTosetup <analysis>

<analysis> ::= {HARMonics | EFFiciency | RIPPle | MODulation | QUALity
   | SLEW | SWITch}
```

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 516
- ":POWer:EFFiciency:APPLy" on page 512
- ":POWer:RIPPle:APPLy" on page 540
- ":POWer:MODulation:APPLy" on page 527
- ":POWer:QUALity:APPLy" on page 538
- ":POWer:SLEW:APPLy" on page 550
- ":POWer:SWITch:APPLy" on page 553
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:CYCLes

N (see page 1088)

Command Syntax

```
:POWer:SIGNals:CYCLes <count>

<count> ::= integer in NR1 format

Legal values are 1 to 100.
```

The :POWer:SIGNals:CYCLes command specifies the number of cycles to include in the power analysis.

Query Syntax

```
:POWer:SIGNals:CYCLes?
```

The :POWer:SIGNals:CYCLes query returns the number of cycles currently set.

Return Format

```
<count><NL>

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

See Also

- ":POWer:HARMonics:DISPlay" on page 516
- ":POWer:QUALity:APPLy" on page 538
- ":POWer:SLEW:APPLy" on page 550
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:DURation

N (see page 1088)

Command Syntax

```
:POWer:SIGNals:DURation <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}
```

The :POWer:SIGNals:DURation command specifies the duration of the power analysis

This command is available in certain types of power analysis only, for example, Efficiency, Modulation, Transient Response, Turn On/Off Time, and Output Ripple.

Query Syntax

```
:POWer:SIGNals:DURation?
```

The :POWer:SIGNals:DURation query returns the set duration time value.

Return Format

```
<value><NL>

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

See Also

- ":POWer:EFFiciency:APPLy" on page 512
- ":POWer:MODulation:APPLy" on page 527
- ":POWer:ONOFf:APPLy" on page 530
- ":POWer:TRANsient:APPLy" on page 559
- ":POWer:RIPPle:APPLy" on page 540
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:IEXPected

N (see page 1088)

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 524
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:OVERshoot

N (see page 1088)

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 559
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:VMAXimum

N (see page 1088)

Command Syntax

```
:POWer:SIGNals:VMAXimum <value>[suffix]

<value> ::= Maximum expected input Voltage in NR3 format

[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VMAXimum command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage.

This command is available for the Inrush Current and Turn On/Off Time types of power analysis.

Query Syntax

```
:POWer:SIGNals:VMAXimum?
```

The :POWer:SIGNals:VMAXimum query returns the expected maximum input voltage setting.

Return Format

```
<value><NL>

<value> ::= Maximum expected input Voltage in NR3 format
```

See Also

- ":POWer:INRush:APPLy" on page 524
- ":POWer:ONOFf:APPLy" on page 530
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:VSTeady

N (see page 1088)

Command Syntax

```
:POWer:SIGNals:VSTeady <value>[suffix]

<value> ::= Expected steady state output Voltage value in NR3 format

[suffix] ::= {V | mV}
```

The :POWer:SIGNals:VSTeady command specifies the expected steady state output DC voltage of the power supply.

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?
```

The :POWer:SIGNals:VSTeady 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 530
- ":POWer:TRANsient:APPLy" on page 559
- ":POWer:SIGNals:AUTosetup" on page 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:SOURce:CURRent<i>

N (see page 1088)

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 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:VOLTage<i>" on page 549

:POWer:SIGNals:SOURce:VOLTage<i>

N (see page 1088)

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 541
- ":POWer:SIGNals:CYCLes" on page 542
- ":POWer:SIGNals:DURation" on page 543
- ":POWer:SIGNals:IEXPected" on page 544
- ":POWer:SIGNals:OVERshoot" on page 545
- ":POWer:SIGNals:VMAXimum" on page 546
- ":POWer:SIGNals:VSTeady" on page 547
- ":POWer:SIGNals:SOURce:CURRent<i>" on page 548

:POWer:SLEW:APPLy

N (see page 1088)

Command Syntax `:POWer:SLEW:APPLy`

The :POWer:SLEW:APPLy command applies the slew rate analysis.

See Also

- ":POWer:SLEW:SOURce" on page 551
- ":POWer:SLEW:VALue" on page 552

:POWer:SLEW:SOURce

N (see page 1088)

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 550
- ":POWer:SLEW:VALue" on page 552

:POWer:SLEW:VALue

N (see page 1088)

Query Syntax `:POWer:SLEW:VALue?`

The :POWer:SLEW:VALue query returns the slew rate dV/dt or dI/dt value, depending on the selected slew source.

Return Format `<value> ::= slew rate in NR3 format`

See Also

- ":POWer:SLEW:APPLy" on page 550
- ":POWer:SLEW:SOURce" on page 551

:POWer:SWITch:APPLy

N (see page 1088)

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 554
- ":POWer:SWITch:IREFerence" on page 555
- ":POWer:SWITch:RDS" on page 556
- ":POWer:SWITch:VCE" on page 557
- ":POWer:SWITch:VREFerence" on page 558
- ":MEASure:ELOSs" on page 454
- ":MEASure:PLOSs" on page 461

:POWer:SWITch:CONDuction

N (see page 1088)

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 553
- ":POWer:SWITch:IREFerence" on page 555
- ":POWer:SWITch:RDS" on page 556
- ":POWer:SWITch:VCE" on page 557
- ":POWer:SWITch:VREFerence" on page 558

:POWer:SWITch:IREFerence

N (see page 1088)

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 553
- ":POWer:SWITch:CONDuction" on page 554
- ":POWer:SWITch:RDS" on page 556
- ":POWer:SWITch:VCE" on page 557
- ":POWer:SWITch:VREFerence" on page 558

:POWer:SWITch:RDS

N (see page 1088)

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 553
- ":POWer:SWITch:CONDuction" on page 554
- ":POWer:SWITch:IREFerence" on page 555
- ":POWer:SWITch:VCE" on page 557
- ":POWer:SWITch:VREFerence" on page 558

:POWer:SWITch:VCE

N (see page 1088)

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 553
- ":POWer:SWITch:CONDuction" on page 554
- ":POWer:SWITch:IREFerence" on page 555
- ":POWer:SWITch:RDS" on page 556
- ":POWer:SWITch:VREFerence" on page 558

:POWer:SWITch:VREFerence

N (see page 1088)

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 553
- ":POWer:SWITch:CONDuction" on page 554
- ":POWer:SWITch:IREFerence" on page 555
- ":POWer:SWITch:RDS" on page 556
- ":POWer:SWITch:VCE" on page 557

:POWer:TRANsient:APPLy

N (see page 1088)

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 560
- ":POWer:TRANsient:IINitial" on page 561
- ":POWer:TRANsient:INEW" on page 562
- ":POWer:TRANsient:NEXT" on page 563
- ":MEASure:TRESponse" on page 465

:POWer:TRANsient:EXIT

N (see page 1088)

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 559
- ":POWer:TRANsient:IINitial" on page 561
- ":POWer:TRANsient:INEW" on page 562
- ":POWer:TRANsient:NEXT" on page 563

:POWer:TRANsient:IINitial

N (see page 1088)

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" on page 547
- ":POWer:TRANsient:APPLy" on page 559
- ":POWer:TRANsient:EXIT" on page 560
- ":POWer:TRANsient:INEW" on page 562
- ":POWer:TRANsient:NEXT" on page 563

:POWer:TRANsient:INEW

N (see page 1088)

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 559
- ":POWer:TRANsient:EXIT" on page 560
- ":POWer:TRANsient:IINitial" on page 561
- ":POWer:TRANsient:NEXT" on page 563

:POWer:TRANsient:NEXT

N (see page 1088)

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 559
- ":POWer:TRANsient:EXIT" on page 560
- ":POWer:TRANsient:IINitial" on page 561
- ":POWer:TRANsient:INEW" on page 562

24 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

Table 95 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARBitrary:[START] [<file_spec>][,<column>] (see page 567)	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
:RECall:FILename <base_name> (see page 568)	:RECall:FILename? (see page 568)	<base_name> ::= quoted ASCII string
:RECall:MASK[:STARt] [<file_spec>] (see page 569)	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 570)	:RECall:PWD? (see page 570)	<path_name> ::= quoted ASCII string

Table 95 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:SETup[:STARt] [<file_spec>] (see page 571)	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>] (see page 572)	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

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 1088)

Command Syntax

```
:RECall:ARBitrary:[STARt] [<file_spec>][, <column>]

<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
```

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.

See Also

- "Introduction to :RECall Commands" on page 566
- ":RECall:FILename" on page 568
- ":SAVE:ARBitrary[:STARt]" on page 576

:RECall:FILename

N (see page 1088)

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 566
- ":RECall:SETup[:STARt]" on page 571
- ":SAVE:FILename" on page 577

:RECall:MASK[:STARt]

N (see page 1088)

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 566
- ":RECall:FILename" on page 568
- ":SAVE:MASK[:STARt]" on page 584
- ":MTESt:DATA" on page 482

:RECall:PWD

N (see page 1088)

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 566
- ":SAVE:PWD" on page 586

:RECall:SETup[:STARt]

N (see page 1088)

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 566
- ":RECall:FILename" on page 568
- ":SAVE:SETup[:STARt]" on page 587

:RECall:WMEMory<r>[:STARt]

N (see page 1088)

Command Syntax

```
:RECall:WMEMory<r>[:STARt] [<file_name>]

<r> ::= 1-2 in NR1 format

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

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".

See Also

- "Introduction to :RECall Commands" on page 566
- ":RECall:FILename" on page 568
- ":SAVE:WMEMory[:STARt]" on page 594

25 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "Introduction to :SAVE Commands" on page 575.

Table 96 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary:[STARt] [<file_spec>] (see page 576)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:FILename <base_name> (see page 577)	:SAVE:FILename? (see page 577)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_name>] (see page 578)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 579)	:SAVE:IMAGe:FACTors? (see page 579)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 580)	:SAVE:IMAGe:FORMat? (see page 580)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 581)	:SAVE:IMAGe:INKSaver? (see page 581)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 582)	:SAVE:IMAGe:PALette? (see page 582)	<palette> ::= {COLor | GRAYscale}
:SAVE:LISTer[:STARt] [<file_name>] (see page 583)	n/a	<file_name> ::= quoted ASCII string

Table 96 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:MASK[:STARt] [<file_spec>] (see page 584)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:POWer[:STARt] [<file_name>] (see page 585)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 586)	:SAVE:PWD? (see page 586)	<path_name> ::= quoted ASCII string
:SAVE:SETup[:STARt] [<file_spec>] (see page 587)	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 588)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 589)	:SAVE:WAVeform:FORMat? (see page 589)	<format> ::= {ALB | ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 590)	:SAVE:WAVeform:LENGth? (see page 590)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGth:MAX {{0 | OFF} | {1 | ON}} (see page 591)	:SAVE:WAVeform:LENGth:MAX? (see page 591)	{0 | 1}
:SAVE:WAVeform:SEGMented <option> (see page 592)	:SAVE:WAVeform:SEGMented? (see page 592)	<option> ::= {ALL | CURRent}

Table 96 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WMEMory:SOURce <source> (see page 593)	:SAVE:WMEMory:SOURce? (see page 593)	<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 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 594)	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]

N (see page 1088)

Command Syntax

```
:SAVE:ARBitrary:[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:ARBitrary:[STARt] command saves the current arbitrary waveform to an internal location or a file on a USB storage device.

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 575
- ":SAVE:FILename" on page 577
- ":RECall:ARBitrary[:STARt]" on page 567

:SAVE:FILename

N (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:SETup[:STARt]" on page 587
- ":SAVE:WAVeform[:STARt]" on page 588
- ":SAVE:PWD" on page 586
- ":RECall:FILename" on page 568

:SAVE:IMAGe[:STARt]

N (see page 1088)

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 575
- ":SAVE:IMAGe:FACTors" on page 579
- ":SAVE:IMAGe:FORMat" on page 580
- ":SAVE:IMAGe:INKSaver" on page 581
- ":SAVE:IMAGe:PALette" on page 582
- ":SAVE:FILename" on page 577

:SAVE:IMAGe:FACTors

N (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:IMAGe:FORMat" on page 580
- ":SAVE:IMAGe:INKSaver" on page 581
- ":SAVE:IMAGe:PALette" on page 582

:SAVE:IMAGe:FORMat

N (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:IMAGe:FACTors" on page 579
- ":SAVE:IMAGe:INKSaver" on page 581
- ":SAVE:IMAGe:PALette" on page 582
- ":SAVE:WAVeform:FORMat" on page 589

:SAVE:IMAGe:INKSaver

N (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:IMAGe:FACTors" on page 579
- ":SAVE:IMAGe:FORMat" on page 580
- ":SAVE:IMAGe:PALette" on page 582

:SAVE:IMAGe:PALette

N (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:IMAGe:FACTors" on page 579
- ":SAVE:IMAGe:FORMat" on page 580
- ":SAVE:IMAGe:INKSaver" on page 581

:SAVE:LISTer[:STARt]

N (see page 1088)

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 575
- ":SAVE:FILename" on page 577
- Chapter 17, ":LISTer Commands," starting on page 359

:SAVE:MASK[:STARt]

N (see page 1088)

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 575
- ":SAVE:FILename" on page 577
- ":RECall:MASK[:STARt]" on page 569
- ":MTESt:DATA" on page 482

:SAVE:POWer[:STARt]

N (see page 1088)

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 575
- ":SAVE:FILename" on page 577
- Chapter 23, ":POWer Commands," starting on page 507

:SAVE:PWD

N (see page 1088)

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 575
- ":SAVE:FILename" on page 577
- ":RECall:PWD" on page 570

:SAVE:SETup[:STARt]

N (see page 1088)

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 575
- ":SAVE:FILename" on page 577
- ":RECall:SETup[:STARt]" on page 571

:SAVE:WAVeform[:STARt]

N (see page 1088)

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 575
- ":SAVE:WAVeform:FORMat" on page 589
- ":SAVE:WAVeform:LENGth" on page 590
- ":SAVE:FILename" on page 577
- ":RECall:SETup[:STARt]" on page 571

:SAVE:WAVeform:FORMat

N (see page 1088)

Command Syntax

```
:SAVE:WAVeform:FORMat <format>

<format> ::= {ALB | ASCiixy | CSV | BINary}
```

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ALB — creates an Agilent module binary format file. These files can be viewed offline by the *Agilent Logic Analyzer* application software. The proper file extension for this format is ".alb".
- 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> ::= {ALB | 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 575
- ":SAVE:WAVeform[:STARt]" on page 588
- ":SAVE:WAVeform:LENGth" on page 590
- ":SAVE:IMAGe:FORMat" on page 580

:SAVE:WAVeform:LENGth

N (see page 1088)

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 575
- ":SAVE:WAVeform:LENGth:MAX" on page 591
- ":SAVE:WAVeform[:STARt]" on page 588
- ":WAVeform:POINts" on page 936
- ":SAVE:WAVeform:FORMat" on page 589

:SAVE:WAVeform:LENGth:MAX

N (see page 1088)

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 575
- ":SAVE:WAVeform[:STARt]" on page 588
- ":SAVE:WAVeform:LENGth" on page 590

:SAVE:WAVeform:SEGMented

N (see page 1088)

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 575
- ":SAVE:WAVeform[:STARt]" on page 588
- ":SAVE:WAVeform:FORMat" on page 589
- ":SAVE:WAVeform:LENGth" on page 590

:SAVE:WMEMory:SOURce

N (see page 1088)

Command Syntax

```
:SAVE:WMEMory:SOURce <source>

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

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

<r> ::= {1 | 2}
```

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 575
- ":SAVE:WMEMory[:STARt]" on page 594
- ":RECall:WMEMory<r>[:STARt]" on page 572

:SAVE:WMEMory[:STARt]

N (see page 1088)

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 575
- ":SAVE:WMEMory:SOURce" on page 593
- ":RECall:WMEMory<r>[:STARt]" on page 572

26 :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 851).

- **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.
- **I2S (Inter-IC Sound or Integrated Interchip Sound bus) triggering**– consists of connecting the oscilloscope to the serial clock, word select, and serial data lines, then triggering on a data value.

- **IIC (Inter-IC bus) triggering**– consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering**– will trigger on LIN sync break at the beginning of a message frame.You can trigger on Sync Break, Frame IDs, or Frame IDs and Data.
- **SPI (Serial Peripheral Interface) triggering**– consists of connecting the oscilloscope to a clock, data (MOSI or MISO), and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 64 bits long.
- **UART/RS-232 triggering** (with Option 232) – lets you trigger on RS-232 serial data.

NOTE Two I2S buses or two SPI buses cannot be decoded on both SBUS1 and SBUS2 at the same time.

Reporting the Setup

Use :SBUS<n>? to query setup information for the :SBUS<n> subsystem.

Return Format

The following is a sample response from the :SBUS1? query. In this case, the query was issued following a *RST command.

```
:SBUS1:DISP 0;MODE IIC;:SBUS1:IIC:ASIZ BIT7;:SBUS1:IIC:TRIG:TYPE
STAR;QUAL EQU;:SBUS1:IIC:SOUR:CLOC CHAN1;DATA
CHAN2;:SBUS1:IIC:TRIG:PATT:ADDR -1;DATA -1;DATA2 -1
```

General :SBUS<n> Commands

Table 97 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 598)	:SBUS<n>:DISPlay? (see page 598)	{0 | 1}
:SBUS<n>:MODE <mode> (see page 599)	:SBUS<n>:MODE? (see page 599)	<mode> ::= {A429 | CAN | FLEXray | I2S | IIC | LIN | M1553 | SPI | UART}

:SBUS<n>:DISPlay

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":CHANnel<n>:DISPlay" on page 258
- ":DIGital<d>:DISPlay" on page 283
- ":POD<n>:DISPlay" on page 503
- ":VIEW" on page 217
- ":BLANk" on page 190
- ":STATus" on page 214

:SBUS<n>:MODE

N (see page 1088)

Command Syntax `:SBUS<n>:MODE <mode>`

`<mode> ::= {A429 | FLEXray | CAN | I2S | IIC | LIN | M1553 | SPI | UART}`

The :SBUS<n>:MODE command determines the decode mode for the serial bus.

NOTE This command is only valid when a serial decode option has been licensed.

Query Syntax `:SBUS<n>:MODE?`

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Return Format `<mode><NL>`

`<mode> ::= {A429 | FLEX | CAN | I2S | IIC | LIN | M1553 | SPI | UART | NONE}`

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:A429 Commands" on page 600
- ":SBUS<n>:CAN Commands" on page 618
- ":SBUS<n>:FLEXray Commands" on page 635
- ":SBUS<n>:I2S Commands" on page 654
- ":SBUS<n>:IIC Commands" on page 673
- ":SBUS<n>:LIN Commands" on page 683
- ":SBUS<n>:M1553 Commands" on page 697
- ":SBUS<n>:SPI Commands" on page 704
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:A429 Commands

NOTE These commands are valid when the DSOX3AERO MIL-STD-1553 and ARINC 429 triggering and serial decode option (Option AERO) has been licensed.

Table 98 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUToset up (see page 602)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 603)	:SBUS<n>:A429:BASE? (see page 603)	<base> ::= {BINary | HEX}
n/a	:SBUS<n>:A429:COUNt:ERRor? (see page 604)	<error_count> ::= integer in NR1 format
:SBUS<n>:A429:COUNt:RESet (see page 605)	n/a	n/a
n/a	:SBUS<n>:A429:COUNt:WORD? (see page 606)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMat <format> (see page 607)	:SBUS<n>:A429:FORMat? (see page 607)	<format> ::= {LDSDi | LDSSm | LDATa}
:SBUS<n>:A429:SIGNal <signal> (see page 608)	:SBUS<n>:A429:SIGNal? (see page 608)	<signal> ::= {A | B | DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 609)	:SBUS<n>:A429:SOURce? (see page 609)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEed <speed> (see page 610)	:SBUS<n>:A429:SPEed? (see page 610)	<speed> ::= {LOW | HIGH}
:SBUS<n>:A429:TRIGger:LABel <value> (see page 611)	:SBUS<n>:A429:TRIGger:LABel? (see page 611)	<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}

Table 98 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :PATTern:DATA <string> (see page 612)	:SBUS<n>:A429:TRIGger :PATTern:DATA? (see page 612)	<string> ::= "nn...n" where n ::= {0 | 1 | X}, length depends on FORMat
:SBUS<n>:A429:TRIGger :PATTern:SDI <string> (see page 613)	:SBUS<n>:A429:TRIGger :PATTern:SDI? (see page 613)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :PATTern:SSM <string> (see page 614)	:SBUS<n>:A429:TRIGger :PATTern:SSM? (see page 614)	<string> ::= "nn" where n ::= {0 | 1 | X}, length always 2 bits
:SBUS<n>:A429:TRIGger :RANGe <min>,<max> (see page 615)	:SBUS<n>:A429:TRIGger :RANGe? (see page 615)	<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 616)	:SBUS<n>:A429:TRIGger :TYPE? (see page 616)	<condition> ::= {WSTArt | WSTOp | LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors | LRANge | ABITs | AOBits | AZBits}

:SBUS<n>:A429:AUTosetup

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:A429:BASE" on page 603
- ":SBUS<n>:A429:FORMat" on page 607
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429 Commands" on page 600

:SBUS<n>:A429:BASE

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:FORMat" on page 607

:SBUS<n>:A429:COUNt:ERRor

N (see page 1088)

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 1049

See Also
- ":SBUS<n>:A429:COUNt:RESet" on page 605
- ":SBUS<n>:A429:COUNt:WORD" on page 606
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429 Commands" on page 600

:SBUS<n>:A429:COUNt:RESet

N (see page 1088)

Command Syntax `:SBUS<n>:A429:COUNt:RESet`

Resets the word and error counters.

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:A429:COUNt:WORD" on page 606
- ":SBUS<n>:A429:COUNt:ERRor" on page 604
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429 Commands" on page 600

:SBUS<n>:A429:COUNt:WORD

N (see page 1088)

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 1049

See Also
- ":SBUS<n>:A429:COUNt:RESet" on page 605
- ":SBUS<n>:A429:COUNt:ERRor" on page 604
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429 Commands" on page 600

:SBUS<n>:A429:FORMat

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 612
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 613
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 614
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- ":SBUS<n>:A429:SIGNal" on page 608
- ":SBUS<n>:A429:SPEed" on page 610
- ":SBUS<n>:A429:BASE" on page 603
- ":SBUS<n>:A429:SOURce" on page 609

:SBUS<n>:A429:SIGNal

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:FORMat" on page 607
- ":SBUS<n>:A429:SPEed" on page 610
- ":SBUS<n>:A429:SOURce" on page 609

:SBUS<n>:A429:SOURce

N (see page 1088)

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 thresold levels for the selected source.

Return Format

```
<source><NL>
```

See Also

- ":TRIGger:LEVel:HIGH" on page 849
- ":TRIGger:LEVel:LOW" on page 850
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- ":SBUS<n>:A429:SIGNal" on page 608
- ":SBUS<n>:A429:SPEed" on page 610
- ":SBUS<n>:A429:FORMat" on page 607
- "Introduction to :TRIGger Commands" on page 843

:SBUS<n>:A429:SPEed

N (see page 1088)

Command Syntax

```
:SBUS<n>:A429:SPEed <speed>

<speed> ::= {LOW | HIGH}
```

The :SBUS<n>:A429:SPEed command specifies the signal speed:

- LOW — 12.5 kb/s.
- HIGH — 100 kb/s.

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}
```

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:SIGNal" on page 608
- ":SBUS<n>:A429:FORMat" on page 607
- ":SBUS<n>:A429:SOURce" on page 609

:SBUS<n>:A429:TRIGger:LABel

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616

:SBUS<n>:A429:TRIGger:PATTern:DATA

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 613
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 614

:SBUS<n>:A429:TRIGger:PATTern:SDI

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:FORMat" on page 607
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 612
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 614

:SBUS<n>:A429:TRIGger:PATTern:SSM

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:FORMat" on page 607
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 612
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 613

:SBUS<n>:A429:TRIGger:RANGe

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:TRIGger:TYPE" on page 616

:SBUS<n>:A429:TRIGger:TYPE

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:A429:TRIGger:LABel" on page 611
- ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 612
- ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 613
- ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 614
- ":SBUS<n>:A429:TRIGger:RANGe" on page 615

- ":SBUS<n>:A429:SOURce" on page 609

:SBUS<n>:CAN Commands

NOTE These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 99 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNt:ERRor? (see page 620)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:OVERload? (see page 621)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNt:RESet (see page 622)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNt:TOTal? (see page 623)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNt:UTILization? (see page 624)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:SAMPlepoint <value> (see page 625)	:SBUS<n>:CAN:SAMPlepoint? (see page 625)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:CAN:SIGNal:BAUDrate <baudrate> (see page 626)	:SBUS<n>:CAN:SIGNal:BAUDrate? (see page 626)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNal:DEFinition <value> (see page 627)	:SBUS<n>:CAN:SIGNal:DEFinition? (see page 627)	<value> ::= {CANH | CANL | RX | TX | DIFFerential | DIFL | DIFH}
:SBUS<n>:CAN:SOURce <source> (see page 628)	:SBUS<n>:CAN:SOURce? (see page 628)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> |} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:CAN:TRIGger <condition> (see page 629)	:SBUS<n>:CAN:TRIGger? (see page 630)	<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote | ALLerrors | OVERload | ACKerror}

Table 99 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:PATTern:DATA <string> (see page 631)	:SBUS<n>:CAN:TRIGger:PATTern:DATA? (see page 631)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth <length> (see page 632)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth? (see page 632)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 633)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 633)	<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 634)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 634)	<value> ::= {STANdard | EXTended}

:SBUS<n>:CAN:COUNt:ERRor

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 622
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN Commands" on page 618

:SBUS<n>:CAN:COUNt:OVERload

N (see page 1088)

Query Syntax `:SBUS<n>:CAN:COUNt:OVERload?`

Returns the overload frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 622
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN Commands" on page 618

:SBUS<n>:CAN:COUNt:RESet

N (see page 1088)

Command Syntax `:SBUS<n>:CAN:COUNt:RESet`

Resets the frame counters.

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:CAN:COUNt:ERRor" on page 620
- ":SBUS<n>:CAN:COUNt:OVERload" on page 621
- ":SBUS<n>:CAN:COUNt:TOTal" on page 623
- ":SBUS<n>:CAN:COUNt:UTILization" on page 624
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN Commands" on page 618

:SBUS<n>:CAN:COUNt:TOTal

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 622
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN Commands" on page 618

:SBUS<n>:CAN:COUNt:UTILization

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:CAN:COUNt:RESet" on page 622
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN Commands" on page 618

:SBUS<n>:CAN:SAMPlepoint

N (see page 1088)

Command Syntax

```
:SBUS<n>:CAN:SAMPlepoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :SBUS<n>:CAN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax

```
:SBUS<n>:CAN:SAMPlepoint?
```

The :SBUS<n>:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Return Format

```
<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:TRIGger" on page 629

:SBUS<n>:CAN:SIGNal:BAUDrate

N (see page 1088)

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 843
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:TRIGger" on page 629
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 627
- ":SBUS<n>:CAN:SOURce" on page 628

:SBUS<n>:CAN:SIGNal:DEFinition

N (see page 1088)

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 843
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:SIGNal:BAUDrate" on page 626
- ":SBUS<n>:CAN:SOURce" on page 628
- ":SBUS<n>:CAN:TRIGger" on page 629

:SBUS<n>:CAN:SOURce

N (see page 1088)

Command Syntax

```
:SBUS<n>:CAN:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:CAN:SOURce command sets the source for the CAN signal.

Query Syntax

```
:SBUS<n>:CAN:SOURce?
```

The :SBUS<n>:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:TRIGger" on page 629
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 627

:SBUS<n>:CAN:TRIGger

N (see page 1088)

Command Syntax

```
:SBUS<n>:CAN:TRIGger <condition>

<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote |
                 ALLerrors | OVERload | ACKerror}
```

The :SBUS<n>:CAN:TRIGger command sets the CAN trigger on condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- ERRor - will trigger on CAN Error frame.
- IDData - will trigger on CAN frames matching the specified Id of a Data frame.
- IDEither - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- IDRemote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
SOF	SOF - Start of Frame
DATA	ID & Data - Data Frame ID and Data
ERRor	Error - Error frame
IDData	ID & ~RTR - Data Frame ID (~RTR)
IDEither	ID - Remote or Data Frame ID
IDRemote	ID & RTR - Remote Frame ID (RTR)
ALLerrors	All Errors - All Errors
OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the :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 | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}
```

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 631
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 632
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 633
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 634
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 627
- ":SBUS<n>:CAN:SOURce" on page 628

:SBUS<n>:CAN:TRIGger:PATTern:DATA

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 632
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 633

:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 631
- ":SBUS<n>:CAN:SOURce" on page 628

:SBUS<n>:CAN:TRIGger:PATTern:ID

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 634
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 631

:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 631
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 632
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 633

:SBUS<n>:FLEXray Commands

NOTE These commands are only valid when the FLEXray triggering and serial decode option (Option FLEX) has been licensed.

Table 100 :SBUS<n>:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:AUTosetup (see page 637)	n/a	n/a
:SBUS<n>:FLEXray:BAUDrate <baudrate> (see page 638)	:SBUS<n>:FLEXray:BAUDrate? (see page 638)	<baudrate> ::= {2500000 | 5000000 | 10000000}
:SBUS<n>:FLEXray:CHANnel <channel> (see page 639)	:SBUS<n>:FLEXray:CHANnel? (see page 639)	<channel> ::= {A | B}
n/a	:SBUS<n>:FLEXray:COUNt:NULL? (see page 640)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:COUNt:RESet (see page 641)	n/a	n/a
n/a	:SBUS<n>:FLEXray:COUNt:SYNC? (see page 642)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:FLEXray:COUNt:TOTal? (see page 643)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:SOURce <source> (see page 644)	:SBUS<n>:FLEXray:SOURce? (see page 644)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format
:SBUS<n>:FLEXray:TRIGger <condition> (see page 645)	:SBUS<n>:FLEXray:TRIGger? (see page 645)	<condition> ::= {FRAMe | ERRor | EVENt}
:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE <error_type> (see page 646)	:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE? (see page 646)	<error_type> ::= {ALL | HCRC | FCRC}
:SBUS<n>:FLEXray:TRIGger:EVENt:AUTOset (see page 647)	n/a	n/a

Table 100 :SBUS<n>:FLEXray Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID <frame_id> (see page 648)	:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID? (see page 648)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE <event> (see page 649)	:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE? (see page 649)	<event> ::= {WAKeup | TSS | {FES | DTS} | BSS}
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase <cycle_count_base> (see page 650)	:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase? (see page 650)	<cycle_count_base> ::= integer from 0-63
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition <cycle_count_repetition> (see page 651)	:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition? (see page 651)	<cycle_count_repetition> ::= {ALL | <rep #>} <rep #> ::= integer values 2, 4, 8, 16, 32, or 64
:SBUS<n>:FLEXray:TRIGger:FRAMe:ID <frame_id> (see page 652)	:SBUS<n>:FLEXray:TRIGger:FRAMe:ID? (see page 652)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE <frame_type> (see page 653)	:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE? (see page 653)	<frame_type> ::= {NORMal | STARtup | NULL | SYNC | NSTArtup | NNULl | NSYNc | ALL}

:SBUS<n>:FLEXray:AUTosetup

N (see page 1088)

Command Syntax `:SBUS<n>:FLEXray:AUTosetup`

The :SBUS<n>:FLEXray:AUTosetup command automatically configures oscilloscope settings to facilitate FlexRay triggering and serial decode.

- Sets the selected source channel's impedance to 50 Ohms.
- Sets the selected source channel' s probe attenuation to 10:1.
- Sets the trigger level (on the selected source channel) to -300 mV.
- Turns on trigger Noise Reject.
- Turns on Serial Decode.
- Sets the trigger to the specified serial bus (n of SBUS<n>).

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger" on page 645
- ":SBUS<n>:FLEXray:BAUDrate" on page 638
- ":TRIGger[:EDGE]:LEVel" on page 868
- ":SBUS<n>:FLEXray:SOURce" on page 644

:SBUS<n>:FLEXray:BAUDrate

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:BAUDrate <baudrate>

<baudrate> ::= {2500000 | 5000000 | 10000000}
```

The :SBUS<n>:FLEXray:BAUDrate command specifies the baud rate as 2.5 Mb/s, 5 Mb/s, or 10 Mb/s.

Query Syntax

```
:SBUS<n>:FLEXray:BAUDrate?
```

The :SBUS<n>:FLEXray:BAUDrate? query returns the current baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= {2500000 | 5000000 | 10000000}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:CHANnel

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:CHANnel <channel>

<channel> ::= {A | B}
```

The :SBUS<n>:FLEXray:CHANnel command specifies the bus channel, A or B, of the FlexRay signal.

Query Syntax

```
:SBUS<n>:FLEXray:CHANnel?
```

The :SBUS<n>:FLEXray:CHANnel? query returns the current bus channel setting.

Return Format

```
<channel><NL>

<channel> ::= {A | B}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:COUNt:NULL

N (see page 1088)

Query Syntax `:SBUS<n>:FLEXray:COUNt:NULL?`

Returns the FlexRay null frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:FLEXray:COUNt:RESet" on page 641
- ":SBUS<n>:FLEXray:COUNt:TOTal" on page 643
- ":SBUS<n>:FLEXray:COUNt:SYNC" on page 642
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:COUNt:RESet

N (see page 1088)

Command Syntax `:SBUS<n>:FLEXray:COUNt:RESet`

Resets the FlexRay frame counters.

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:FLEXray:COUNt:NULL" on page 640
- ":SBUS<n>:FLEXray:COUNt:TOTal" on page 643
- ":SBUS<n>:FLEXray:COUNt:SYNC" on page 642
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:COUNt:SYNC

N (see page 1088)

Query Syntax

```
:SBUS<n>:FLEXray:COUNt:SYNC?
```

Returns the FlexRay sync frame count.

Return Format

```
<frame_count><NL>

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

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:FLEXray:COUNt:RESet" on page 641
- ":SBUS<n>:FLEXray:COUNt:TOTal" on page 643
- ":SBUS<n>:FLEXray:COUNt:NULL" on page 640
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:COUNt:TOTal

N (see page 1088)

Query Syntax `:SBUS<n>:FLEXray:COUNt:TOTal?`

Returns the FlexRay total frame count.

Return Format `<frame_count><NL>`

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

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:FLEXray:COUNt:RESet" on page 641
- ":SBUS<n>:FLEXray:COUNt:TOTal" on page 643
- ":SBUS<n>:FLEXray:COUNt:NULL" on page 640
- ":SBUS<n>:FLEXray:COUNt:SYNC" on page 642
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:FLEXray Commands" on page 635

:SBUS<n>:FLEXray:SOURce

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:SOURce <source>

<source> ::= {CHANnel<n>}

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

The :SBUS<n>:FLEXray:SOURce command specifies the input source for the FlexRay signal.

Query Syntax

```
:SBUS<n>:FLEXray:SOURce?
```

The :SBUS<n>:FLEXray:SOURce? query returns the current source for the FlexRay signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger" on page 645
- ":SBUS<n>:FLEXray:TRIGger:EVENt:TYPE" on page 649
- ":SBUS<n>:FLEXray:AUTosetup" on page 637

:SBUS<n>:FLEXray:TRIGger

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger <condition>

<condition> ::= {FRAMe | ERRor | EVENt}
```

The :SBUS<n>:FLEXray:TRIGger:TRIGger command sets the FLEXray trigger on condition:

- FRAMe — triggers on specified frames (without errors).
- ERRor — triggers on selected active error frames and unknown bus conditions.
- EVENt — triggers on specified FlexRay event/symbol.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger?
```

The :SBUS<n>:FLEXray:TRIGger? query returns the current FLEXray trigger on condition.

Return Format

```
<condition><NL>

<condition> ::= {FRAM | ERR | EVEN}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:FLEXray:TRIGger:ERRor:TYPE" on page 646
- ":SBUS<n>:FLEXray:TRIGger:EVENt:AUToset" on page 647
- ":SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID" on page 648
- ":SBUS<n>:FLEXray:TRIGger:EVENt:TYPE" on page 649
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase" on page 650
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 651
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:ID" on page 652
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE" on page 653

:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE <error_type>

<error_type> ::= {ALL | HCRC | FCRC}
```

Selects the FlexRay error type to trigger on. The error type setting is only valid when the FlexRay trigger mode is set to ERRor.

- ALL — triggers on ALL errors.
- HCRC — triggers on only Header CRC errors.
- FCRC — triggers on only Frame CRC errors.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE?
```

The :SBUS<n>:FLEXray:TRIGger:ERRor:TYPE? query returns the currently selected FLEXray error type.

Return Format

```
<error_type><NL>

<error_type> ::= {ALL | HCRC | FCRC}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:FLEXray:TRIGger:EVENt:AUToset

N (see page 1088)

Command Syntax `:SBUS<n>:FLEXray:TRIGger:EVENt:AUToset`

The :SBUS<n>:FLEXray:TRIGger:EVENt:AUToset command automatically configures oscilloscope settings (as shown on the display) for the selected event trigger.

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger:EVENt:TYPE" on page 649
- ":SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID" on page 648
- ":SBUS<n>:FLEXray:TRIGger" on page 645
- ":SBUS<n>:FLEXray:BAUDrate" on page 638
- ":TRIGger[:EDGE]:LEVel" on page 868
- ":SBUS<n>:FLEXray:SOURce" on page 644

:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID <frame_id>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

The :SBUS<N>:FLEXray:TRIGger:EVENt:BSS:ID command sets the frame ID used by the Byte Start Sequence (BSS) event trigger. This setting is only valid if the trigger mode is EVENt and the EVENt:TYPE is BSS.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID?
```

The :SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID? query returns the current frame ID setting for the Byte Start Sequence (BSS) event trigger setup.

Return Format

```
<frame_id><NL>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger:EVENt:TYPE" on page 649
- ":SBUS<n>:FLEXray:TRIGger:EVENt:AUToset" on page 647
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE <event>

<event> ::= {WAKeup | TSS | {FES | DTS} | BSS}
```

Selects the FlexRay event to trigger on. The event setting is only valid when the FlexRay trigger mode is set to EVENt.

- WAKeup — triggers on Wake-Up event.
- TSS — triggers on Transmission Start Sequence event.
- FES — triggers on either Frame End or Dynamic Trailing Sequence event.
- DTS — triggers on either Frame End or Dynamic Trailing Sequence event.
- BSS — triggers on Byte Start Sequence event.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE?
```

The :SBUS<n>:FLEXray:TRIGger:EVENt:TYPE? query returns the currently selected FLEXray event.

Return Format

```
<event><NL>

<event> ::= {WAK | TSS | {FES | DTS} | BSS}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger:EVENt:AUToset" on page 647
- ":SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID" on page 648
- ":SBUS<n>:FLEXray:TRIGger" on page 645
- ":SBUS<n>:FLEXray:AUTosetup" on page 637
- ":SBUS<n>:FLEXray:SOURce" on page 644

:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase <cycle_count_base>

<cycle_count_base> ::= integer from 0-63
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase command sets the base of the FlexRay cycle count (in the frame header) to trigger on. The cycle count base setting is only valid when the FlexRay trigger mode is set to FRAME.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase? query returns the current cycle count base setting for the FlexRay frame trigger setup.

Return Format

```
<cycle_count_base><NL>

<cycle_count_base> ::= integer from 0-63
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition <cycle_count_repetition>

<cycle_count_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer values 2, 4, 8, 16, 32, or 64
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition command sets the repetition number of the FlexRay cycle count (in the frame header) to trigger on. The cycle count repetition setting is only valid when the FlexRay trigger mode is set to FRAME.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition? query returns the current cycle count repetition setting for the FlexRay frame trigger setup.

Return Format

```
<cycle_count_repetition><NL>

<cycle_count_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer values 2, 4, 8, 16, 32, or 64
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:FLEXray:TRIGger:FRAMe:ID

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:ID <frame_id>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:ID command sets the FlexRay frame ID to trigger on. The frame ID setting is only valid when the FlexRay trigger mode is set to FRAMe.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:ID?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:ID? query returns the current frame ID setting for the FlexRay frame trigger setup.

Return Format

```
<frame_id><NL>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE

N (see page 1088)

Command Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE <frame_type>

<frame_type> ::= {NORMal | STARtup | NULL | SYNC | NSTArtup | NNULl |
                  NSYNc | ALL}
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE command sets the FlexRay frame type to trigger on. The frame type setting is only valid when the FlexRay trigger mode is set to FRAME.

- NORMal – will trigger on only normal (NSTArtup & NNULl & NSYNc) frames.
- STARtup – will trigger on only startup frames.
- NULL – will trigger on only null frames.
- SYNC – will trigger on only sync frames.
- NSTArtup – will trigger on frames other than startup frames.
- NNULl – will trigger on frames other than null frames.
- NSYNc – will trigger on frames other than sync frames.
- ALL – will trigger on all FlexRay frame types.

Query Syntax

```
:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE?
```

The :SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE? query returns the current frame type setting for the FlexRay frame trigger setup.

Return Format

```
<frame_type><NL>

<frame_type> ::= {NORM | STAR | NULL | SYNC | NSTA | NNUL |
                  NSYN | ALL}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:FLEXray:TRIGger" on page 645

:SBUS<n>:I2S Commands

NOTE These commands are only valid when the I2S serial decode option (Option SND) has been licensed.

Table 101 :SBUS<n>:I2S Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:I2S:ALIGnment <setting> (see page 656)	:SBUS<n>:I2S:ALIGnment? (see page 656)	<setting> ::= {I2S | LJ | RJ}
:SBUS<n>:I2S:BASE <base> (see page 657)	:SBUS<n>:I2S:BASE? (see page 657)	<base> ::= {DECimal | HEX}
:SBUS<n>:I2S:CLOCk:SLOPe <slope> (see page 658)	:SBUS<n>:I2S:CLOCk:SLOPe? (see page 658)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:I2S:RWIDth <receiver> (see page 659)	:SBUS<n>:I2S:RWIDth? (see page 659)	<receiver> ::= 4-32 in NR1 format
:SBUS<n>:I2S:SOURce:CLOCk <source> (see page 660)	:SBUS<n>:I2S:SOURce:CLOCk? (see page 660)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:I2S:SOURce:DATA <source> (see page 661)	:SBUS<n>:I2S:SOURce:DATA? (see page 661)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:I2S:SOURce:WSELect <source> (see page 662)	:SBUS<n>:I2S:SOURce:WSELect? (see page 662)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 101 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:TRIGger <operator> (see page 663)	:SBUS<n>:I2S:TRIGger? (see page 663)	<operator> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange | OUTRange | INCReasing | DECReasing}
:SBUS<n>:I2S:TRIGger:AUDio <audio_ch> (see page 665)	:SBUS<n>:I2S:TRIGger:AUDio? (see page 665)	<audio_ch> ::= {RIGHt | LEFT | EITHer}
:SBUS<n>:I2S:TRIGger:PATTern:DATA <string> (see page 666)	:SBUS<n>:I2S:TRIGger:PATTern:DATA? (see page 667)	<string> ::= "n" where n ::= 32-bit integer in signed 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>:I2S:TRIGger:PATTern:FORMat <base> (see page 668)	:SBUS<n>:I2S:TRIGger:PATTern:FORMat? (see page 668)	<base> ::= {BINary | HEX | DECimal}
:SBUS<n>:I2S:TRIGger:RANGe <lower>,<upper> (see page 669)	:SBUS<n>:I2S:TRIGger:RANGe? (see page 669)	<lower> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SBUS<n>:I2S:TWIDth <word_size> (see page 671)	:SBUS<n>:I2S:TWIDth? (see page 671)	<word_size> ::= 4-32 in NR1 format
:SBUS<n>:I2S:WSLow <low_def> (see page 672)	:SBUS<n>:I2S:WSLow? (see page 672)	<low_def> ::= {LEFT | RIGHt}

:SBUS<n>:I2S:ALIGnment

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:ALIGnment <setting>

<setting> ::= {I2S | LJ | RJ}
```

The :SBUS<n>:I2S:ALIGnment command selects the data alignment of the I2S bus for the serial decoder and/or trigger when in I2S mode:

- I2S — standard.
- LJ — left justified.
- RJ — right justified.

Note that the word select (WS) polarity is specified separately with the :SBUS<n>:I2S:WSLow command.

Query Syntax

```
:SBUS<n>:I2S:ALIGnment?
```

The :SBUS<n>:I2S:ALIGnment? query returns the currently selected I2S data alignment.

Return Format

```
<setting><NL>

<setting> ::= {I2S | LJ | RJ}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:CLOCk:SLOPe" on page 658
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:TWIDth" on page 671
- ":SBUS<n>:I2S:WSLow" on page 672

:SBUS<n>:I2S:BASE

N (see page 1088)

Command Syntax `:SBUS<n>:I2S:BASE <base>`

`<base> ::= {DECimal | HEX}`

The :SBUS<n>:I2S:BASE command determines the base to use for the I2S decode display.

Query Syntax `:SBUS<n>:I2S:BASE?`

The :SBUS<n>:I2S:BASE? query returns the current I2S display decode base.

Return Format `<base><NL>`

`<base> ::= {DECimal | HEX}`

Errors
- "-241, Hardware missing" on page 1049

See Also
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:I2S Commands" on page 654

:SBUS<n>:I2S:CLOCk:SLOPe

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:CLOCk:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :SBUS<n>:I2S:CLOCk:SLOPe command specifies which edge of the I2S serial clock signal clocks in data.

- NEGative — Falling edge.
- POSitive — Rising edge.

Query Syntax

```
:SBUS<n>:I2S:CLOCk:SLOPe?
```

The :SBUS<n>:I2S:CLOCk:SLOPe? query returns the current I2S clock slope setting.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:ALIGnment" on page 656
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:TWIDth" on page 671
- ":SBUS<n>:I2S:WSLow" on page 672

:SBUS<n>:I2S:RWIDth

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:RWIDth <receiver>

<receiver> ::= 4-32 in NR1 format
```

The :SBUS<n>:I2S:RWIDth command sets the width of the receiver (decoded) data word in I2S anywhere from 4 bits to 32 bits.

Query Syntax

```
:SBUS<n>:I2S:RWIDth?
```

The :SBUS<n>:I2S:RWIDth? query returns the currently set I2S receiver data word width.

Return Format

```
<receiver><NL>

<receiver> ::= 4-32 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:ALIGnment" on page 656
- ":SBUS<n>:I2S:CLOCk:SLOPe" on page 658
- ":SBUS<n>:I2S:TWIDth" on page 671
- ":SBUS<n>:I2S:WSLow" on page 672

:SBUS<n>:I2S:SOURce:CLOCk

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:I2S:SOURce:CLOCk controls which signal is used as the serial clock (SCLK) source by the serial decoder and/or trigger when in I2S mode.

Query Syntax

```
:SBUS<n>:I2S:SOURce:CLOCk?
```

The :SBUS<n>:I2S:SOURce:CLOCk? query returns the current source for the I2S serial clock (SCLK).

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:SOURce:DATA" on page 661
- ":SBUS<n>:I2S:SOURce:WSELect" on page 662

:SBUS<n>:I2S:SOURce:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:SOURce:DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:I2S:SOURce:DATA command controls which signal is used as the serial data (SDATA) source by the serial decoder and/or trigger when in I2S mode.

Query Syntax

```
:SBUS<n>:I2S:SOURce:DATA?
```

The :SBUS<n>:I2S:SOURce:DATA? query returns the current source for the I2S serial data (SDATA).

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:SOURce:CLOCk" on page 660
- ":SBUS<n>:I2S:SOURce:WSELect" on page 662

:SBUS<n>:I2S:SOURce:WSELect

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:SOURce:WSELect <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:I2S:SOURce:WSELect command controls which signal is used as the word select (WS) source by the serial decoder and/or trigger when in I2S mode.

Query Syntax

```
:SBUS<n>:I2S:SOURce:WSELect?
```

The :SBUS<n>:I2S:SOURce:WSELect? query returns the current source for I2S word select (WS).

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:SOURce:CLOCk" on page 660
- ":SBUS<n>:I2S:SOURce:DATA" on page 661

:SBUS<n>:I2S:TRIGger

N (see page 1088)

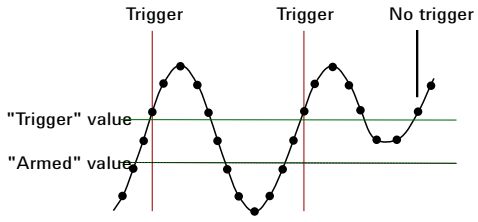
Command Syntax

```
:SBUS<n>:I2S:TRIGger <operator>

<operator> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange
               | OUTRange | INCReasing | DECReasing}
```

The :SBUS<n>:I2S:TRIGger command sets the I2S trigger operator:

- EQUal– triggers on the specified audio channel's data word when it equals the specified word.
- NOTequal – triggers on any word other than the specified word.
- LESSthan – triggers when the channel's data word is less than the specified value.
- GREaterthan – triggers when the channel's data word is greater than the specified value.
- INRange – enter upper and lower values to specify the range in which to trigger.
- OUTRange – enter upper and lower values to specify range in which trigger will not occur.
- INCReasing – triggers when the data value makes a certain increase over time and the specified value is met or exceeded. Use the :SBUS<n>:I2S:TRIGger:RANGe command to set "Trigger" and "Armed" values. The "Trigger" value is the value that must be met or exceeded to cause the trigger. The "Armed" value is the value the data must go below in order to re-arm the oscilloscope (ready it to trigger again).



- DECReasing – similar to INCReasing except the trigger occurs on a certain descrease over time and the "Trigger" data value is less than the "Armed" data value.

Query Syntax

```
:SBUS<n>:I2S:TRIGger?
```

The :SBUS<n>:I2S:TRIGger? query returns the current I2S trigger operator.

Return Format

```
<operator><NL>

<operator> ::= {EQU | NOT | LESS | GRE | INR | OUTR | INCR | DECR}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:TRIGger:AUDio" on page 665
- ":SBUS<n>:I2S:TRIGger:RANGe" on page 669
- ":SBUS<n>:I2S:TRIGger:PATTern:FORMat" on page 668

:SBUS<n>:I2S:TRIGger:AUDio

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:TRIGger:AUDio <audio_ch>

<audio_ch> ::= {RIGHt | LEFT | EITHer}
```

The :SBUS<n>:I2S:TRIGger:AUDio command specifies the audio channel to trigger on:

- RIGHt – right channel.
- LEFT– left channel.
- EITHer – right or left channel.

Query Syntax

```
:SBUS<n>:I2S:TRIGger:AUDio?
```

The :SBUS<n>:I2S:TRIGger:AUDio? query returns the current audio channel for the I2S trigger.

Return Format

```
<audio_ch><NL>

<audio_ch> ::= {RIGH | LEFT | EITH}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:TRIGger" on page 663

:SBUS<n>:I2S:TRIGger:PATTern:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:TRIGger:PATTern:DATA <string>

<string> ::= "n" where n ::= 32-bit integer in signed 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>:I2S:TRIGger:PATTern:FORMat command. The default <base> is DECimal.

The :SBUS<n>:I2S:TRIGger:PATTern:DATA command specifies the I2S trigger data pattern searched for in each I2S message.

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 :SBUS<n>:I2S:TRIGger:PATTern:DATA command specifies the I2S trigger data pattern used by the EQUal, NOTequal, GREaterthan, and LESSthan trigger conditions. If the GREaterthan or LESSthan trigger condition is selected, the bits specified to be masked off ("X") will be interpreted as 0's.

NOTE The length of the trigger data value is determined by the :SBUS<n>:I2S:RWIDth and :SBUS<n>:I2S:TWIDth commands. When the receiver word size is less than the transmitter word size, the data length is equal to the receiver word size. When the receiver word size is greater than the transmitter word size, the data length is equal to the transmitter word size.

NOTE If more bits are sent for <string> than the specified trigger data length, the most significant bits will be truncated. If the word 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>:I2S:TRIGger:PATTern:DATA?
```

The :SBUS<n>:I2S:TRIGger:PATTern:DATA? query returns the currently specified I2S trigger data pattern.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:TRIGger:PATTern:FORMat" on page 668
- ":SBUS<n>:I2S:TRIGger" on page 663
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:TWIDth" on page 671
- ":SBUS<n>:I2S:TRIGger:AUDio" on page 665

:SBUS<n>:I2S:TRIGger:PATTern:FORMat

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:TRIGger:PATTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}
```

The :SBUS<n>:I2S:TRIGger:PATTern:FORMat command sets the entry (and query) number base used by the :SBUS<n>:I2S:TRIGger:PATTern:DATA command. The default <base> is DECimal.

Query Syntax

```
:SBUS<n>:I2S:TRIGger:PATTern:FORMat?
```

The :SBUS<n>:I2S:TRIGger:PATTern:FORMat? query returns the currently set number base for I2S pattern data.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX | DEC}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:TRIGger:AUDio" on page 665
- ":SBUS<n>:I2S:TRIGger" on page 663

:SBUS<n>:I2S:TRIGger:RANGe

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:TRIGger:RANGe <lower>,<upper>

<lower> ::= 32-bit integer in signed decimal, <nondecimal>
            or <string>

<upper> ::= 32-bit integer in signed decimal, <nondecimal>,
            or <string>

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

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

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

The :SBUS<n>:I2S:TRIGger:RANGe command sets the lower and upper range boundaries used by the INRange, OUTRange, INCReasing, and DECReasing trigger conditions. You can enter the parameters in any order – the smaller value becomes the <lower> and the larger value becomes the <upper>.

Note that for INCReasing and DECReasing, the <lower> and <upper> values correspond to the "Armed" and "Trigger" softkeys.

NOTE

The length of the <lower> and <upper> values is determined by the :SBUS<n>:I2S:RWIDth and :SBUS<n>:I2S:TWIDth commands. When the receiver word size is less than the transmitter word size, the length is equal to the receiver word size. When the receiver word size is greater than the transmitter word size, the length is equal to the transmitter word size.

Query Syntax

```
:SBUS<n>:I2S:TRIGger:RANGe?
```

The :SBUS<n>:I2S:TRIGger:RANGe? query returns the currently set lower and upper range boundaries.

Return Format

```
<lower>,<upper><NL>

<lower> ::= 32-bit integer in signed decimal, <nondecimal>
            or <string>

<upper> ::= 32-bit integer in signed decimal, <nondecimal>,
            or <string>

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

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

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

See Also

- "Introduction to :TRIGger Commands" on page 843

- ":SBUS<n>:I2S:TRIGger" on page 663
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:TWIDth" on page 671
- ":SBUS<n>:I2S:WSLow" on page 672

:SBUS<n>:I2S:TWIDth

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:TWIDth <word_size>

<word_size> ::= 4-32 in NR1 format
```

The :SBUS<n>:I2S:TWIDth command sets the width of the transmitted data word in I2S anywhere from 4 bits to 32 bits.

Query Syntax

```
:SBUS<n>:I2S:TWIDth?
```

The :SBUS<n>:I2S:TWIDth? query returns the currently set I2S transmitted data word width.

Return Format

```
<word_size><NL>

<word_size> ::= 4-32 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:ALIGnment" on page 656
- ":SBUS<n>:I2S:CLOCk:SLOPe" on page 658
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:WSLow" on page 672

:SBUS<n>:I2S:WSLow

N (see page 1088)

Command Syntax

```
:SBUS<n>:I2S:WSLow <low_def>

<low_def> ::= {LEFT | RIGHt}
```

The :SBUS<n>:I2S:WSLow command selects the polarity of the word select (WS) signal:

- LEFT– a word select (WS) state of low indicates left channel data is active on the I2S bus, and a WS state of high indicates right channel data is active on the bus.
- RIGHt – a word select (WS) state of low indicates right channel data is active on the I2S bus, and a WS state of high indicates left channel data is active on the bus.

Query Syntax

```
:SBUS<n>:I2S:WSLow?
```

The :SBUS<n>:I2S:WSLow? query returns the currently selected I2S word select (WS) polarity.

Return Format

```
<low_def><NL>

<low_def> ::= {LEFT | RIGHt}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:I2S:ALIGnment" on page 656
- ":SBUS<n>:I2S:CLOCk:SLOPe" on page 658
- ":SBUS<n>:I2S:RWIDth" on page 659
- ":SBUS<n>:I2S:TWIDth" on page 671

: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 102 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZe <size> (see page 674)	:SBUS<n>:IIC:ASIZe? (see page 674)	<size> ::= {BIT7 | BIT8}
:SBUS<n>:IIC[:SOURce] :CLOCk <source> (see page 675)	:SBUS<n>:IIC[:SOURce] :CLOCk? (see page 675)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC[:SOURce] :DATA <source> (see page 676)	:SBUS<n>:IIC[:SOURce] :DATA? (see page 676)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:IIC:TRIGger: PATTern:ADDRess <value> (see page 677)	:SBUS<n>:IIC:TRIGger: PATTern:ADDRess? (see page 677)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger: PATTern:DATA <value> (see page 678)	:SBUS<n>:IIC:TRIGger: PATTern:DATA? (see page 678)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger: PATTern:DATa2 <value> (see page 679)	:SBUS<n>:IIC:TRIGger: PATTern:DATa2? (see page 679)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SBUS<n>:IIC:TRIGger: QUALifier <value> (see page 680)	:SBUS<n>:IIC:TRIGger: QUALifier? (see page 680)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:SBUS<n>:IIC:TRIGger[:TYPE] <type> (see page 681)	:SBUS<n>:IIC:TRIGger[:TYPE]? (see page 681)	<type> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10 | NACKnowledge | ANACk | R7Data2 | W7Data2 | RESTart}

:SBUS<n>:IIC:ASIZe

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:IIC Commands" on page 673

:SBUS<n>:IIC[:SOURce]:CLOCk

N (see page 1088)

Command Syntax

```
:SBUS<n>:IIC:[SOURce:]CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:IIC:[SOURce:]CLOCk command sets the source for the IIC serial clock (SCL).

Query Syntax

```
:SBUS<n>:IIC:[SOURce:]CLOCk?
```

The :SBUS<n>:IIC:[SOURce:]CLOCk? query returns the current source for the IIC serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:IIC[:SOURce]:DATA" on page 676

:SBUS<n>:IIC[:SOURce]:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:IIC:[SOURce:]DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:IIC:[SOURce:]DATA command sets the source for IIC serial data (SDA).

Query Syntax

```
:SBUS<n>:IIC:[SOURce:]DATA?
```

The :SBUS<n>:IIC:[SOURce:]DATA? query returns the current source for IIC serial data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:IIC[:SOURce]:CLOCk" on page 675

:SBUS<n>:IIC:TRIGger:PATTern:ADDRess

N (see page 1088)

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 843
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 678
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 679
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 681

:SBUS<n>:IIC:TRIGger:PATTern:DATA

N (see page 1088)

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 843
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 677
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 679
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 681

:SBUS<n>:IIC:TRIGger:PATTern:DATa2

N (see page 1088)

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 843
- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 677
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 678
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 681

:SBUS<n>:IIC:TRIGger:QUALifier

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 681

:SBUS<n>:IIC:TRIGger[:TYPE]

N (see page 1088)

Command Syntax

```
:SBUS<n>:IIC:TRIGger[:TYPE] <value>

<value> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10
| NACKnowledge | ANACk | R7Data2 | W7Data2 | RESTart}
```

The :SBUS<n>:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- STARt — Start condition.
- STOP — Stop condition.
- READ7 — 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- R7Data2 — 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- READEprom — EEPROM data read.
- WRITe7 — 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- W7Data2 — 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- WRITe10 — 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- NACKnowledge — Missing acknowledge.
- ANACk — Address with no acknowledge.
- RESTart — Another start condition occurs before a stop condition.

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 1090).

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 | READ7 | READE | WRIT7 | WRIT10 | NACK | ANAC
            | R7D2 | W7D2 | REST}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851

- ":SBUS<n>:IIC:TRIGger:PATTern:ADDRess" on page 677
- ":SBUS<n>:IIC:TRIGger:PATTern:DATA" on page 678
- ":SBUS<n>:IIC:TRIGger:PATTern:DATa2" on page 679
- ":SBUS<n>:IIC:TRIGger:QUALifier" on page 680
- "Long Form to Short Form Truncation Rules" on page 1090

:SBUS<n>:LIN Commands

NOTE These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 103 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 685)	:SBUS<n>:LIN:PARity? (see page 685)	{0 | 1}
:SBUS<n>:LIN:SAMPlepoint <value> (see page 686)	:SBUS<n>:LIN:SAMPlepoint? (see page 686)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:SBUS<n>:LIN:SIGNal:BAUDrate <baudrate> (see page 687)	:SBUS<n>:LIN:SIGNal:BAUDrate? (see page 687)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 688)	:SBUS<n>:LIN:SOURce? (see page 688)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:LIN:STANdard <std> (see page 689)	:SBUS<n>:LIN:STANdard? (see page 689)	<std> ::= {LIN13 | LIN20}
:SBUS<n>:LIN:SYNCbreak <value> (see page 690)	:SBUS<n>:LIN:SYNCbreak? (see page 690)	<value> ::= integer = {11 | 12 | 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 691)	:SBUS<n>:LIN:TRIGger? (see page 691)	<condition> ::= {SYNCbreak | ID | DATA}
:SBUS<n>:LIN:TRIGger:ID <value> (see page 692)	:SBUS<n>:LIN:TRIGger:ID? (see page 692)	<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 103 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger:PATTern:DATA <string> (see page 693)	:SBUS<n>:LIN:TRIGger:PATTern:DATA? (see page 693)	<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 695)	:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth? (see page 695)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger:PATTern:FORMat <base> (see page 696)	:SBUS<n>:LIN:TRIGger:PATTern:FORMat? (see page 696)	<base> ::= {BINary | HEX | DECimal}

:SBUS<n>:LIN:PARity

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:LIN Commands" on page 683

:SBUS<n>:LIN:SAMPlepoint

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:TRIGger" on page 691

:SBUS<n>:LIN:SIGNal:BAUDrate

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:TRIGger" on page 691
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:SOURce

N (see page 1088)

Command Syntax

```
:SBUS<n>:LIN:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:LIN:SOURce command sets the source for the LIN signal.

Query Syntax

```
:SBUS<n>:LIN:SOURce?
```

The :SBUS<n>:LIN:SOURce? query returns the current source for the LIN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:TRIGger" on page 691
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042

:SBUS<n>:LIN:STANdard

N (see page 1088)

Command Syntax

```
:SBUS<n>:LIN:STANdard <std>

<std> ::= {LIN13 | LIN20}
```

The :SBUS<n>:LIN:STANdard command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

Query Syntax

```
:SBUS<n>:LIN:STANdard?
```

The :SBUS<n>:LIN:STANdard? query returns the current LIN standard setting.

Return Format

```
<std><NL>

<std> ::= {LIN13 | LIN20}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:SYNCbreak

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:TRIGger

N (see page 1088)

Command Syntax

```
:SBUS<n>:LIN:TRIGger <condition>

<condition> ::= {SYNCbreak | ID | DATA}
```

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.

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}
```

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:TRIGger:ID" on page 692
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 695
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 693
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:TRIGger:ID

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:TRIGger" on page 691
- ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1042
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:TRIGger:PATTern:DATA

N (see page 1088)

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 843
- ":SBUS<n>:LIN:TRIGger:PATTern:FORMat" on page 696
- ":SBUS<n>:LIN:TRIGger" on page 691
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 695

:SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 693
- ":SBUS<n>:LIN:SOURce" on page 688

:SBUS<n>:LIN:TRIGger:PATTern:FORMat

N (see page 1088)

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 843
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA" on page 693
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGth" on page 695

:SBUS<n>:M1553 Commands

NOTE These commands are valid when the DSOX3AERO MIL-STD-1553 and ARINC 429 triggering and serial decode option (Option AERO) has been licensed.

Table 104 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTosetup (see page 698)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 699)	:SBUS<n>:M1553:BASE? (see page 699)	<base> ::= {BINary | HEX}
:SBUS<n>:M1553:SOURce <source> (see page 700)	:SBUS<n>:M1553:SOURce? (see page 700)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGger:PATTern:DATA <string> (see page 701)	:SBUS<n>:M1553:TRIGger:PATTern:DATA? (see page 701)	<string> ::= "nn...n" where n ::= {0 | 1 | X}
:SBUS<n>:M1553:TRIGger:RTA <value> (see page 702)	:SBUS<n>:M1553:TRIGger:RTA? (see page 702)	<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 703)	:SBUS<n>:M1553:TRIGger:TYPE? (see page 703)	<type> ::= {DSTArt | DSTOp | CSTArt | CSTOp | RTA | PERRor | SERRor | MERRor | RTA11}

:SBUS<n>:M1553:AUTosetup

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:M1553:SOURce" on page 700

:SBUS<n>:M1553:BASE

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:M1553 Commands" on page 697

:SBUS<n>:M1553:SOURce

N (see page 1088)

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 thresold 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 849
- ":TRIGger:LEVel:LOW" on page 850
- ":TRIGger:MODE" on page 851
- "Introduction to :TRIGger Commands" on page 843

:SBUS<n>:M1553:TRIGger:PATTern:DATA

N (see page 1088)

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 843
- ":SBUS<n>:M1553:TRIGger:TYPE" on page 703
- ":SBUS<n>:M1553:TRIGger:RTA" on page 702

:SBUS<n>:M1553:TRIGger:RTA

N (see page 1088)

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 843
- ":SBUS<n>:M1553:TRIGger:TYPE" on page 703

:SBUS<n>:M1553:TRIGger:TYPE

N (see page 1088)

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 Comamnd/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 specifed as a hex value, and the remaining 11 bits can be specifed 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 843
- ":SBUS<n>:M1553:TRIGger:RTA" on page 702
- ":SBUS<n>:M1553:TRIGger:PATTern:DATA" on page 701
- ":TRIGger:MODE" on page 851

:SBUS<n>:SPI Commands

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Table 105 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 706)	:SBUS<n>:SPI:BITorder? (see page 706)	<order> ::= {LSBFirst | MSBFirst}
:SBUS<n>:SPI:CLOCk:SLOPe <slope> (see page 707)	:SBUS<n>:SPI:CLOCk:SLOPe? (see page 707)	<slope> ::= {NEGative | POSitive}
:SBUS<n>:SPI:CLOCk:TIMeout <time_value> (see page 708)	:SBUS<n>:SPI:CLOCk:TIMeout? (see page 708)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 709)	:SBUS<n>:SPI:FRAMing? (see page 709)	<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
:SBUS<n>:SPI:SOURce:CLOCk <source> (see page 710)	:SBUS<n>:SPI:SOURce:CLOCk? (see page 710)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:DATA <source> (see page 711)	:SBUS<n>:SPI:SOURce:DATA? (see page 711)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:FRAMe <source> (see page 712)	:SBUS<n>:SPI:SOURce:FRAMe? (see page 712)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 105 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:SOURce:MISO <source> (see page 713)	:SBUS<n>:SPI:SOURce:MISO? (see page 713)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:SOURce:MOSI <source> (see page 714)	:SBUS<n>:SPI:SOURce:MOSI? (see page 714)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string> (see page 715)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 715)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width> (see page 716)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? (see page 716)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string> (see page 717)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? (see page 717)	<string> ::= "nn...n" where n ::= {0 | 1 | X | $} <string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width> (see page 718)	:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? (see page 718)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 719)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 719)	<value> ::= {MOSI | MISO}
:SBUS<n>:SPI:WIDTh <word_width> (see page 720)	:SBUS<n>:SPI:WIDTh? (see page 720)	<word_width> ::= integer 4-16 in NR1 format

:SBUS<n>:SPI:BITorder

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:BITorder <order>

<order> ::= {LSBFirst | MSBFirst}
```

The :SBUS<n>:SPI:BITorder command selects the bit order, most significant bit first (MSB) or least significant bit first (LSB), used when displaying data in the serial decode waveform and in the Lister.

Query Syntax

```
:SBUS<n>:SPI:BITorder?
```

The :SBUS<n>:SPI:BITorder? query returns the current SPI decode bit order.

Return Format

```
<order><NL>

<order> ::= {LSBF | MSBF}
```

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:SPI Commands" on page 704

:SBUS<n>:SPI:CLOCk:SLOPe

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:CLOCk:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :SBUS<n>:SPI:CLOCk:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

Query Syntax

```
:SBUS<n>:SPI:CLOCk:SLOPe?
```

The :SBUS<n>:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 708
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710

:SBUS<n>:SPI:CLOCk:TIMeout

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:CLOCk:TIMeout <time_value>

<time_value> ::= time in seconds in NR3 format
```

The :SBUS<n>:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 100 ns to 10 s when the :SBUS<n>:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

Query Syntax

```
:SBUS<n>:SPI:CLOCk:TIMeout?
```

The :SBUS<n>:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Return Format

```
<time value><NL>

<time_value> ::= time in seconds in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:CLOCk:SLOPe" on page 707
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710
- ":SBUS<n>:SPI:FRAMing" on page 709

:SBUS<n>:SPI:FRAMing

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:FRAMing <value>

<value> ::= {CHIPselect | {NCHipselect | NOTC} | TIMeout}
```

The :SBUS<n>:SPI:FRAMing command sets the SPI trigger framing value. If TIMeout is selected, the timeout value is set by the :SBUS<n>:SPI:CLOCk:TIMeout command.

NOTE The NOTC value is deprecated. It is the same as NCHipselect.

Query Syntax

```
:SBUS<n>:SPI:FRAMing?
```

The :SBUS<n>:SPI:FRAMing? query returns the current SPI framing value.

Return Format

```
<value><NL>

<value> ::= {CHIP | NCH | TIM}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 708
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 712

:SBUS<n>:SPI:SOURce:CLOCk

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

Query Syntax

```
:SBUS<n>:SPI:SOURce:CLOCk?
```

The :SBUS<n>:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:CLOCk:SLOPe" on page 707
- ":SBUS<n>:SPI:CLOCk:TIMeout" on page 708
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 712
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:MISO" on page 713
- ":SBUS<n>:SPI:SOURce:DATA" on page 711

:SBUS<n>:SPI:SOURce:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:SOURce:DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:SPI:SOURce:DATA command sets the source for the SPI serial MOSI data.

This command is the same as the :SBUS<n>:SPI:SOURce:MOSI command.

Query Syntax

```
:SBUS<n>:SPI:SOURce:DATA?
```

The :SBUS<n>:SPI:SOURce:DATA? query returns the current source for the SPI serial MOSI data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:MISO" on page 713
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 712
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 715
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 717
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 716
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 718

:SBUS<n>:SPI:SOURce:FRAMe

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:SPI:SOURce:FRAMe command sets the frame source when :SBUS<n>:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax

```
:SBUS<n>:SPI:SOURce:FRAMe?
```

The :SBUS<n>:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:MISO" on page 713
- ":SBUS<n>:SPI:SOURce:DATA" on page 711
- ":SBUS<n>:SPI:FRAMing" on page 709

:SBUS<n>:SPI:SOURce:MISO

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:SOURce:MISO <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:SPI:SOURce:MISO command sets the source for the SPI serial MISO data.

Query Syntax

```
:SBUS<n>:SPI:SOURce:MISO?
```

The :SBUS<n>:SPI:SOURce:MISO? query returns the current source for the SPI serial MISO data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:DATA" on page 711
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 712
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 715
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 717
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 716
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 718

:SBUS<n>:SPI:SOURce:MOSI

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:SOURce:MOSI <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:SPI:SOURce:MOSI command sets the source for the SPI serial MOSI data.

You can also use the equivalent :SBUS<n>:SPI:SOURce:DATA command to set the MOSI data source.

Query Syntax

```
:SBUS<n>:SPI:SOURce:MOSI?
```

The :SBUS<n>:SPI:SOURce:MOSI? query returns the current source for the SPI serial MOSI data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:SOURce:DATA" on page 711
- ":SBUS<n>:SPI:SOURce:MISO" on page 713
- ":SBUS<n>:SPI:SOURce:CLOCk" on page 710
- ":SBUS<n>:SPI:SOURce:FRAMe" on page 712
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 715
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 717
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 716
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 718

:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 716
- ":SBUS<n>:SPI:SOURce:MISO" on page 713

:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh <width>

<width> ::= integer from 4 to 64 in NR1 format
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>

<width> ::= integer from 4 to 64 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 715
- ":SBUS<n>:SPI:SOURce:MISO" on page 713

:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA <string>

<string> ::= "nn...n" where n ::= {0 | 1 | X | $}

<string ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X | $}
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA command defines the SPI data pattern resource according to the string parameter. This pattern, along with the data width, control the data pattern searched for in the data stream.

If the string parameter starts with "0x", it is a hexadecimal string made up of hexadecimal and X (don't care) characters; otherwise, it is a binary string made up of 0, 1, and X (don't care) characters.

NOTE The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA? query returns the current settings of the specified SPI data pattern resource in the binary string format.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 718
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:DATA" on page 711

:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh <width>

<width> ::= integer from 4 to 64 in NR1 format
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 64 bits.

NOTE The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh should be set before :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh?
```

The :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>

<width> ::= integer from 4 to 64 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 717
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:DATA" on page 711

:SBUS<n>:SPI:TRIGger:TYPE

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:TRIGger:TYPE <value>

<value> ::= {MOSI | MISO}
```

The :SBUS<n>:SPI:TRIGger:TYPE command specifies whether the SPI trigger will be on the MOSI data or the MISO data.

When triggering on MOSI data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh commands.

When triggering on MISO data, the data value is specified by the :SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA and :SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh commands.

Query Syntax

```
:SBUS<n>:SPI:TRIGger:TYPE?
```

The :SBUS<n>:SPI:TRIGger:TYPE? query returns the current SPI trigger type setting.

Return Format

```
<value><NL>

<value> ::= {MOSI | MISO}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:SPI:SOURce:DATA" on page 711
- ":SBUS<n>:SPI:SOURce:MOSI" on page 714
- ":SBUS<n>:SPI:SOURce:MISO" on page 713
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 715
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 717
- ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 716
- ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 718
- ":TRIGger:MODE" on page 851

:SBUS<n>:SPI:WIDTh

N (see page 1088)

Command Syntax

```
:SBUS<n>:SPI:WIDTh <word_width>

<word_width> ::= integer 4-16 in NR1 format
```

The :SBUS<n>:SPI:WIDTh command determines the number of bits in a word of data for SPI.

Query Syntax

```
:SBUS<n>:SPI:WIDTh?
```

The :SBUS<n>:SPI:WIDTh? query returns the current SPI decode word width.

Return Format

```
<word_width><NL>

<word_width> ::= integer 4-16 in NR1 format
```

Errors

- "-241, Hardware missing" on page 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:SPI Commands" on page 704

:SBUS<n>:UART Commands

NOTE These commands are only valid when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Table 106 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 724)	:SBUS<n>:UART:BASE? (see page 724)	<base> ::= {ASCii | BINary | HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 725)	:SBUS<n>:UART:BAUDrate? (see page 725)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorder <bitorder> (see page 726)	:SBUS<n>:UART:BITorder? (see page 726)	<bitorder> ::= {LSBFirst | MSBFirst}
n/a	:SBUS<n>:UART:COUNt:ERRor? (see page 727)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNt:RESet (see page 728)	n/a	n/a
n/a	:SBUS<n>:UART:COUNt:RXFRames? (see page 729)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNt:TXFRames? (see page 730)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 731)	:SBUS<n>:UART:FRAMing? (see page 731)	<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 732)	:SBUS<n>:UART:PARity? (see page 732)	<parity> ::= {EVEN | ODD | NONE}
:SBUS<n>:UART:POLarity <polarity> (see page 733)	:SBUS<n>:UART:POLarity? (see page 733)	<polarity> ::= {HIGH | LOW}

Table 106 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce:RX <source> (see page 734)	:SBUS<n>:UART:SOURce:RX? (see page 734)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:SOURce:TX <source> (see page 735)	:SBUS<n>:UART:SOURce:TX? (see page 735)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:SBUS<n>:UART:TRIGger:BASE <base> (see page 736)	:SBUS<n>:UART:TRIGger:BASE? (see page 736)	<base> ::= {ASCii | HEX}
:SBUS<n>:UART:TRIGger:BURSt <value> (see page 737)	:SBUS<n>:UART:TRIGger:BURSt? (see page 737)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger:DATA <value> (see page 738)	:SBUS<n>:UART:TRIGger:DATA? (see page 738)	<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 739)	:SBUS<n>:UART:TRIGger:IDLE? (see page 739)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger:QUALifier <value> (see page 740)	:SBUS<n>:UART:TRIGger:QUALifier? (see page 740)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

Table 106 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 741)	:SBUS<n>:UART:TRIGger :TYPE? (see page 741)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:SBUS<n>:UART:WIDTh <width> (see page 742)	:SBUS<n>:UART:WIDTh? (see page 742)	<width> ::= {5 | 6 | 7 | 8 | 9}

:SBUS<n>:UART:BASE

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:BAUDrate

N (see page 1088)

Command Syntax

```
:SBUS<n>:UART:BAUDrate <baudrate>

<baudrate> ::= integer from 100 to 8000000
```

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 from 100 b/s to 8 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
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:BITorder

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741
- ":SBUS<n>:UART:SOURce:RX" on page 734
- ":SBUS<n>:UART:SOURce:TX" on page 735

:SBUS<n>:UART:COUNt:ERRor

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 728
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:COUNt:RESet

N (see page 1088)

Command Syntax `:SBUS<n>:UART:COUNt:RESet`

Resets the UART frame counters.

Errors

- "-241, Hardware missing" on page 1049

See Also

- ":SBUS<n>:UART:COUNt:ERRor" on page 727
- ":SBUS<n>:UART:COUNt:RXFRames" on page 729
- ":SBUS<n>:UART:COUNt:TXFRames" on page 730
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:COUNt:RXFRames

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 728
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:COUNt:TXFRames

N (see page 1088)

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 1049

See Also

- ":SBUS<n>:UART:COUNt:RESet" on page 728
- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:FRAMing

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:UART Commands" on page 721

:SBUS<n>:UART:PARity

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:POLarity

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:SOURce:RX

N (see page 1088)

Command Syntax

```
:SBUS<n>:UART:SOURce:RX <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:RX?
```

The :SBUS<n>:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741
- ":SBUS<n>:UART:BITorder" on page 726

:SBUS<n>:UART:SOURce:TX

N (see page 1088)

Command Syntax

```
:SBUS<n>:UART:SOURce:TX <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :SBUS<n>:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:SBUS<n>:UART:SOURce:TX?
```

The :SBUS<n>:UART:SOURce:TX? query returns the current source for the UART Tx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741
- ":SBUS<n>:UART:BITorder" on page 726

:SBUS<n>:UART:TRIGger:BASE

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:DATA" on page 738

:SBUS<n>:UART:TRIGger:BURSt

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:IDLE" on page 739
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:TRIGger:DATA

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:BASE" on page 736
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:TRIGger:IDLE

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:BURSt" on page 737
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:TRIGger:QUALifier

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

:SBUS<n>:UART:TRIGger:TYPE

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:DATA" on page 738
- ":SBUS<n>:UART:TRIGger:QUALifier" on page 740
- ":SBUS<n>:UART:WIDTh" on page 742

:SBUS<n>:UART:WIDTh

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:UART:TRIGger:TYPE" on page 741

27 :SEARch Commands

Control the event search modes and parameters for each search type. See:

General :SEARch Commands

Table 107 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNt? (see page 745)	<count> ::= an integer count value
:SEARch:MODE <value> (see page 746)	:SEARch:MODE? (see page 746)	<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2}}
:SEARch:STATe <value> (see page 747)	:SEARch:STATe? (see page 747)	<value> ::= {{0 | OFF} | {1 | ON}}

:SEARch:COUNt

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743

:SEARch:MODE

N (see page 1088)

Command Syntax

```
:SEARch:MODE <value>

<value> ::= {EDGE | GLITch | RUNT | TRANsition | SERial{1 | 2}}
```

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} | OFF}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:STATe" on page 747

:SEARch:STATe

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746

:SEARch:EDGE Commands

Table 108 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 749)	:SEARch:EDGE:SLOPe? (see page 749)	<slope> ::= {POSitive | NEGative | EITHer}
:SEARch:EDGE:SOURce <source> (see page 750)	:SEARch:EDGE:SOURce? (see page 750)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:EDGE:SLOPe

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743

:SEARch:EDGE:SOURce

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743

:SEARch:GLITch Commands

Table 109 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREaterthan <greater_than_time>[suffix] (see page 752)	:SEARch:GLITch:GREaterthan? (see page 752)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:LESSthan <less_than_time>[suffix] (see page 753)	:SEARch:GLITch:LESSthan? (see page 753)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:SEARch:GLITch:POLarity <polarity> (see page 754)	:SEARch:GLITch:POLarity? (see page 754)	<polarity> ::= {POSitive | NEGative}
:SEARch:GLITch:QUALifier <qualifier> (see page 755)	:SEARch:GLITch:QUALifier? (see page 755)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:SEARch:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 756)	:SEARch:GLITch:RANGe? (see page 756)	<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 757)	:SEARch:GLITch:SOURce? (see page 757)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:GLITch:GREaterthan

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:GLITch:SOURce" on page 757
- ":SEARch:GLITch:QUALifier" on page 755
- ":SEARch:MODE" on page 746

:SEARch:GLITch:LESSthan

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:GLITch:SOURce" on page 757
- ":SEARch:GLITch:QUALifier" on page 755
- ":SEARch:MODE" on page 746

:SEARch:GLITch:POLarity

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:GLITch:SOURce" on page 757

:SEARch:GLITch:QUALifier

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:GLITch:SOURce" on page 757
- ":SEARch:MODE" on page 746

:SEARch:GLITch:RANGe

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:GLITch:SOURce" on page 757
- ":SEARch:GLITch:QUALifier" on page 755
- ":SEARch:MODE" on page 746

:SEARch:GLITch:SOURce

N (see page 1088)

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 27, “:SEARch Commands,” starting on page 743
- Chapter 27, “:SEARch Commands,” starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:GLITch:POLarity" on page 754
- ":SEARch:GLITch:QUALifier" on page 755
- ":SEARch:GLITch:RANGe" on page 756

:SEARch:RUNT Commands

Table 110 :SEARch:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARch:RUNT:POLarity <polarity> (see page 759)	:SEARch:RUNT:POLarity? (see page 759)	<polarity> ::= {POSitive | NEGative | EITHer}
:SEARch:RUNT:QUALifier <qualifier> (see page 760)	:SEARch:RUNT:QUALifier? (see page 760)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:SEARch:RUNT:SOURce <source> (see page 761)	:SEARch:RUNT:SOURce? (see page 761)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:RUNT:TIME <time>[suffix] (see page 762)	:SEARch:RUNT:TIME? (see page 762)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:SEARch:RUNT:POLarity

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:RUNT:SOURce" on page 761

:SEARch:RUNT:QUALifier

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746

:SEARch:RUNT:SOURce

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:RUNT:POLarity" on page 759

:SEARch:RUNT:TIME

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:RUNT:QUALifier" on page 760

:SEARch:TRANsition Commands

Table 111 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 764)	:SEARch:TRANsition:QUALifier? (see page 764)	<qualifier> ::= {GREaterthan | LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 765)	:SEARch:TRANsition:SLOPe? (see page 765)	<slope> ::= {NEGative | POSitive}
:SEARch:TRANsition:SOURce <source> (see page 766)	:SEARch:TRANsition:SOURce? (see page 766)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 767)	:SEARch:TRANsition:TIME? (see page 767)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:SEARch:TRANsition:QUALifier

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:TRANsition:TIME" on page 767

:SEARch:TRANsition:SLOPe

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:TRANsition:SOURce" on page 766
- ":SEARch:TRANsition:TIME" on page 767

:SEARch:TRANsition:SOURce

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:MODE" on page 746
- ":SEARch:TRANsition:SLOPe" on page 765

:SEARch:TRANsition:TIME

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:TRANsition:QUALifier" on page 764

:SEARch:SERial:A429 Commands

Table 112 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:LABel <value> (see page 769)	:SEARch:SERial:A429:LABel? (see page 769)	<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 770)	:SEARch:SERial:A429:MODE? (see page 770)	<condition> ::= {LABel | LBITs | PERRor | WERRor | GERRor | WGERrors | ALLerrors}
:SEARch:SERial:A429:PATTern:DATA <string> (see page 771)	:SEARch:SERial:A429:PATTern:DATA? (see page 771)	<string> ::= "nn...n" where n ::= {0 | 1}, length depends on FORMat
:SEARch:SERial:A429:PATTern:SDI <string> (see page 772)	:SEARch:SERial:A429:PATTern:SDI? (see page 772)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits
:SEARch:SERial:A429:PATTern:SSM <string> (see page 773)	:SEARch:SERial:A429:PATTern:SSM? (see page 773)	<string> ::= "nn" where n ::= {0 | 1}, length always 2 bits

:SEARch:SERial:A429:LABel

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SEARch:SERial:A429:MODE" on page 770

:SEARch:SERial:A429:MODE

N (see page 1088)

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 1049

See Also

- "Introduction to :SBUS<n> Commands" on page 595
- ":SBUS<n>:MODE" on page 599
- ":SEARch:SERial:A429:LABel" on page 769
- ":SEARch:SERial:A429:PATTern:DATA" on page 771
- ":SEARch:SERial:A429:PATTern:SDI" on page 772
- ":SEARch:SERial:A429:PATTern:SSM" on page 773
- ":SBUS<n>:A429:SOURce" on page 609

:SEARch:SERial:A429:PATTern:DATA

N (see page 1088)

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 1049

See Also
- "Introduction to :TRIGger Commands" on page 843
- ":SEARch:SERial:A429:MODE" on page 770
- ":SEARch:SERial:A429:PATTern:SDI" on page 772
- ":SEARch:SERial:A429:PATTern:SSM" on page 773

:SEARch:SERial:A429:PATTern:SDI

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:FORMat" on page 607
- ":SEARch:SERial:A429:MODE" on page 770
- ":SEARch:SERial:A429:PATTern:DATA" on page 771
- ":SEARch:SERial:A429:PATTern:SSM" on page 773

:SEARch:SERial:A429:PATTern:SSM

N (see page 1088)

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 1049

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":SBUS<n>:A429:FORMat" on page 607
- ":SEARch:SERial:A429:MODE" on page 770
- ":SEARch:SERial:A429:PATTern:DATA" on page 771
- ":SEARch:SERial:A429:PATTern:SDI" on page 772

:SEARch:SERial:CAN Commands

Table 113 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 775)	:SEARch:SERial:CAN:MODE? (see page 775)	<value> ::= {DATA | IDData | IDEither | IDRemote | ALLerrors | OVERload | ERRor}
:SEARch:SERial:CAN:PATTern:DATA <string> (see page 776)	:SEARch:SERial:CAN:PATTern:DATA? (see page 776)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:DATA:LENGth <length> (see page 777)	:SEARch:SERial:CAN:PATTern:DATA:LENGth? (see page 777)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:PATTern:ID <string> (see page 778)	:SEARch:SERial:CAN:PATTern:ID? (see page 778)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X} for hexadecimal
:SEARch:SERial:CAN:PATTern:ID:MODE <value> (see page 779)	:SEARch:SERial:CAN:PATTern:ID:MODE? (see page 779)	<value> ::= {STANdard | EXTended}

:SEARch:SERial:CAN:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:CAN:MODE <value>

<value> ::= {DATA | IDData | IDEither | IDRemote | ALLerrors
            | OVERload | ERRor}
```

The :SEARch:SERial:CAN:MODE command selects the type of CAN information to find in the Lister display:

- DATA - searches for CAN Data frames matching the specified ID, Data, and the DLC (Data length code).
- IDData - searches for CAN frames matching the specified ID of a Data frame.
- IDEither - searches for the specified ID, regardless if it is a Remote frame or a Data frame.
- IDRemote - searches for CAN frames matching the specified ID of a Remote frame.
- ALLerrors - searches for CAN active error frames and unknown bus conditions.
- OVERload - searches for CAN overload frames.
- ERRor - searches for CAN Error frame.

Query Syntax

```
:SEARch:SERial:CAN:MODE?
```

The :SEARch:SERial:CAN:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {DATA | IDD | IDE | IDR | ALL | OVER | ERR}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:CAN:PATTern:DATA" on page 776
- ":SEARch:SERial:CAN:PATTern:ID" on page 778

:SEARch:SERial:CAN:PATTern:DATA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:CAN:MODE" on page 775
- ":SEARch:SERial:CAN:PATTern:DATA:LENGth" on page 777

:SEARch:SERial:CAN:PATTern:DATA:LENGth

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:CAN:MODE" on page 775
- ":SEARch:SERial:CAN:PATTern:DATA" on page 776

:SEARch:SERial:CAN:PATTern:ID

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:CAN:MODE" on page 775
- ":SEARch:SERial:CAN:PATTern:ID:MODE" on page 779

:SEARch:SERial:CAN:PATTern:ID:MODE

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:CAN:MODE" on page 775
- ":SEARch:SERial:CAN:PATTern:ID" on page 778

:SEARch:SERial:FLEXray Commands

Table 114 :SEARch:SERial:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:FLEXray:CYCLe <cycle> (see page 781)	:SEARch:SERial:FLEXray:CYCLe? (see page 781)	<cycle> ::= {ALL | <cycle #>} <cycle #> ::= integer from 0-63
:SEARch:SERial:FLEXray:DATA <string> (see page 782)	:SEARch:SERial:FLEXray:DATA? (see page 782)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X }
:SEARch:SERial:FLEXray:DATA:LENGth <length> (see page 783)	:SEARch:SERial:FLEXray:DATA:LENGth? (see page 783)	<length> ::= integer from 1 to 12 in NR1 format
:SEARch:SERial:FLEXray:FRAMe <frame id> (see page 784)	:SEARch:SERial:FLEXray:FRAMe? (see page 784)	<frame_id> ::= {ALL | <frame #>} <frame #> ::= integer from 1-2047
:SEARch:SERial:FLEXray:MODE <value> (see page 785)	:SEARch:SERial:FLEXray:MODE? (see page 785)	<value> := {FRAMe | CYCLe | DATA | HERRor | FERRor | AERRor}

:SEARch:SERial:FLEXray:CYCLe

N (see page 1088)

Command Syntax

```
:SEARch:SERial:FLEXray:CYCLe <cycle>

<cycle> ::= {ALL | <cycle #>}

<cycle #> ::= integer from 0-63
```

The :SEARch:SERial:FLEXray:CYCLe command specifies the cycle value to find when searching for FlexRay frames.

A cycle value of -1 is the same as ALL.

Query Syntax

```
:SEARch:SERial:FLEXray:CYCLe?
```

The :SEARch:SERial:FLEXray:CYCLe? query returns the current cycle value setting.

Return Format

```
<cycle><NL>

<cycle> ::= {ALL | <cycle #>}

<cycle #> ::= integer from 0-63
```

See Also

- Chapter 27, “:SEARch Commands,” starting on page 743
- ":SEARch:SERial:FLEXray:MODE" on page 785

:SEARch:SERial:FLEXray:DATA

N (see page 1088)

Command Syntax

```
:SEARch:SERial:FLEXray:DATA <string>

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

The :SEARch:SERial:FLEXray:DATA command specifies the data value to find when searching for FlexRay frames.

The length of the data value is specified by the :SEARch:SERial:FLEXray:DATA:LENGth command.

Query Syntax

```
:SEARch:SERial:FLEXray:DATA?
```

The :SEARch:SERial:FLEXray:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

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

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:FLEXray:MODE" on page 785
- ":SEARch:SERial:FLEXray:DATA:LENGth" on page 783

:SEARch:SERial:FLEXray:DATA:LENGth

N (see page 1088)

Command Syntax

```
:SEARch:SERial:FLEXray:DATA:LENGth <length>

<length> ::= integer from 1 to 12 in NR1 format
```

The :SEARch:SERial:FLEXray:DATA:LENGth command specifies the length of data values when searching for FlexRay frames.

The data value is specified using the :SEARch:SERial:FLEXray:DATA command.

Query Syntax

```
:SEARch:SERial:FLEXray:DATA:LENGth?
```

The :SEARch:SERial:FLEXray:DATA:LENGth? query returns the current data length setting.

Return Format

```
<length><NL>

<length> ::= integer from 1 to 12 in NR1 format
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:FLEXray:MODE" on page 785
- ":SEARch:SERial:FLEXray:DATA" on page 782

:SEARch:SERial:FLEXray:FRAMe

N (see page 1088)

Command Syntax

```
:SEARch:SERial:FLEXray:FRAMe <frame_id>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

The :SEARch:SERial:FLEXray:FRAMe command specifies the frame ID value to find when searching for FlexRay frames.

Query Syntax

```
:SEARch:SERial:FLEXray:FRAMe?
```

The :SEARch:SERial:FLEXray:FRAMe? query returns the current frame ID setting.

Return Format

```
<frame_id><NL>

<frame_id> ::= {ALL | <frame #>}

<frame #> ::= integer from 1-2047
```

See Also

- Chapter 27, “:SEARch Commands,” starting on page 743
- ":SEARch:SERial:FLEXray:MODE" on page 785

:SEARch:SERial:FLEXray:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:FLEXray:MODE <value>

<value> := {FRAMe | CYCLe | DATA | HERRor | FERRor | AERRor}
```

The :SEARch:SERial:FLEXray:MODE command selects the type of FlexRay information to find in the Lister display:

- FRAMe — searches for FlexRay frames with the specified frame ID.
- CYCLe — searches for FlexRay frames with the specified cycle number and frame ID.
- DATA — searches for FlexRay frames with the specified data, cycle number, and frame ID.
- HERRor — searches for header CRC errors.
- FERRor — searches for frame CRC errors.
- AERRor — searches for all errors.

Query Syntax

```
:SEARch:SERial:FLEXray:MODE?
```

The :SEARch:SERial:FLEXray:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> := {FRAM | CYCL | DATA | HERR | FERR | AERR}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:FLEXray:FRAMe" on page 784
- ":SEARch:SERial:FLEXray:CYCLe" on page 781
- ":SEARch:SERial:FLEXray:DATA" on page 782

:SEARch:SERial:I2S Commands

Table 115 :SEARch:SERial:I2S Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:I2S:AUDio <audio_ch> (see page 787)	:SEARch:SERial:I2S:AUDio? (see page 787)	<audio_ch> ::= {RIGHt | LEFT | EITHer}
:SEARch:SERial:I2S:MODE <value> (see page 788)	:SEARch:SERial:I2S:MODE? (see page 788)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange | OUTRange}
:SEARch:SERial:I2S:PATTern:DATA <string> (see page 789)	:SEARch:SERial:I2S:PATTern:DATA? (see page 789)	<string> ::= "n" where n ::= 32-bit integer in signed 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
:SEARch:SERial:I2S:PATTern:FORMat <base> (see page 790)	:SEARch:SERial:I2S:PATTern:FORMat? (see page 790)	<base> ::= {BINary | HEX | DECimal}
:SEARch:SERial:I2S:RANGe <lower>, <upper> (see page 791)	:SEARch:SERial:I2S:RANGe? (see page 791)	<lower> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal

:SEARch:SERial:I2S:AUDio

N (see page 1088)

Command Syntax

```
:SEARch:SERial:I2S:AUDio <audio_ch>

<audio_ch> ::= {RIGHt | LEFT | EITHer}
```

The :SEARch:SERial:I2S:AUDio command specifies the channel on which to search for I2S events: right, left, or either channel.

Query Syntax

```
:SEARch:SERial:I2S:AUDio?
```

The :SEARch:SERial:I2S:AUDio? query returns the current channel setting.

Return Format

```
<audio_ch><NL>

<audio_ch> ::= {RIGH | LEFT | EITH}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:I2S:MODE" on page 788

:SEARch:SERial:I2S:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:I2S:MODE <value>

<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan | INRange
             | OUTRange}
```

The :SEARch:SERial:I2S:MODE command selects the type of I2S information to find in the Lister display:

- EQUal– searches for the specified audio channel's data word when it equals the specified word.
- NOTequal – searches for any word other than the specified word.
- LESSthan – searches for channel data words less than the specified value.
- GREaterthan – searches for channel data words greater than the specified value.
- INRange – searches for channel data words in the range.
- OUTRange – searches for channel data words outside the range.

Data word values are specified using the :SEARch:SERial:I2S:PATTern:DATA command.

Value ranges are specified using the :SEARch:SERial:I2S:RANGe command.

Query Syntax

```
:SEARch:SERial:I2S:MODE?
```

The :SEARch:SERial:I2S:MODE? query returns the currently selected mode.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | LESS | GRE | INR | OUTR}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:I2S:PATTern:DATA" on page 789
- ":SEARch:SERial:I2S:RANGe" on page 791
- ":SEARch:SERial:I2S:AUDio" on page 787

:SEARch:SERial:I2S:PATTern:DATA

N (see page 1088)

Command Syntax

```
:SEARch:SERial:I2S:PATTern:DATA <string>

<string> ::= "n" where n ::= 32-bit integer in signed 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
```

The :SEARch:SERial:I2S:PATTern:DATA command specifies the data word value when searching for I2S events.

The base of the value entered with this command is specified using the :SEARch:SERial:I2S:PATTern:FORMat command.

Query Syntax

```
:SEARch:SERial:I2S:PATTern:DATA?
```

The :SEARch:SERial:I2S:PATTern:DATA? query returns the current data word value setting.

Return Format

```
<string><NL>

<string> ::= "n" where n ::= 32-bit integer in signed 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
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:I2S:MODE" on page 788
- ":SEARch:SERial:I2S:PATTern:FORMat" on page 790

:SEARch:SERial:I2S:PATTern:FORMat

N (see page 1088)

Command Syntax

```
:SEARch:SERial:I2S:PATTern:FORMat <base>

<base> ::= {BINary | HEX | DECimal}
```

The :SEARch:SERial:I2S:PATTern:FORMat command specifies the number base used with the :SEARch:SERial:I2S:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:I2S:PATTern:FORMat?
```

The :SEARch:SERial:I2S:PATTern:FORMat? query returns the current number base setting.

Return Format

```
<base><NL>

<base> ::= {BIN | HEX | DEC}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:I2S:PATTern:DATA" on page 789

:SEARch:SERial:I2S:RANGe

N (see page 1088)

Command Syntax

```
:SEARch:SERial:I2S:RANGe <lower>, <upper>

<lower> ::= 32-bit integer in signed decimal, <nondecimal>, or <string>

<upper> ::= 32-bit integer in signed decimal, <nondecimal>, or <string>

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

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

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

The :SEARch:SERial:I2S:RANGe command specifies the data value range when searching for I2S events in the INRange and OUTRange search modes (set by the :SEARch:SERial:I2S:MODE command).

You can enter the parameters in any order – the smaller value becomes the <lower> and the larger value becomes the <upper>.

Query Syntax

```
:SEARch:SERial:I2S:RANGe?
```

The :SEARch:SERial:I2S:RANGe? query returns the current data value range setting.

Return Format

```
<lower>, <upper><NL>

<lower> ::= 32-bit integer in signed decimal

<upper> ::= 32-bit integer in signed decimal
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:I2S:MODE" on page 788

:SEARch:SERial:IIC Commands

Table 116 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MODE <value> (see page 793)	:SEARch:SERial:IIC:MODE? (see page 793)	<value> ::= { READ7 | WRITE7 | NACKnowledge | ANACk | R7Data2 | W7Data2 | RESTart | READEprom}
:SEARch:SERial:IIC:PATTern:ADDRess <value> (see page 795)	:SEARch:SERial:IIC:PATTern:ADDRess? (see page 795)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA <value> (see page 796)	:SEARch:SERial:IIC:PATTern:DATA? (see page 796)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:PATTern:DATA2 <value> (see page 797)	:SEARch:SERial:IIC:PATTern:DATA2? (see page 797)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:SEARch:SERial:IIC:QUALifier <value> (see page 798)	:SEARch:SERial:IIC:QUALifier? (see page 798)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}

:SEARch:SERial:IIC:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:IIC:MODE <value>

<value> ::= {READ7 | WRITE7 | NACKnowledge | ANACk | R7Data2
             | W7Data2 | RESTart | READEprom}
```

The :SEARch:SERial:IIC:MODE command selects the type of IIC information to find in the Lister display:

- 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.
- NACKnowledge — searches for missing acknowledge events.
- ANACk — searches for address with no acknowledge events.
- 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.
- RESTart — searches for another start condition occurring before a stop condition.
- READEprom — searches for EEPROM data reads.

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 1090).

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> ::= {READ7 | WRITE7 | NACK | ANAC | R7D2 | W7D2 | REST
             | READE}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743

:SEARch:SERial:IIC:PATTern:ADDRess

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:IIC:MODE" on page 793

:SEARch:SERial:IIC:PATTern:DATA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:IIC:MODE" on page 793
- ":SEARch:SERial:IIC:QUALifier" on page 798
- ":SEARch:SERial:IIC:PATTern:DATA2" on page 797

:SEARch:SERial:IIC:PATTern:DATA2

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:IIC:MODE" on page 793
- ":SEARch:SERial:IIC:PATTern:DATA" on page 796

:SEARch:SERial:IIC:QUALifier

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:IIC:MODE" on page 793
- ":SEARch:SERial:IIC:PATTern:DATA" on page 796

:SEARch:SERial:LIN Commands

Table 117 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 800)	:SEARch:SERial:LIN:ID? (see page 800)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:SEARch:SERial:LIN:MODE <value> (see page 801)	:SEARch:SERial:LIN:MODE? (see page 801)	<value> ::= {ID | DATA | ERRor}
:SEARch:SERial:LIN:PATTern:DATA <string> (see page 802)	:SEARch:SERial:LIN:PATTern:DATA? (see page 802)	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 803)	:SEARch:SERial:LIN:PATTern:DATA:LENGth? (see page 803)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PATTern:FORMat <base> (see page 804)	:SEARch:SERial:LIN:PATTern:FORMat? (see page 804)	<base> ::= {HEX | DECimal}

:SEARch:SERial:LIN:ID

N (see page 1088)

Command Syntax

```
:SEARch:SERial:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
            from 0-63 or 0x00-0x3f (with Option AMS)

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

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

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

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 Option AMS)
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:LIN:MODE" on page 801

:SEARch:SERial:LIN:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:LIN:MODE <value>

<value> ::= {ID | DATA | ERRor}
```

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 IDs are specified using the :SEARch:SERial:LIN:ID command.

Data values are specified using the:SEARch:SERial:LIN:PATTern:DATA 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}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:LIN:ID" on page 800
- ":SEARch:SERial:LIN:PATTern:DATA" on page 802

:SEARch:SERial:LIN:PATTern:DATA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:LIN:MODE" on page 801
- ":SEARch:SERial:LIN:PATTern:FORMat" on page 804
- ":SEARch:SERial:LIN:PATTern:DATA:LENGth" on page 803

:SEARch:SERial:LIN:PATTern:DATA:LENGth

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:LIN:PATTern:DATA" on page 802

:SEARch:SERial:LIN:PATTern:FORMat

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:LIN:PATTern:DATA" on page 802

:SEARch:SERial:M1553 Commands

Table 118 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553:MODE <value> (see page 806)	:SEARch:SERial:M1553:MODE? (see page 806)	<value> ::= {DSTArt | CSTArt | RTA | RTA11 | PERRor | SERRor | MERRor}
:SEARch:SERial:M1553:PATTern:DATA <string> (see page 807)	:SEARch:SERial:M1553:PATTern:DATA? (see page 807)	<string> ::= "nn...n" where n ::= {0 | 1}
:SEARch:SERial:M1553:RTA <value> (see page 808)	:SEARch:SERial:M1553:RTA? (see page 808)	<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 1088)

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 Comamnd/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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:M1553:RTA" on page 808
- ":SEARch:SERial:M1553:PATTern:DATA" on page 807

:SEARch:SERial:M1553:PATTern:DATA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:M1553:MODE" on page 806

:SEARch:SERial:M1553:RTA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743

:SEARch:SERial:SPI Commands

Table 119 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MODE <value> (see page 810)	:SEARch:SERial:SPI:MODE? (see page 810)	<value> ::= {MOSI | MISO}
:SEARch:SERial:SPI:PATTern:DATA <string> (see page 811)	:SEARch:SERial:SPI:PATTern:DATA? (see page 811)	<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F | X}
:SEARch:SERial:SPI:PATTern:WIDTh <width> (see page 812)	:SEARch:SERial:SPI:PATTern:WIDTh? (see page 812)	<width> ::= integer from 1 to 10

:SEARch:SERial:SPI:MODE

N (see page 1088)

Command Syntax

```
:SEARch:SERial:SPI:MODE <value>

<value> ::= {MOSI | MISO}
```

The :SEARch:SERial:SPI:MODE command specifies whether the SPI search will be on the MOSI data or the MISO data.

Data values are specified using the :SEARch:SERial:SPI:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:SPI:MODE?
```

The :SEARch:SERial:SPI:MODE? query returns the current SPI search mode setting.

Return Format

```
<value><NL>

<value> ::= {MOSI | MISO}
```

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:SPI:PATTern:DATA" on page 811

:SEARch:SERial:SPI:PATTern:DATA

N (see page 1088)

Command Syntax

```
:SEARch:SERial:SPI:PATTern:DATA <string>

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

The :SEARch:SERial:SPI:PATTern:DATA command specifies the data value when searching for SPI events.

The width of the data value is specified using the :SEARch:SERial:SPI:PATTern:WIDTh command.

Query Syntax

```
:SEARch:SERial:SPI:PATTern:DATA?
```

The :SEARch:SERial:SPI:PATTern:DATA? query returns the current data value setting.

Return Format

```
<string><NL>

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

See Also

- Chapter 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:SPI:PATTern:WIDTh" on page 812

:SEARch:SERial:SPI:PATTern:WIDTh

N (see page 1088)

Command Syntax

```
:SEARch:SERial:SPI:PATTern:WIDTh <width>

<width> ::= integer from 1 to 10
```

The :SEARch:SERial:SPI:PATTern:WIDTh command specifies the width of the data value (in bytes) when searching for SPI events.

The data value is specified using the :SEARch:SERial:SPI:PATTern:DATA command.

Query Syntax

```
:SEARch:SERial:SPI:PATTern:WIDTh?
```

The :SEARch:SERial:SPI:PATTern:WIDTh? query returns the current data width setting.

Return Format

```
<width><NL>

<width> ::= integer from 1 to 10
```

See Also

- Chapter 27, “:SEARch Commands,” starting on page 743
- ":SEARch:SERial:SPI:PATTern:DATA" on page 811

:SEARch:SERial:UART Commands

Table 120 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:DATA <value> (see page 814)	:SEARch:SERial:UART:DATA? (see page 814)	<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 815)	:SEARch:SERial:UART:MODE? (see page 815)	<value> ::= {RDATa | RD1 | RD0 | RDX | TDATa | TD1 | TD0 | TDX | PARityerror | AERRor}
:SEARch:SERial:UART:QUALifier <value> (see page 816)	:SEARch:SERial:UART:QUALifier? (see page 816)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}

:SEARch:SERial:UART:DATA

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:UART:MODE" on page 815
- ":SEARch:SERial:UART:QUALifier" on page 816

:SEARch:SERial:UART:MODE

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:UART:DATA" on page 814
- ":SEARch:SERial:UART:QUALifier" on page 816

:SEARch:SERial:UART:QUALifier

N (see page 1088)

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 27, ":SEARch Commands," starting on page 743
- ":SEARch:SERial:UART:DATA" on page 814

28 :SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 818.

Table 121 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 819)	:SYSTem:DATE? (see page 819)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy | AUGust | SEPtember | OCTober | NOVember | DECember} <day> ::= {1,..31}
:SYSTem:DSP <string> (see page 820)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 821)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1047).
:SYSTem:LOCK <value> (see page 822)	:SYSTem:LOCK? (see page 822)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:MENU <menu> (see page 823)	n/a	<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}
:SYSTem:PRESet (see page 824)	n/a	See :SYSTem:PRESet (see page 824)
:SYSTem:PROTection:LOCK <value> (see page 827)	:SYSTem:PROTection:LOCK? (see page 827)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:SETup <setup_data> (see page 828)	:SYSTem:SETup? (see page 828)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 830)	:SYSTem:TIME? (see page 830)	<time> ::= hours,minutes,seconds in NR1 format

Introduction to :SYSTem Commands

SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem:DATE

N (see page 1088)

Command Syntax

```
:SYSTem:DATE <date>

<date> ::= <year>,<month>,<day>

<year> ::= 4-digit year in NR1 format

<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe
             | JULy | AUGust | SEPtember | OCTober | NOVember | DECember}

<day> ::= {1,..,31}
```

The :SYSTem:DATE command sets the date. Validity checking is performed to ensure that the date is valid.

Query Syntax

```
:SYSTem:DATE?
```

The SYSTem:DATE? query returns the date.

Return Format

```
<year>,<month>,<day><NL>
```

See Also

- "Introduction to :SYSTem Commands" on page 818
- ":SYSTem:TIME" on page 830

:SYSTem:DSP

N (see page 1088)

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 in the center of the display. Use :SYStem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

See Also

- "Introduction to :SYSTem Commands" on page 818

:SYSTem:ERRor

C (see page 1088)

Query Syntax `:SYSTem:ERRor?`

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

Return Format

```
<error number>,<error string><NL>

<error number> ::= an integer error code in NR1 format

<error string> ::= quoted ASCII string containing the error message
```

Error messages are listed in Chapter 35, “Error Messages,” starting on page 1047.

See Also

- "Introduction to :SYSTem Commands" on page 818
- "*ESR (Standard Event Status Register)" on page 160
- "*CLS (Clear Status)" on page 157

:SYSTem:LOCK

N (see page 1088)

Command Syntax

```
:SYSTem:LOCK <value>

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

The :SYSTem:LOCK command disables the front panel. LOCK ON is the equivalent of sending a local lockout message over the programming interface.

Query Syntax

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format

```
<value><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 818

:SYSTem:MENU

N (see page 1088)

Command Syntax `:SYSTem:MENU <menu>`

`<menu> ::= {MASK | MEASure | SEGMented | LISTer | POWer}`

The :SYSTem:MENU command changes the front panel softkey menu.

:SYSTem:PRESet

C (see page 1088)

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

Digital Channel Menu (MSO models only)	
Channel 0 - 7	Off
Labels	Off
Threshold	TTL (1.4V)

Display Menu	
Persistence	Off
Grid	33%

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	60 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 155
- "*RST (Reset)" on page 168

:SYSTem:PROTection:LOCK

N (see page 1088)

Command Syntax

```
:SYSTem:PROTection:LOCK <value>

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

The :SYSTem:PROTection:LOCK command disables the fifty ohm impedance setting for all analog channels.

Query Syntax

```
:SYSTem:PROTection:LOCK?
```

The :SYSTem:PROTection:LOCK? query returns the analog channel protection lock status.

Return Format

```
<value><NL>

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

See Also

- "Introduction to :SYSTem Commands" on page 818

:SYSTem:SETup

C (see page 1088)

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 818
- "*LRN (Learn Device Setup)" on page 163

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 39, “Programming Examples,” starting on page 1097

:SYSTem:TIME

N (see page 1088)

Command Syntax

```
:SYSTem:TIME <time>

<time> ::= hours,minutes,seconds in NR1 format
```

The :SYSTem:TIME command sets the system time, using a 24-hour format. Commas are used as separators. Validity checking is performed to ensure that the time is valid.

Query Syntax

```
:SYSTem:TIME? <time>
```

The :SYSTem:TIME? query returns the current system time.

Return Format

```
<time><NL>

<time> ::= hours,minutes,seconds in NR1 format
```

See Also

- "Introduction to :SYSTem Commands" on page 818
- ":SYSTem:DATE" on page 819

29 :TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 832.

Table 122 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 833)	:TIMebase:MODE? (see page 833)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 834)	:TIMebase:POSition? (see page 834)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 835)	:TIMebase:RANGe? (see page 835)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase:REFerence {LEFT | CENTer | RIGHt} (see page 836)	:TIMebase:REFerence? (see page 836)	<return_value> ::= {LEFT | CENTer | RIGHt}
:TIMebase:SCALe <scale_value> (see page 837)	:TIMebase:SCALe? (see page 837)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 838)	:TIMebase:VERNier? (see page 838)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 839)	:TIMebase:WINDow:POSition? (see page 839)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 840)	:TIMebase:WINDow:RANGe? (see page 840)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 841)	:TIMebase:WINDow:SCALe? (see page 841)	<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

C (see page 1088)

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 832
- "*RST (Reset)" on page 168
- ":TIMebase:RANGe" on page 835
- ":TIMebase:POSition" on page 834
- ":TIMebase:REFerence" on page 836

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 39, “Programming Examples,” starting on page 1097

:TIMebase:POSition

C (see page 1088)

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 832
- ":TIMebase:REFerence" on page 836
- ":TIMebase:RANGe" on page 835
- ":TIMebase:SCALe" on page 837
- ":TIMebase:WINDow:POSition" on page 839
- ":TIMebase:DELay" on page 1043

:TIMebase:RANGe

C (see page 1088)

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 832
- ":TIMebase:MODE" on page 833
- ":TIMebase:SCALe" on page 837
- ":TIMebase:WINDow:RANGe" on page 840

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 39, “Programming Examples,” starting on page 1097

:TIMebase:REFerence

C (see page 1088)

Command Syntax

```
:TIMebase:REFerence <reference>

<reference> ::= {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the time reference to one division from the left side of the screen, to the center of the screen, or to one division from the right side of the screen. Time reference is the point on the display where the trigger point is referenced.

Query Syntax

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current display reference for the main window.

Return Format

```
<reference><NL>

<reference> ::= {LEFT | CENT | RIGH}
```

See Also

- "Introduction to :TIMebase Commands" on page 832
- ":TIMebase:MODE" on page 833

Example Code

```
   ' TIME_REFERENCE - Possible values are LEFT, CENTer, or RIGHt.
   '  - LEFT sets the display reference one time division from the left.
   '  - CENTer sets the display reference to the center of the screen.
   '  - RIGHt sets the display reference one time division from the righ
t.
   myScope.WriteString ":TIMebase:REFerence CENTer"   ' Set reference to
 center.
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

:TIMebase:SCALe

N (see page 1088)

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 832
- ":TIMebase:RANGe" on page 835
- ":TIMebase:WINDow:SCALe" on page 841
- ":TIMebase:WINDow:RANGe" on page 840

:TIMebase:VERNier

N (see page 1088)

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 832

:TIMebase:WINDow:POSition

C (see page 1088)

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 832
- ":TIMebase:MODE" on page 833
- ":TIMebase:POSition" on page 834
- ":TIMebase:RANGe" on page 835
- ":TIMebase:SCALe" on page 837
- ":TIMebase:WINDow:RANGe" on page 840
- ":TIMebase:WINDow:SCALe" on page 841

:TIMebase:WINDow:RANGe

C (see page 1088)

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 832
- ":TIMebase:RANGe" on page 835
- ":TIMebase:POSition" on page 834
- ":TIMebase:SCALe" on page 837

:TIMebase:WINDow:SCALe

N (see page 1088)

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 832
- ":TIMebase:RANGe" on page 835
- ":TIMebase:POSition" on page 834
- ":TIMebase:SCALe" on page 837
- ":TIMebase:WINDow:RANGe" on page 840

30 :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 853) 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 851).

- **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 º division of sync amplitude with any analog channel as the trigger source.
- **USB (Universal Serial Bus) triggering**– will trigger on a Start of Packet (SOP), End of Packet (EOP), Reset Complete, Enter Suspend, or Exit Suspend signal on the differential USB data lines. USB Low Speed and Full Speed are supported by this trigger.

Reporting the Setup

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

Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a *RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```

General :TRIGger Commands

Table 123 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 846)	n/a	n/a
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 847)	:TRIGger:HFReject? (see page 847)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 848)	:TRIGger:HOLDoff? (see page 848)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:HIGH <level>, <source> (see page 849)	:TRIGger:LEVel:HIGH? <source> (see page 849)	<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 850)	:TRIGger:LEVel:LOW? <source> (see page 850)	<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 851)	:TRIGger:MODE? (see page 851)	<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT | SHOLd | TRANsition | SBUS{1 | 2} | USB} <return_value> ::= {<mode> | <none>} <none> ::= query returns "NONE" if the :TIMebase:MODE is ROLL or XY
:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 852)	:TRIGger:NREJect? (see page 852)	{0 | 1}
:TRIGger:SWEep <sweep> (see page 853)	:TRIGger:SWEep? (see page 853)	<sweep> ::= {AUTO | NORMal}

:TRIGger:FORCe

N (see page 1088)

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 843

:TRIGger:HFReject

C (see page 1088)

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 843
- ":TRIGger[:EDGE]:REJect" on page 869

:TRIGger:HOLDoff

C (see page 1088)

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>

<holdoff_time> ::= 60 ns to 10 s in NR3 format
```

The :TRIGger:HOLDoff command defines the holdoff time value in seconds. Holdoff keeps a trigger from occurring until after a certain amount of time has passed since the last trigger. This feature is valuable when a waveform crosses the trigger level multiple times during one period of the waveform. Without holdoff, the oscilloscope could trigger on each of the crossings, producing a confusing waveform. With holdoff set correctly, the oscilloscope always triggers on the same crossing. The correct holdoff setting is typically slightly less than one period.

Query Syntax

```
:TRIGger:HOLDoff?
```

The :TRIGger:HOLDoff? query returns the holdoff time value for the current trigger mode.

Return Format

```
<holdoff_time><NL>

<holdoff_time> ::= the holdoff time value in seconds in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 843

:TRIGger:LEVel:HIGH

N (see page 1088)

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 843
- ":TRIGger:LEVel:LOW" on page 850
- ":TRIGger:RUNT Commands" on page 892
- ":TRIGger:TRANsition Commands" on page 903
- ":TRIGger[:EDGE]:SOURce" on page 871

:TRIGger:LEVel:LOW

N (see page 1088)

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 843
- ":TRIGger:LEVel:HIGH" on page 849
- ":TRIGger:RUNT Commands" on page 892
- ":TRIGger:TRANsition Commands" on page 903
- ":TRIGger[:EDGE]:SOURce" on page 871

:TRIGger:MODE

C (see page 1088)

Command Syntax

```
:TRIGger:MODE <mode>

<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT
           | SHOLd | TRANsition | SBUS{1 | 2} | USB}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMebase:MODE is ROLL or XY, the query returns "NONE".

Return Format

```
<mode><NL>

<mode> ::= {EDGE | GLIT | PATT | TV | DEL | EBUR | OR | RUNT | SHOL
           | TRAN | SBUS{1 | 2} | USB}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:SWEep" on page 853
- ":TIMebase:MODE" on page 833

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

:TRIGger:NREJect

C (see page 1088)

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 843

:TRIGger:SWEep

C (see page 1088)

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 843

:TRIGger:DELay Commands

Table 124 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SLOPe <slope> (see page 855)	:TRIGger:DELay:ARM:SLOPe? (see page 855)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:ARM:SOURce <source> (see page 856)	:TRIGger:DELay:ARM:SOURce? (see page 856)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:DELay:TDELay:TIME <time_value> (see page 857)	:TRIGger:DELay:TDELay:TIME? (see page 857)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGger:COUNt <count> (see page 858)	:TRIGger:DELay:TRIGger:COUNt? (see page 858)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGger:SLOPe <slope> (see page 859)	:TRIGger:DELay:TRIGger:SLOPe? (see page 859)	<slope> ::= {NEGative | POSitive}
:TRIGger:DELay:TRIGger:SOURce <source> (see page 860)	:TRIGger:DELay:TRIGger:SOURce? (see page 860)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) 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.

If a digital channel is selected as the source, the :DIGital<n>:THReshold or :POD<n>:THReshold command is used to set the trigger level.

:TRIGger:DELay:ARM:SLOPe

N (see page 1088)

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 triggger.

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 843
- ":TRIGger:DELay:ARM:SOURce" on page 856
- ":TRIGger:DELay:TDELay:TIME" on page 857

:TRIGger:DELay:ARM:SOURce

N (see page 1088)

Command Syntax

```
:TRIGger:DELay:ARM:SOURce <source>

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

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

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

The :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> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:DELay:ARM:SLOPe" on page 855
- ":TRIGger:DELay:TDELay:TIME" on page 857
- ":TRIGger:MODE" on page 851

:TRIGger:DELay:TDELay:TIME

N (see page 1088)

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 843
- ":TRIGger:DELay:TRIGger:SLOPe" on page 859
- ":TRIGger:DELay:TRIGger:COUNt" on page 858

:TRIGger:DELay:TRIGger:COUNt

N (see page 1088)

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 843
- ":TRIGger:DELay:TRIGger:SLOPe" on page 859
- ":TRIGger:DELay:TRIGger:SOURce" on page 860
- ":TRIGger:DELay:TDELay:TIME" on page 857

:TRIGger:DELay:TRIGger:SLOPe

N (see page 1088)

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 triggger.

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 843
- ":TRIGger:DELay:TRIGger:SOURce" on page 860
- ":TRIGger:DELay:TDELay:TIME" on page 857
- ":TRIGger:DELay:TRIGger:COUNt" on page 858

:TRIGger:DELay:TRIGger:SOURce

N (see page 1088)

Command Syntax

```
:TRIGger:DELay:TRIGger:SOURce <source>

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

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

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

The :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> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:DELay:TRIGger:SLOPe" on page 859
- ":TRIGger:DELay:TDELay:TIME" on page 857
- ":TRIGger:DELay:TRIGger:COUNt" on page 858
- ":TRIGger:MODE" on page 851

:TRIGger:EBURst Commands

Table 125 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNt <count> (see page 862)	:TRIGger:EBURst:COUNt? (see page 862)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 863)	:TRIGger:EBURst:IDLE? (see page 863)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 864)	:TRIGger:EBURst:SLOPe? (see page 864)	<slope> ::= {NEGative | POSitive}
:TRIGger:EBURst:SOURce <source> (see page 865)	:TRIGger:EBURst:SOURce? (see page 865)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) 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. If a digital channel is selected as the source, the :DIGital<n>:THReshold or :POD<n>:THReshold command is used to set the Nth Edge Burst trigger level.

:TRIGger:EBURst:COUNt

N (see page 1088)

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 843
- ":TRIGger:EBURst:SLOPe" on page 864
- ":TRIGger:EBURst:IDLE" on page 863

:TRIGger:EBURst:IDLE

N (see page 1088)

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 843
- ":TRIGger:EBURst:SLOPe" on page 864
- ":TRIGger:EBURst:COUNt" on page 862

:TRIGger:EBURst:SLOPe

N (see page 1088)

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 843
- ":TRIGger:EBURst:IDLE" on page 863
- ":TRIGger:EBURst:COUNt" on page 862

:TRIGger:EBURst:SOURce

C (see page 1088)

Command Syntax

```
:TRIGger:EBURst:SOURce <source>

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

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

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

The :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> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851

:TRIGger[:EDGE] Commands

Table 126 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LFReject} (see page 867)	:TRIGger[:EDGE]:COUPling? (see page 867)	{AC | DC | LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 868)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 868)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LFReject | HFReject} (see page 869)	:TRIGger[:EDGE]:REJect? (see page 869)	{OFF | LFReject | HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 870)	:TRIGger[:EDGE]:SLOPe? (see page 870)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 871)	:TRIGger[:EDGE]:SOURce? (see page 871)	<source> ::= {CHANnel<n> | EXTernal | LINE | WGEN} for the DSO models <source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

:TRIGger[:EDGE]:COUPling

C (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger[:EDGE]:REJect" on page 869

:TRIGger[:EDGE]:LEVel

C (see page 1088)

Command Syntax

```
:TRIGger[:EDGE]:LEVel <level>

<level> ::= <level>[,<source>]

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= ±(external range setting) in NR3 format
            for external triggers

<level> ::= ±8 V for digital channels (MSO models)

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d> | EXTernal}
             for the MSO models

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

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

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

```
:TRIGger[:EDGE]:LEVel? [<source>]
```

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger[:EDGE]:SOURce" on page 871
- ":EXTernal:RANGe" on page 304
- ":POD<n>:THReshold" on page 505
- ":DIGital<d>:THReshold" on page 287

:TRIGger[:EDGE]:REJect

C (see page 1088)

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 843
- ":TRIGger:HFReject" on page 847
- ":TRIGger[:EDGE]:COUPling" on page 867

:TRIGger[:EDGE]:SLOPe

C (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:TV:POLarity" on page 911

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 39, “Programming Examples,” starting on page 1097

:TRIGger[:EDGE]:SOURce

C (see page 1088)

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal | LINE | WGEN} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN}
             for the MSO models

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

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

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTernal — triggers on the rear panel EXT TRIG IN signal.
- LINE — triggers at the 50% level of the rising or falling edge of the AC power source signal.
- WGEN — 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.

Query Syntax

```
:TRIGger[:EDGE]:SOURce?
```

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | EXT | LINE | WGEN | NONE} for the DSO models

<source> ::= {CHAN<n> | DIG<d> | EXTernal | LINE | WGEN | NONE}
             for the MSO models
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851

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 39, “Programming Examples,” starting on page 1097

:TRIGger:GLITch Commands

Table 127 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see page 874)	:TRIGger:GLITch:GREaterthan? (see page 874)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see page 875)	:TRIGger:GLITch:LESSthan? (see page 875)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 876)	:TRIGger:GLITch:LEVel? (see page 876)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 877)	:TRIGger:GLITch:POLarity? (see page 877)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 878)	:TRIGger:GLITch:QUALifier? (see page 878)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}

Table 127 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see page 879)	:TRIGger:GLITch:RANGe? (see page 879)	<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 880)	:TRIGger:GLITch:SOURce? (see page 880)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

:TRIGger:GLITch:GREaterthan

N (see page 1088)

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 843
- ":TRIGger:GLITch:SOURce" on page 880
- ":TRIGger:GLITch:QUALifier" on page 878
- ":TRIGger:MODE" on page 851

:TRIGger:GLITch:LESSthan

N (see page 1088)

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 843
- ":TRIGger:GLITch:SOURce" on page 880
- ":TRIGger:GLITch:QUALifier" on page 878
- ":TRIGger:MODE" on page 851

:TRIGger:GLITch:LEVel

N (see page 1088)

Command Syntax

```
:TRIGger:GLITch:LEVel <level_argument>

<level_argument> ::= <level>[, <source>]

<level> ::= .75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= ±(external range setting) in NR3 format
            for external triggers (DSO models)

<level> ::= ±8 V for digital channels (MSO models)

<source> ::= {CHANnel<n> | EXTernal} for DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for MSO models

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

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

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:LEVel?
```

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

Return Format

```
<level_argument><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:GLITch:SOURce" on page 880
- ":EXTernal:RANGe" on page 304

:TRIGger:GLITch:POLarity

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:GLITch:SOURce" on page 880

:TRIGger:GLITch:QUALifier

N (see page 1088)

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 843
- ":TRIGger:GLITch:SOURce" on page 880
- ":TRIGger:MODE" on page 851

:TRIGger:GLITch:RANGe

N (see page 1088)

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 843
- ":TRIGger:GLITch:SOURce" on page 880
- ":TRIGger:GLITch:QUALifier" on page 878
- ":TRIGger:MODE" on page 851

:TRIGger:GLITch:SOURce

N (see page 1088)

Command Syntax

```
:TRIGger:GLITch:SOURce <source>

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

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

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

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:SOURce?
```

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE".

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:GLITch:LEVel" on page 876
- ":TRIGger:GLITch:POLarity" on page 877
- ":TRIGger:GLITch:QUALifier" on page 878
- ":TRIGger:GLITch:RANGe" on page 879

Example Code

- "Example Code" on page 871

:TRIGger:OR Commands

Table 128 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 882)	:TRIGger:OR? (see page 882)	<string> ::= "nn...n" where n ::= {R | F | E | X} R = rising edge, F = falling edge, E = either edge, X = don't care. Each character in the string is for an analog or digital channel as shown on the front panel display.

:TRIGger:OR

N (see page 1088)

Command Syntax

```
:TRIGger:OR <string>

<string> ::= "nn...n" where n ::= {R | F | E | X}

R = rising edge, F = falling edge, E = either edge, X = don't care.
```

The :TRIGger:OR command specifies the edges to include in the OR'ed edge trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 4 through 1.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 2 and 1.
4 analog channels only	Bits 0 through 3 - analog channels 4 through 1.
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 843
- ":TRIGger:MODE" on page 851

:TRIGger:PATTern Commands

Table 129 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 884)	:TRIGger:PATTern? (see page 885)	<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} for DSO models <edge_source> ::= {CHANnel<n> | DIGital<d> | NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive | NEGative}
:TRIGger:PATTern:FORMat <base> (see page 886)	:TRIGger:PATTern:FORMat? (see page 886)	<base> ::= {ASCii | HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 887)	:TRIGger:PATTern:GREaterthan? (see page 887)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:LESSthan <less_than_time>[suffix] (see page 888)	:TRIGger:PATTern:LESSthan? (see page 888)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:PATTern:QUALifier <qualifier> (see page 889)	:TRIGger:PATTern:QUALifier? (see page 889)	<qualifier> ::= {ENTered | GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:PATTern:RANGe <less_than_time>[suffix],<greater_than_time>[suffix] (see page 891)	:TRIGger:PATTern:RANGe? (see page 891)	<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

C (see page 1088)

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} for DSO models

<edge_source> ::= {CHANnel<n> | DIGital<d>
                  | NONE} for MSO models

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

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

<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTern command specifies the channel values to be used in the pattern trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 4 through 1.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 2 and 1.
4 analog channels only	Bits 0 through 3 - analog channels 4 through 1.
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 843
- ":TRIGger:PATTern:FORMat" on page 886
- ":TRIGger:PATTern:QUALifier" on page 889
- ":TRIGger:MODE" on page 851

:TRIGger:PATTern:FORMat

N (see page 1088)

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 843
- ":TRIGger:PATTern" on page 884

:TRIGger:PATTern:GREaterthan

N (see page 1088)

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 843
- ":TRIGger:PATTern" on page 884
- ":TRIGger:PATTern:QUALifier" on page 889
- ":TRIGger:MODE" on page 851

:TRIGger:PATTern:LESSthan

N (see page 1088)

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 843
- ":TRIGger:PATTern" on page 884
- ":TRIGger:PATTern:QUALifier" on page 889
- ":TRIGger:MODE" on page 851

:TRIGger:PATTern:QUALifier

N (see page 1088)

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 843
- ":TRIGger:PATTern:GREaterthan" on page 887

- ":TRIGger:PATTern:LESSthan" on page 888
- ":TRIGger:PATTern:RANGe" on page 891

:TRIGger:PATTern:RANGe

N (see page 1088)

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 843
- ":TRIGger:PATTern" on page 884
- ":TRIGger:PATTern:QUALifier" on page 889
- ":TRIGger:MODE" on page 851

:TRIGger:RUNT Commands

Table 130 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarity <polarity> (see page 893)	:TRIGger:RUNT:POLarity? (see page 893)	<polarity> ::= {POSitive | NEGative | EITHer}
:TRIGger:RUNT:QUALifier <qualifier> (see page 894)	:TRIGger:RUNT:QUALifier? (see page 894)	<qualifier> ::= {GREaterthan | LESSthan | NONE}
:TRIGger:RUNT:SOURce <source> (see page 895)	:TRIGger:RUNT:SOURce? (see page 895)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 896)	:TRIGger:RUNT:TIME? (see page 896)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:RUNT:POLarity

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:LEVel:HIGH" on page 849
- ":TRIGger:LEVel:LOW" on page 850
- ":TRIGger:RUNT:SOURce" on page 895

:TRIGger:RUNT:QUALifier

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:RUNT:TIME" on page 896

:TRIGger:RUNT:SOURce

N (see page 1088)

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 843
- ":TRIGger:RUNT:POLarity" on page 893

:TRIGger:RUNT:TIME

N (see page 1088)

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 qualifer.

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 843
- ":TRIGger:RUNT:QUALifier" on page 894

:TRIGger:SHOLd Commands

Table 131 :TRIGger:SHOLd Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLd:SLOPe <slope> (see page 898)	:TRIGger:SHOLd:SLOPe? (see page 898)	<slope> ::= {NEGative | POSitive}
:TRIGger:SHOLd:SOURce :CLOCk <source> (see page 899)	:TRIGger:SHOLd:SOURce :CLOCk? (see page 899)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:SOURce :DATA <source> (see page 900)	:TRIGger:SHOLd:SOURce :DATA? (see page 900)	<source> ::= {CHANnel<n> | DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:SHOLd:TIME:HOLD <time>[suffix] (see page 901)	:TRIGger:SHOLd:TIME:HOLD? (see page 901)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:SHOLd:TIME:SETup <time>[suffix] (see page 902)	:TRIGger:SHOLd:TIME:SETup? (see page 902)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:SHOLd:SLOPe

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:SHOLd:SOURce:CLOCk" on page 899
- ":TRIGger:SHOLd:SOURce:DATA" on page 900

:TRIGger:SHOLd:SOURce:CLOCk

N (see page 1088)

Command Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk <source>

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

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

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

The :TRIGger:SHOLd:SOURce:CLOCk command selects the input channel probing the clock signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:CLOCk?
```

The :TRIGger:SHOLd:SOURce:CLOCk? query returns the currently set clock signal source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:SHOLd:SLOPe" on page 898

:TRIGger:SHOLd:SOURce:DATA

N (see page 1088)

Command Syntax

```
:TRIGger:SHOLd:SOURce:DATA <source>

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

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

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

The :TRIGger:SHOLd:SOURce:DATA command selects the input channel probing the data signal.

Query Syntax

```
:TRIGger:SHOLd:SOURce:DATA?
```

The :TRIGger:SHOLd:SOURce:DATA? query returns the currently set data signal source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:SHOLd:SLOPe" on page 898

:TRIGger:SHOLd:TIME:HOLD

N (see page 1088)

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 843

:TRIGger:SHOLd:TIME:SETup

N (see page 1088)

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 843

:TRIGger:TRANsition Commands

The :TRIGger:TRANsition comamnds set the rise/fall time trigger options.

Table 132 :TRIGger:TRANsition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANsition:QUALifier <qualifier> (see page 904)	:TRIGger:TRANsition:QUALifier? (see page 904)	<qualifier> ::= {GREaterthan | LESSthan}
:TRIGger:TRANsition:SLOPe <slope> (see page 905)	:TRIGger:TRANsition:SLOPe? (see page 905)	<slope> ::= {NEGative | POSitive}
:TRIGger:TRANsition:SOURce <source> (see page 906)	:TRIGger:TRANsition:SOURce? (see page 906)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANsition:TIME <time>[suffix] (see page 907)	:TRIGger:TRANsition:TIME? (see page 907)	<time> ::= floating-point number in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:TRANsition:QUALifier

N (see page 1088)

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 843
- ":TRIGger:TRANsition:TIME" on page 907
- ":TRIGger:MODE" on page 851

:TRIGger:TRANsition:SLOPe

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:TRANsition:SOURce" on page 906

:TRIGger:TRANsition:SOURce

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:TRANsition:SLOPe" on page 905

:TRIGger:TRANsition:TIME

N (see page 1088)

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 843
- ":TRIGger:TRANsition:QUALifier" on page 904

:TRIGger:TV Commands

Table 133 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 909)	:TRIGger:TV:LINE? (see page 909)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 910)	:TRIGger:TV:MODE? (see page 910)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | LFIeld1 | LFIeld2 | LALTernate}
:TRIGger:TV:POLarity <polarity> (see page 911)	:TRIGger:TV:POLarity? (see page 911)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 912)	:TRIGger:TV:SOURce? (see page 912)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see page 913)	:TRIGger:TV:STANdard? (see page 913)	<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 914)	:TRIGger:TV:UDTV:ENUMber? (see page 914)	<count> ::= edge number in NR1 format
:TRIGger:TV:UDTV:HSYNc {{0 | OFF} | {1 | ON}} (see page 915)	:TRIGger:TV:UDTV:HSYNc? (see page 915)	{0 | 1}
:TRIGger:TV:UDTV:HTIMe <time> (see page 916)	:TRIGger:TV:UDTV:HTIMe? (see page 916)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTHan <min_time> (see page 917)	:TRIGger:TV:UDTV:PGTHan? (see page 917)	<min_time> ::= seconds in NR3 format

:TRIGger:TV:LINE

N (see page 1088)

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 134 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 843
- ":TRIGger:TV:STANdard" on page 913
- ":TRIGger:TV:MODE" on page 910

:TRIGger:TV:MODE

N (see page 1088)

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 843
- ":TRIGger:TV:STANdard" on page 913
- ":TRIGger:MODE" on page 851

:TRIGger:TV:POLarity

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:TV:SOURce" on page 912

:TRIGger:TV:SOURce

N (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:TV:POLarity" on page 911

Example Code

- "Example Code" on page 871

:TRIGger:TV:STANdard

N (see page 1088)

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

N (see page 1088)

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.

This command is available with the DSOX3VID extended Video triggering license.

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 913
- ":TRIGger:TV:UDTV:PGTHan" on page 917
- ":TRIGger:TV:UDTV:HSYNc" on page 915

:TRIGger:TV:UDTV:HSYNc

N (see page 1088)

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.

This command is available with the DSOX3VID extended Video triggering license.

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 913
- ":TRIGger:TV:UDTV:HTIMe" on page 916
- ":TRIGger:TV:UDTV:ENUMber" on page 914
- ":TRIGger:TV:UDTV:PGTHan" on page 917

:TRIGger:TV:UDTV:HTIMe

N (see page 1088)

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.

This command is available with the DSOX3VID extended Video triggering license.

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 913
- ":TRIGger:TV:UDTV:HSYNc" on page 915

:TRIGger:TV:UDTV:PGTHan

N (see page 1088)

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.

This command is available with the DSOX3VID extended Video triggering license.

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 913
- ":TRIGger:TV:UDTV:ENUMber" on page 914
- ":TRIGger:TV:UDTV:HSYNc" on page 915

:TRIGger:USB Commands

Table 135 :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:DMINus <source> (see page 919)	:TRIGger:USB:SOURce:DMINus? (see page 919)	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:USB:SOURce:DPLus <source> (see page 920)	:TRIGger:USB:SOURce:DPLus? (see page 920)	<source> ::= {CHANnel<n> | EXTernal} for the DSO models <source> ::= {CHANnel<n> | DIGital<d>} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:USB:SPEed <value> (see page 921)	:TRIGger:USB:SPEed? (see page 921)	<value> ::= {LOW | FULL}
:TRIGger:USB:TRIGger <value> (see page 922)	:TRIGger:USB:TRIGger? (see page 922)	<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}

:TRIGger:USB:SOURce:DMINus

N (see page 1088)

Command Syntax

```
:TRIGger:USB:SOURce:DMINus <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :TRIGger:USB:SOURce:DMINus command sets the source for the USB D- signal.

Query Syntax

```
:TRIGger:USB:SOURce:DMINus?
```

The :TRIGger:USB:SOURce:DMINus? query returns the current source for the USB D- signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:USB:SOURce:DPLus" on page 920
- ":TRIGger:USB:TRIGger" on page 922

:TRIGger:USB:SOURce:DPLus

N (see page 1088)

Command Syntax

```
:TRIGger:USB:SOURce:DPLus <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital<d>} for the MSO models

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

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

The :TRIGger:USB:SOURce:DPLus command sets the source for the USB D+ signal.

Query Syntax

```
:TRIGger:USB:SOURce:DPLus?
```

The :TRIGger:USB:SOURce:DPLus? query returns the current source for the USB D+ signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:USB:SOURce:DMINus" on page 919
- ":TRIGger:USB:TRIGger" on page 922

:TRIGger:USB:SPEed

N (see page 1088)

Command Syntax

```
:TRIGger:USB:SPEed <value>

<value> ::= {LOW | FULL}
```

The :TRIGger:USB:SPEed command sets the expected USB signal speed to be Low Speed (1.5 Mb/s) or Full Speed (12 Mb/s).

Query Syntax

```
:TRIGger:USB:SPEed?
```

The :TRIGger:USB:SPEed? query returns the current speed value for the USB signal.

Return Format

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

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:USB:SOURce:DMINus" on page 919
- ":TRIGger:USB:SOURce:DPLus" on page 920
- ":TRIGger:USB:TRIGger" on page 922

:TRIGger:USB:TRIGger

N (see page 1088)

Command Syntax

```
:TRIGger:USB:TRIGger <value>

<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}
```

The :TRIGger:USB:TRIGger command sets where the USB trigger will occur:

- SOP – Start of packet.
- EOP – End of packet.
- ENTersuspend – Enter suspend state.
- EXITsuspend – Exit suspend state.
- RESet – Reset complete.

Query Syntax

```
:TRIGger:USB:TRIGger?
```

The :TRIGger:USB:TRIGger? query returns the current USB trigger value.

Return Format

```
<value><NL>

<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}
```

See Also

- "Introduction to :TRIGger Commands" on page 843
- ":TRIGger:MODE" on page 851
- ":TRIGger:USB:SPEed" on page 921

31 :WAVeform Commands

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 925.

Table 136 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 931)	:WAVeform:BYTeorder? (see page 931)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 932)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 933)	<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 935)	:WAVeform:FORMat? (see page 935)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <# points> (see page 936)	:WAVeform:POINts? (see page 936)	<# 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}

Table 136 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINts:MODE <points_mode> (see page 938)	:WAVeform:POINts:MODE? (see page 938)	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble? (see page 940)	<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 943)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGMented:TTAG? (see page 944)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVeform:SOURce <source> (see page 945)	:WAVeform:SOURce? (see page 945)	<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS} for DSO models <source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion | MATH | SBUS} for MSO models <n> ::= 1 to (# analog channels) in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 949)	:WAVeform:SOURce:SUBSource? (see page 949)	<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
n/a	:WAVeform:TYPE? (see page 950)	<return_mode> ::= {NORM | PEAK | AVER | HRES}

Table 136 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 951)	:WAVeform:UNSigned? (see page 951)	{0 | 1}
:WAVeform:VIEW <view> (see page 952)	:WAVeform:VIEW? (see page 952)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 953)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 954)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 955)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 956)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 957)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 958)	<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 933) and :WAVeform:PREamble (see page 940). 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 231): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGitize command (see page 191) or :RUN command (see page 211), 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 933) 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 224).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINts command (see page 936). If :WAVeform:POINts MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes or as many as 8,000,000 for a digital channel on the mixed signal oscilloscope. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVeform:POINts may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVeform:POINts must be an even divisor of 1,000 or be set to MAXimum. :WAVeform:POINts determines the increment between time buckets that will be transferred. If POINts = MAXimum, the data cannot be decimated. For example:

- `:WAVeform:POINts 1000` — returns time buckets 0, 1, 2, 3, 4 ,.., 999.
- `:WAVeform:POINts 500` — returns time buckets 0, 2, 4, 6, 8 ,.., 998.
- `:WAVeform:POINts 250` — returns time buckets 0, 4, 8, 12, 16 ,.., 996.
- `:WAVeform:POINts 100` — returns time buckets 0, 10, 20, 30, 40 ,.., 990.

Analog Channel Data

NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVeform:POINts? query (see page 936). 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 222). 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 936). 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 936). 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 231), the value returned by the :WAVeform:XINCrement query (see page 953) 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 935), 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 231), 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 935). 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 951) 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 935) 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 931) and :WAVeform:UNSigned (see page 951) have no effect when the format is ASCii.

Data Format for Transfer - WORD format

WORD format (see ":WAVeform:FORMat" on page 935) 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 936). 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 931) 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 935) 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 931) has no effect when the data format is BYTE.

Digital Channel Data (MSO models only)

The waveform record for digital channels is similar to that of analog channels. The main difference is that the data points represent either DIGital0,..,7 (POD1), DIGital8,..,15 (POD2), or any grouping of digital channels (BUS1 or BUS2).

For digital channels, :WAVeform:UNSigned (see page 951) must be set to ON.

Digital Channel POD Data Format

Data for digital channels is only available in groups of 8 bits (Pod1 = D0 - D7, Pod2 = D8 - D15). The bytes are organized as:

:WAVeform:SOURce	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
POD1	D7	D6	D5	D4	D3	D2	D1	D0
POD2	D15	D14	D13	D12	D11	D10	D9	D8

If the :WAVeform:FORMat is WORD (see page 935) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see page 931) controls which byte is 0.

If a digital channel is not displayed, its bit value in the pod data byte is not defined.

Digital Channel BUS Data Format

Digital channel BUS definitions can include any or all of the digital channels. Therefore, data is always returned as 16-bit values. :BUS commands (see page 233) are used to select the digital channels for a bus.

Reporting the Setup

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

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS
NONE
```

:WAVeform:BYTeorder

C (see page 1088)

Command Syntax

```
:WAVeform:BYTeorder <value>

<value> ::= {LSBFirst | MSBFirst}
```

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFirst sets the most significant byte to be transmitted first. The parameter LSBFirst sets the least significant byte to be transmitted first. This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected. The default setting is LSBFirst.

Query Syntax

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder query returns the current output sequence.

Return Format

```
<value><NL>

<value> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :WAVeform Commands" on page 925
- ":WAVeform:DATA" on page 933
- ":WAVeform:FORMat" on page 935
- ":WAVeform:PREamble" on page 940

Example Code

- "Example Code" on page 946
- "Example Code" on page 941

:WAVeform:COUNt

C (see page 1088)

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 925
- ":ACQuire:COUNt" on page 222
- ":ACQuire:TYPE" on page 231

:WAVeform:DATA

C (see page 1088)

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 925
- ":WAVeform:UNSigned" on page 951
- ":WAVeform:BYTeorder" on page 931
- ":WAVeform:FORMat" on page 935
- ":WAVeform:POINts" on page 936
- ":WAVeform:PREamble" on page 940
- ":WAVeform:SOURce" on page 945
- ":WAVeform:TYPE" on page 950

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 39, “Programming Examples,” starting on page 1097

:WAVeform:FORMat

C (see page 1088)

Command Syntax

```
:WAVeform:FORMat <value>

<value> ::= {WORD | BYTE | ASCii}
```

The :WAVeform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

  ASCII formatted data is transferred ASCii text.
- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCii is the only waveform format allowed.

When the :WAVeform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

Query Syntax

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

Return Format

```
<value><NL>

<value> ::= {WORD | BYTE | ASC}
```

See Also

- "Introduction to :WAVeform Commands" on page 925
- ":WAVeform:BYTeorder" on page 931
- ":WAVeform:SOURce" on page 945
- ":WAVeform:DATA" on page 933
- ":WAVeform:PREamble" on page 940

Example Code

- "Example Code" on page 946

:WAVeform:POINts

C (see page 1088)

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 or digital sources, the records that can be transferred depend on the waveform points mode. The maximum number of points returned for math (function) waveforms is determined by the NORMal waveform points mode. See the :WAVeform:POINts:MODE command (see page 938) 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 938) 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 925
- ":ACQuire:POINts" on page 224
- ":WAVeform:DATA" on page 933
- ":WAVeform:SOURce" on page 945
- ":WAVeform:VIEW" on page 952
- ":WAVeform:PREamble" on page 940
- ":WAVeform:POINts:MODE" on page 938

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 39, “Programming Examples,” starting on page 1097

:WAVeform:POINts:MODE

N (see page 1088)

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 62,500-point (maximum) representation of the raw acquisition record. The measurement record can be retrieved from any source.

If the <points_mode> is NORMal the measurement record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see page 215) or the :DIGitize command (see page 191) in the root subsystem) in order to return more than the *measurement record*.
- :TIMebase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than the *measurement record*.
- 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 925
- ":WAVeform:DATA" on page 933
- ":ACQuire:POINts" on page 224
- ":WAVeform:VIEW" on page 952
- ":WAVeform:PREamble" on page 940
- ":WAVeform:POINts" on page 936
- ":TIMebase:MODE" on page 833
- ":ACQuire:TYPE" on page 231
- ":ACQuire:COUNt" on page 222

:WAVeform:PREamble

C (see page 1088)

Query Syntax

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble query requests the preamble information for the selected waveform source. The preamble data contains information concerning the vertical and horizontal scaling of the data of the corresponding channel.

Return Format

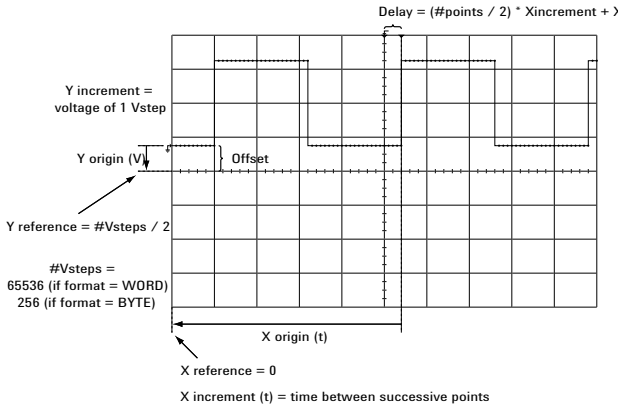
```
<preamble_block><NL>

<preamble_block> ::= <format 16-bit NR1>,
                     <type 16-bit NR1>,
                     <points 32-bit NR1>,
                     <count 32-bit NR1>,
                     <xincrement 64-bit floating point NR3>,
                     <xorigin 64-bit floating point NR3>,
                     <xreference 32-bit NR1>,
                     <yincrement 32-bit floating point NR3>,
                     <yorigin 32-bit floating point NR3>,
                     <yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 4 for ASCii format;
             an integer in NR1 format (format set by :WAVeform:FORMat).

<type> ::= 2 for AVERage type, 0 for NORMal type, 1 for PEAK detect
           type; an integer in NR1 format (type set by :ACQuire:TYPE).

<count> ::= Average count or 1 if PEAK or NORMal; an integer in NR1
            format (count set by :ACQuire:COUNt).
```



See Also

- "Introduction to :WAVeform Commands" on page 925
- ":ACQuire:COUNt" on page 222
- ":ACQuire:POINts" on page 224
- ":ACQuire:TYPE" on page 231

- ":DIGitize" on page 191
- ":WAVeform:COUNt" on page 932
- ":WAVeform:DATA" on page 933
- ":WAVeform:FORMat" on page 935
- ":WAVeform:POINts" on page 936
- ":WAVeform:TYPE" on page 950
- ":WAVeform:XINCrement" on page 953
- ":WAVeform:XORigin" on page 954
- ":WAVeform:XREFerence" on page 955
- ":WAVeform:YINCrement" on page 956
- ":WAVeform:YORigin" on page 957
- ":WAVeform:YREFerence" on page 958

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 = 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)
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

:WAVeform:SEGMented:COUNt

N (see page 1088)

Query Syntax `:WAVeform:SEGMented:COUNt?`

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

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 223
- ":ACQuire:SEGMented:COUNt" on page 226
- ":DIGitize" on page 191
- ":SINGle" on page 213
- ":RUN" on page 211
- "Introduction to :WAVeform Commands" on page 925

Example Code

- "Example Code" on page 227

:WAVeform:SEGMented:TTAG

N (see page 1088)

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 227
- "Introduction to :WAVeform Commands" on page 925

Example Code
- "Example Code" on page 227

:WAVeform:SOURce

C (see page 1088)

Command Syntax

```
:WAVeform:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMory<r> | SBUS{1 | 2}}
             for DSO models

<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCtion
             | MATH | WMEMory<r> | SBUS{1 | 2}}
             for MSO models

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

<r> ::= {1 | 2}
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, reference waveform, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply, integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCii is the only waveform format allowed, and the :WAVeform:DATA? query returns a string with timestamps and associated bus decode information.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCii formats (see ":WAVeform:FORMat" on page 935).

  When the WORD format is chosen, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

  When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with unsigned decimal values separated by commas.

- When BUS1 or BUS2 is selected as the waveform source, you can choose the WORD or ASCii formats (but not BYTE because bus values are always returned as 16-bit values).

  When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with hexadecimal bus values, for example: 0x1938,0xff38,...

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE MATH is an alias for FUNCtion. The :WAVeform:SOURce? Query returns FUNC if the source is FUNCtion or MATH.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | WMEM<r> | SBUS{1 | 2}} for DSO models

<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC
             | WMEM<r> | SBUS{1 | 2}} for MSO models

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

<r> ::= {1 | 2}
```

See Also

- "Introduction to :WAVeform Commands" on page 925
- ":DIGitize" on page 191
- ":WAVeform:FORMat" on page 935
- ":WAVeform:BYTeorder" on page 931
- ":WAVeform:DATA" on page 933
- ":WAVeform:PREamble" on page 940

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 39, “Programming Examples,” starting on page 1097

:WAVeform:SOURce:SUBSource

C (see page 1088)

Command Syntax

```
:WAVeform:SOURce:SUBSource <subsource>

<subsource> ::= {{SUB0 | RX | MOSI} | {SUB1 | TX | MISO}}
```

If the :WAVeform:SOURce is SBUS<n> (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

When using UART serial decode, this option lets you get "TX" data. (TX is an alias for SUB1.) The default, SUB0, specifies "RX" data. (RX is an alias for SUB0.)

When using SPI serial decode, this option lets you get "MISO" data. (MISO is an alias for SUB1.) The default, SUB0, specifies "MOSI" data. (MOSI is an alias for SUB0.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS<n>:MODE is not UART or SPI, the only valid subsource is SUB0.

Query Syntax

```
:WAVeform:SOURce:SUBSource?
```

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

Return Format

```
<subsource><NL>

<subsource> ::= {SUB0 | SUB1}
```

See Also

- "Introduction to :WAVeform Commands" on page 925
- ":WAVeform:SOURce" on page 945

:WAVeform:TYPE

C (see page 1088)

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 925
- ":ACQuire:TYPE" on page 231
- ":WAVeform:DATA" on page 933
- ":WAVeform:PREamble" on page 940
- ":WAVeform:SOURce" on page 945

:WAVeform:UNSigned

C (see page 1088)

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 925
- ":WAVeform:SOURce" on page 945

:WAVeform:VIEW

 (see page 1088)

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 925
- ":WAVeform:POINts" on page 936

:WAVeform:XINCrement

C (see page 1088)

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 925
- ":WAVeform:PREamble" on page 940

Example Code

- "Example Code" on page 941

:WAVeform:XORigin

C (see page 1088)

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 925
- ":WAVeform:PREamble" on page 940
- ":WAVeform:XREFerence" on page 955

Example Code

- "Example Code" on page 941

:WAVeform:XREFerence

C (see page 1088)

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 925
- ":WAVeform:PREamble" on page 940
- ":WAVeform:XORigin" on page 954

Example Code

- "Example Code" on page 941

:WAVeform:YINCrement

C (see page 1088)

Query Syntax

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values. The y-increment for digital waveforms is always "1".

Return Format

```
<value><NL>

<value> ::= y-increment value in the current preamble in 32-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 925
- ":WAVeform:PREamble" on page 940

Example Code

- "Example Code" on page 941

:WAVeform:YORigin

C (see page 1088)

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 925
- ":WAVeform:PREamble" on page 940
- ":WAVeform:YREFerence" on page 958

Example Code

- "Example Code" on page 941

:WAVeform:YREFerence

C (see page 1088)

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 925
- ":WAVeform:PREamble" on page 940
- ":WAVeform:YORigin" on page 957

Example Code

- "Example Code" on page 941

32 :WGEN Commands

When the built-in waveform generator is licensed (Option WGN), 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 commands are used to select the waveform function and parameters. See "Introduction to :WGEN Commands" on page 961.

Table 137 :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:ARBitrary:BYTeorder <order> (see page 962)	:WGEN:ARBitrary:BYTeorder? (see page 962)	<order> ::= {MSBFirst | LSBFirst}
:WGEN:ARBitrary:DATA {<binary> | <value>, <value> ...} (see page 963)	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
n/a	:WGEN:ARBitrary:DATA:ATTRibute:POINts? (see page 964)	<points> ::= number of points in NR1 format
:WGEN:ARBitrary:DATA:CLEar (see page 965)	n/a	n/a
:WGEN:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...} (see page 966)	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
:WGEN:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}} (see page 967)	:WGEN:ARBitrary:INTerpolate? (see page 967)	{0 | 1}

Table 137 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:ARBitrary:STORe <source> (see page 968)	n/a	<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format
:WGEN:FREQuency <frequency> (see page 969)	:WGEN:FREQuency? (see page 969)	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNCtion <signal> (see page 970)	:WGEN:FUNCtion? (see page 972)	<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISe | DC | SINC | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary}
:WGEN:FUNCtion:PULSe:WIDTh <width> (see page 973)	:WGEN:FUNCtion:PULSe:WIDTh? (see page 973)	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNCtion:RAMP:SYMMetry <percent> (see page 974)	:WGEN:FUNCtion:RAMP:SYMMetry? (see page 974)	<percent> ::= symmetry percentage from 0% to 100% in NR3 format
:WGEN:FUNCtion:SQUare:DCYCle <percent> (see page 975)	:WGEN:FUNCtion:SQUare:DCYCle? (see page 975)	<percent> ::= duty cycle percentage from 20% to 80% in NR3 format
:WGEN:MODulation:NOISe <percent> (see page 976)	:WGEN:MODulation:NOISe? (see page 976)	<percent> ::= 0 to 100
:WGEN:OUTPut {{0 | OFF} | {1 | ON}} (see page 977)	:WGEN:OUTPut? (see page 977)	{0 | 1}
:WGEN:OUTPut:LOAD <impedance> (see page 978)	:WGEN:OUTPut:LOAD? (see page 978)	<impedance> ::= {ONEMeg | FIFTy}
:WGEN:PERiod <period> (see page 979)	:WGEN:PERiod? (see page 979)	<period> ::= period in seconds in NR3 format
:WGEN:RST (see page 980)	n/a	n/a
:WGEN:VOLTage <amplitude> (see page 981)	:WGEN:VOLTage? (see page 981)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTage:HIGH <high> (see page 982)	:WGEN:VOLTage:HIGH? (see page 982)	<high> ::= high-level voltage in volts, in NR3 format

Table 137 :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:VOLTage:LOW <low> (see page 983)	:WGEN:VOLTage:LOW? (see page 983)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTage:OFFSet <offset> (see page 984)	:WGEN:VOLTage:OFFSet? (see page 984)	<offset> ::= offset in volts in NR3 format

Introduction to :WGEN Commands

The :WGEN subsystem provides commands to select the waveform generator function and parameters.

Reporting the Setup

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

Return Format

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

```
:WGEN:FUNC SIN;OUTP 0;FREQ +1.0000E+03;VOLT +500.0E-03;VOLT:OFFS
+0.0E+00;:WGEN:OUTP:LOAD ONEM
```

:WGEN:ARBitrary:BYTeorder

N (see page 1088)

Command Syntax

```
:WGEN:ARBitrary:BYTeorder <order>

<order> ::= {MSBFirst | LSBFirst}
```

The :WGEN:ARBitrary:BYTeorder command selects the byte order for binary transfers.

Query Syntax

```
:WGEN:ARBitrary:BYTeorder?
```

The :WGEN:ARBitrary:BYTeorder query returns the current byte order selection.

Return Format

```
<order><NL>

<order> ::= {MSBFirst | LSBFirst}
```

See Also

- ":WGEN:ARBitrary:DATA" on page 963
- ":WGEN:ARBitrary:DATA:DAC" on page 966

:WGEN:ARBitrary:DATA

N (see page 1088)

Command Syntax

```
:WGEN:ARBitrary:DATA {<binary> | <value>, <value> ...}

<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:ARBitrary:DATA command downloads an arbitrary waveform in floating-point values format.

See Also

- ":WGEN:ARBitrary:DATA:DAC" on page 966
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:ARBitrary:DATA:ATTRibute:POINts

N (see page 1088)

Query Syntax `:WGEN:ARBitrary:DATA:ATTRibute:POINts?`

The :WGEN: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:ARBitrary:DATA" on page 963
- ":WGEN:ARBitrary:DATA:DAC" on page 966
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:ARBitrary:DATA:CLEar

N (see page 1088)

Command Syntax `:WGEN:ARBitrary:DATA:CLEar`

The :WGEN:ARBitrary:DATA:CLEar command clears the arbitrary waveform memory and loads it with the default waveform.

See Also

- ":WGEN:ARBitrary:DATA" on page 963
- ":WGEN:ARBitrary:DATA:DAC" on page 966
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:ARBitrary:DATA:DAC

N (see page 1088)

Command Syntax

```
:WGEN:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...}

<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:ARBitrary:DATA:DAC command downloads an arbitrary waveform using 16-bit integer (DAC) values.

See Also

- ":WGEN:ARBitrary:DATA" on page 963
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:ARBitrary:INTerpolate

N (see page 1088)

Command Syntax `:WGEN:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}}`

The :WGEN: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:ARBitrary:INTerpolate?`

The :WGEN:ARBitrary:INTerpolate query returns the current interpolation setting.

Return Format `{0 | 1}`

See Also

- ":WGEN:ARBitrary:DATA" on page 963
- ":WGEN:ARBitrary:DATA:DAC" on page 966
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:ARBitrary:STORe

N (see page 1088)

Command Syntax

```
:WGEN:ARBitrary:STORe <source>

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

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

<r> ::= 1-2 in NR1 format
```

The :WGEN:ARBitrary:STORe command stores the source's waveform into the arbitrary waveform memory.

See Also

- ":WGEN:ARBitrary:DATA" on page 963
- ":WGEN:ARBitrary:DATA:DAC" on page 966
- ":SAVE:ARBitrary[:STARt]" on page 576
- ":RECall:ARBitrary[:STARt]" on page 567

:WGEN:FREQuency

N (see page 1088)

Command Syntax

```
:WGEN:FREQuency <frequency>

<frequency> ::= frequency in Hz in NR3 format
```

For all waveforms except Noise and DC, the :WGEN:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN:PERiod command.

Query Syntax

```
:WGEN:FREQuency?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:PERiod" on page 979

:WGEN:FUNCtion

N (see page 1088)

Command Syntax `:WGEN:FUNCtion <signal>`

```
<signal> ::= {SINusoid | SQUare | RAMP | PULSe | DC | NOISe | SINC
             | EXPRise | EXPFall | CARDiac | GAUSsian | ARBitrary}
```

The :WGEN:FUNCtion command selects the type of waveform:

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset (High-Z)[1]
SINusoid	Use these commands to set the sine signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983	100 mHz to 20 MHz	20 mVpp to 5 Vpp	±2.50 V
SQUare	Use these commands to set the square wave signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983 • ":WGEN:FUNCtion:SQUare:DCYCle" on page 975 The duty cycle can be adjusted from 20% to 80%.	100 mHz to 10 MHz	20 mVpp to 5 Vpp	±2.50 V
RAMP	Use these commands to set the ramp signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983 • ":WGEN:FUNCtion:RAMP:SYMMetry" on page 974 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 5 Vpp	±2.50 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset (High-Z)[1]
PULSe	Use these commands to set the pulse signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983 • ":WGEN:FUNCtion:PULSe:WIDTh" on page 973 The pulse width can be adjusted from 20 ns to the period minus 20 ns.	100 mHz to 10 MHz.	20 mVpp to 5 Vpp	±2.50 V
DC	Use this command to set the DC level: • ":WGEN:VOLTage:OFFSet" on page 984	n/a	n/a	±2.50 V
NOISe	Use these commands to set the noise signal parameters: • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983	n/a	20 mVpp to 5 Vpp	±2.50 V
SINC	Use these commands to set the sine cardinal signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984	100 mHz to 1 MHz	20 mVpp to 5 Vpp	±1.25 V
EXPRise	Use these commands to set the exponential rise signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983	100 mHz to 5 MHz	20 mVpp to 5 Vpp	±2.50 V
EXPFall	Use these commands to set the exponential fall signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983	100 mHz to 5 MHz	20 mVpp to 5 Vpp	±2.50 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude (High-Z)[1]	Offset (High-Z)[1]
CARDiac	Use these commands to set the cardiac signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984	100 mHz to 200 kHz	20 mVpp to 5 Vpp	±1.25 V
GAUSsian	Use these commands to set the gaussian pulse signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984	100 mHz to 5 MHz	20 mVpp to 4 Vpp	±1.25 V
ARBitrary	Use these commands to set the arbitrary signal parameters: • ":WGEN:FREQuency" on page 969 • ":WGEN:PERiod" on page 979 • ":WGEN:VOLTage" on page 981 • ":WGEN:VOLTage:OFFSet" on page 984 • ":WGEN:VOLTage:HIGH" on page 982 • ":WGEN:VOLTage:LOW" on page 983	100 mHz to 12 MHz	20 mVpp to 5 Vpp	±2.50 V
[1]When the output load is 50 Ω, these values are halved.				

Query Syntax `:WGEN:FUNCtion?`

The :WGEN: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 Commands" on page 961
- ":WGEN:MODulation:NOISe" on page 976

:WGEN:FUNCtion:PULSe:WIDTh

N (see page 1088)

Command Syntax

```
:WGEN:FUNCtion:PULSe:WIDTh <width>

<width> ::= pulse width in seconds in NR3 format
```

For Pulse waveforms, the :WGEN:FUNCtion:PULSe:WIDTh command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

Query Syntax

```
:WGEN:FUNCtion:PULSe:WIDTh?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970

:WGEN:FUNCtion:RAMP:SYMMetry

N (see page 1088)

Command Syntax

```
:WGEN:FUNCtion:RAMP:SYMMetry <percent>

<percent> ::= symmetry percentage from 0% to 100% in NR3 format
```

For Ramp waveforms, the :WGEN:FUNCtion:RAMP:SYMMetry command specifies the symmetry of the waveform.

Symmetry represents the amount of time per cycle that the ramp waveform is rising.

Query Syntax

```
:WGEN:FUNCtion:RAMP:SYMMetry?
```

The :WGEN:FUNCtion:RAMP:SYMMetry? query returns the currently set ramp symmetry.

Return Format

```
<percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR3 format
```

See Also

- "Introduction to :WGEN Commands" on page 961
- ":WGEN:FUNCtion" on page 970

:WGEN:FUNCtion:SQUare:DCYCle

N (see page 1088)

Command Syntax

```
:WGEN:FUNCtion:SQUare:DCYCle <percent>

<percent> ::= duty cycle percentage from 20% to 80% in NR3 format
```

For Square waveforms, the :WGEN:FUNCtion:SQUare:DCYCle command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query Syntax

```
:WGEN:FUNCtion:SQUare:DCYCle?
```

The :WGEN: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 NR3 format
```

See Also

- "Introduction to :WGEN Commands" on page 961
- ":WGEN:FUNCtion" on page 970

:WGEN:MODulation:NOISe

N (see page 1088)

Command Syntax

```
:WGEN:MODulation:NOISe <percent>

<percent> ::= 0 to 100
```

The :WGEN:MODulation:NOISe command adds noise to the currently selected signal. The sum of the amplitude between the original signal and injected noise is limited to the regular amplitude limit (for example, 5 Vpp in 1 MOhm), so the range for <percent> varies according to current amplitude.

Note that adding noise affects edge triggering on the waveform generator source as well as the waveform generator sync pulse output signal (which can be sent to TRIG OUT). This is because the trigger comparator is located after the noise source.

Query Syntax

```
:WGEN:MODulation:NOISe?
```

The :WGEN:MODulation:NOISe query returns the percent of added noise.

Return Format

```
<percent><NL>

<percent> ::= 0 to 100
```

See Also

- ":WGEN:FUNCtion" on page 970

:WGEN:OUTPut

N (see page 1088)

Command Syntax

```
:WGEN:OUTPut <on_off>

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

The :WGEN:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query Syntax

```
:WGEN:OUTPut?
```

The :WGEN: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 Commands" on page 961

:WGEN:OUTPut:LOAD

N (see page 1088)

Command Syntax

```
:WGEN:OUTPut:LOAD <impedance>

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

The :WGEN:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax

```
:WGEN:OUTPut:LOAD?
```

The :WGEN:OUTPut:LOAD? query returns the current expected output load impedance.

Return Format

```
<impedance><NL>

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

See Also

- "Introduction to :WGEN Commands" on page 961

:WGEN:PERiod

N (see page 1088)

Command Syntax

```
:WGEN:PERiod <period>

<period> ::= period in seconds in NR3 format
```

For all waveforms except Noise and DC, the :WGEN:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN:FREQuency command.

Query Syntax

```
:WGEN:PERiod?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:FREQuency" on page 969

:WGEN:RST

N (see page 1088)

Command Syntax `:WGEN:RST`

The :WGEN:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- "Introduction to :WGEN Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:FREQuency" on page 969

:WGEN:VOLTage

N (see page 1088)

Command Syntax

```
:WGEN:VOLTage <amplitude>

<amplitude> ::= amplitude in volts in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage command specifies the waveform's amplitude. Use the :WGEN:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, 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.

Query Syntax

```
:WGEN:VOLTage?
```

The :WGEN:VOLTage? query returns the currently specified waveform amplitude.

Return Format

```
<amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format
```

See Also

- "Introduction to :WGEN Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:VOLTage:OFFSet" on page 984
- ":WGEN:VOLTage:HIGH" on page 982
- ":WGEN:VOLTage:LOW" on page 983

:WGEN:VOLTage:HIGH

N (see page 1088)

Command Syntax

```
:WGEN:VOLTage:HIGH <high>

<high> ::= high-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, 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.

Query Syntax

```
:WGEN:VOLTage:HIGH?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:VOLTage:LOW" on page 983
- ":WGEN:VOLTage" on page 981
- ":WGEN:VOLTage:OFFSet" on page 984

:WGEN:VOLTage:LOW

N (see page 1088)

Command Syntax

```
:WGEN:VOLTage:LOW <low>

<low> ::= low-level voltage in volts, in NR3 format
```

For all waveforms except DC, the :WGEN:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, 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.

Query Syntax

```
:WGEN:VOLTage:LOW?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:VOLTage:LOW" on page 983
- ":WGEN:VOLTage" on page 981
- ":WGEN:VOLTage:OFFSet" on page 984

:WGEN:VOLTage:OFFSet

N (see page 1088)

Command Syntax

```
:WGEN:VOLTage:OFFSet <offset>

<offset> ::= offset in volts in NR3 format
```

The :WGEN:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, 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.

Query Syntax

```
:WGEN:VOLTage:OFFSet?
```

The :WGEN: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 Commands" on page 961
- ":WGEN:FUNCtion" on page 970
- ":WGEN:VOLTage" on page 981
- ":WGEN:VOLTage:HIGH" on page 982
- ":WGEN:VOLTage:LOW" on page 983

33 :WMEMory<r> Commands

Control reference waveforms.

Table 138 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 987)	n/a	<r> ::= 1-2 in NR1 format
:WMEMory<r>:DISPlay {{0 | OFF} | {1 | ON}} (see page 988)	:WMEMory<r>:DISPlay? (see page 988)	<r> ::= 1-2 in NR1 format {0 | 1}
:WMEMory<r>:LABel <string> (see page 989)	:WMEMory<r>:LABel? (see page 989)	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 990)	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 991)	:WMEMory<r>:SKEW? (see page 991)	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFset <offset>[suffix] (see page 992)	:WMEMory<r>:YOFFset? (see page 992)	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V | mV}

Table 138 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YRANge <range>[suffix] (see page 993)	:WMEMory<r>:YRANge? (see page 993)	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V | mV}
:WMEMory<r>:YSCale <scale>[suffix] (see page 994)	:WMEMory<r>:YSCale? (see page 994)	<r> ::= 1-2 in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V | mV}

:WMEMory<r>:CLEar

N (see page 1088)

Command Syntax

```
:WMEMory<r>:CLEar

<r> ::= 1-2 in NR1 format
```

The :WMEMory<r>:CLEar command clears the specified reference waveform location.

See Also

- Chapter 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:SAVE" on page 990
- ":WMEMory<r>:DISPlay" on page 988

:WMEMory<r>:DISPlay

N (see page 1088)

Command Syntax

```
:WMEMory<r>:DISPlay <on_off>

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:CLEar" on page 987
- ":WMEMory<r>:LABel" on page 989

:WMEMory<r>:LABel

N (see page 1088)

Command Syntax

```
:WMEMory<r>:LABel <string>

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988

:WMEMory<r>:SAVE

N (see page 1088)

Command Syntax

```
:WMEMory<r>:SAVE <source>

<r> ::= 1-2 in NR1 format

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

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

The :WMEMory<r>:SAVE command copies the analog channel or math function waveform to the specified reference waveform location.

NOTE Only ADD or SUBtract math operations can be saved as reference waveforms.

See Also

- Chapter 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988

:WMEMory<r>:SKEW

N (see page 1088)

Command Syntax

```
:WMEMory<r>:SKEW <skew>

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988
- ":WMEMory<r>:YOFFset" on page 992
- ":WMEMory<r>:YRANge" on page 993
- ":WMEMory<r>:YSCale" on page 994

:WMEMory<r>:YOFFset

N (see page 1088)

Command Syntax

```
:WMEMory<r>:YOFFset <offset> [<suffix>]

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988
- ":WMEMory<r>:YRANge" on page 993
- ":WMEMory<r>:YSCale" on page 994
- ":WMEMory<r>:SKEW" on page 991

:WMEMory<r>:YRANge

N (see page 1088)

Command Syntax

```
:WMEMory<r>:YRANge <range>[<suffix>]

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988
- ":WMEMory<r>:YOFFset" on page 992
- ":WMEMory<r>:SKEW" on page 991
- ":WMEMory<r>:YSCale" on page 994

:WMEMory<r>:YSCale

N (see page 1088)

Command Syntax

```
:WMEMory<r>:YSCale <scale>[<suffix>]

<r> ::= 1-2 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 33, ":WMEMory<r> Commands," starting on page 985
- ":WMEMory<r>:DISPlay" on page 988
- ":WMEMory<r>:YOFFset" on page 992
- ":WMEMory<r>:YRANge" on page 993
- ":WMEMory<r>:SKEW" on page 991

34 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 1088).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 256)	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see page 257)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 260)	
ANALog<n>:LABel	:CHANnel<n>:LABel (see page 261)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 262)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 263)	
ANALog<n>:PMODe	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see page 269)	
:CHANnel:ACTivity (see page 1000)	:ACTivity (see page 183)	
:CHANnel:LABel (see page 1001)	:CHANnel<n>:LABel (see page 261) or :DIGital<n>:LABel (see page 284)	use CHANnel<n>:LABel for analog channels and use DIGital<n>:LABel for digital channels
:CHANnel:THReshold (see page 1002)	:POD<n>:THReshold (see page 505) or :DIGital<d>:THReshold (see page 287)	
:CHANnel2:SKEW (see page 1003)	:CHANnel<n>:PROBe:SKEW (see page 266)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel<n>:INPut (see page 1004)	:CHANnel<n>:IMPedance (see page 259)	
:CHANnel<n>:PMODe (see page 1005)	none	
:DISPlay:CONNect (see page 1006)	:DISPlay:VECTors (see page 300)	
:DISPlay:ORDer (see page 1007)	none	
:ERASe (see page 1008)	:DISplay:CLEar (see page 295)	
:EXTernal:PMODe (see page 1009)	none	
FUNCtion1, FUNCtion2	:FUNCtion Commands (see page 307)	ADD not included
:FUNCtion:SOURce (see page 1010)	:FUNCtion:SOURce1 (see page 336)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNCtion:VIEW (see page 1011)	:FUNCtion:DISPlay (see page 317)	
:HARDcopy:DESTination (see page 1012)	:HARDcopy:FILename (see page 1013)	
:HARDcopy:FILename (see page 1013)	:RECall:FILename (see page 568) :SAVE:FILename (see page 568)	
:HARDcopy:GRAYscale (see page 1014)	:HARDcopy:PALette (see page 355)	
:HARDcopy:IGColors (see page 1015)	:HARDcopy:INKSaver (see page 347)	
:HARDcopy:PDRiver (see page 1016)	:HARDcopy:APRinter (see page 344)	
:MEASure:LOWer (see page 1017)	:MEASure:DEFine:THResholds (see page 396)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see page 1018)	:MEASure:CLEar (see page 394)	
:MEASure:TDELta (see page 1019)	:MARKer:XDELta (see page 370)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:THResholds (see page 1020)	:MEASure:DEFine:THResholds (see page 396)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 1021)	:MEASure:XMAX (see page 444)	
:MEASure:TMIN (see page 1022)	:MEASure:XMIN (see page 445)	
:MEASure:TSTArt (see page 1023)	:MARKer:X1Position (see page 366)	
:MEASure:TSTOp (see page 1024)	:MARKer:X2Position (see page 368)	
:MEASure:TVOLt (see page 1025)	:MEASure:TVALue (see page 431)	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see page 1027)	:MEASure:DEFine:THResholds (see page 396)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 1028)	:MARKer:YDELta (see page 375)	
:MEASure:VSTArt (see page 1029)	:MARKer:Y1Position (see page 373)	
:MEASure:VSTOp (see page 1030)	:MARKer:Y2Position (see page 374)	
:MTESt:AMASk:{SAVE | STORe} (see page 1031)	:SAVE:MASK[:STARt] (see page 584)	
:MTESt:AVERage (see page 1032)	:ACQuire:TYPE AVERage (see page 231)	
:MTESt:AVERage:COUNt (see page 1033)	:ACQuire:COUNt (see page 222)	
:MTESt:LOAD (see page 1034)	:RECall:MASK[:STARt] (see page 569)	
:MTESt:RUMode (see page 1035)	:MTESt:RMODe (see page 486)	
:MTESt:RUMode:SOFailure (see page 1036)	:MTESt:RMODe:FACTion:STOP (see page 490)	
:MTESt:{STARt | STOP} (see page 1037)	:RUN (see page 211) or :STOP (see page 215)	
:MTESt:TRIGger:SOURce (see page 1038)	:TRIGger Commands (see page 843)	There are various commands for setting the source with different types of triggers.

Obsolete Command	Current Command Equivalent	Behavior Differences
:PRINt? (see page 1039)	:DISPlay:DATA? (see page 296)	
:SAVE:IMAGe:AREA (see page 1041)	none	
:TIMebase:DELay (see page 1043)	:TIMebase:POSition (see page 834) or :TIMebase:WINDow:POSition (see page 839)	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 1042)	none	
:TRIGger:THReshold (see page 1044)	:POD<n>:THReshold (see page 505) or :DIGital<d>:THReshold (see page 287)	
:TRIGger:TV:TVMode (see page 1045)	:TRIGger:TV:MODE (see page 910)	

Discontinued Commands

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

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERSistence INFinite (see page 299)	
CHANnel:MATH	:FUNCtion:OPERation (see page 331)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see page 268)	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	

Discontinued Command	Current Command Equivalent	Comments
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCtion:MOVE	none	
FUNCtion:PEAKs	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see page 177)	
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 910)	
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
VAUToscale	none	

Discontinued Parameters

Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 3000 X-Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:ACTivity

(see page 1088)

Command Syntax `:CHANnel:ACTivity`

The :CHANnel:ACTivity command clears the cumulative edge variables for the next activity query.

NOTE The :CHANnel:ACTivity command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :ACTivity command (see page 183) instead.

Query Syntax `:CHANnel:ACTivity?`

The :CHANnel:ACTivity? query returns the active edges since the last clear, and returns the current logic levels.

Return Format

```
<edges>,<levels><NL>

<edges> ::= presence of edges (32-bit integer in NR1 format).

<levels> ::= logical highs or lows (32-bit integer in NR1 format).
```

NOTE A bit equal to zero indicates that no edges were detected at the specified threshold since the last clear on that channel. Edges may have occurred that were not detected because of the threshold setting.

A bit equal to one indicates that edges have been detected at the specified threshold since the last clear on that channel.

:CHANnel:LABel

0 (see page 1088)

Command Syntax

```
:CHANnel:LABel <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2 | DIGital<d>}

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

<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 261) or :DIGital<n>:LABel command (see page 284).

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
```

:CHANnel:THReshold

O (see page 1088)

Command Syntax

```
:CHANnel:THReshold <channel group>, <threshold type> [, <value>]

<channel group> ::= {POD1 | POD2}

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value> ::= voltage for USERdef in NR3 format [volt_type]

[volt_type] ::= {V | mV (-3) | uV (-6)}
```

The :CHANnel:THReshold command sets the threshold for a group of channels. The threshold is either set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is ignored.

NOTE

The :CHANnel:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold command (see page 505) or :DIGital<n>:THReshold command (see page 287).

Query Syntax

```
:CHANnel:THReshold? <channel group>
```

The :CHANnel:THReshold? query returns the voltage and threshold text for a specific group of channels.

Return Format

```
<threshold type> [, <value>]<NL>

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value> ::= voltage for USERdef (float 32 NR3)
```

NOTE

- CMOS = 2.5V
- TTL = 1.5V
- ECL = -1.3V
- USERdef ::= -6.0V to 6.0V

:CHANnel2:SKEW

O (see page 1088)

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 266) 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 254

:CHANnel<n>:INPut

 (see page 1088)

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 259) 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 1088)

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 1088)

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 300) 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 300

:DISPlay:ORDer

 (see page 1088)

Query Syntax `:DISPlay:ORDer?`

The :DISPlay:ORDer? query returns a list of digital channel numbers in screen order, from top to bottom, separated by commas. Busing is displayed as digital channels with no separator. For example, in the following list, the bus consists of digital channels 4 and 5: DIG1, DIG4 DIG5, DIG7.

NOTE The :DISPlay:ORDer command is an obsolete command provided for compatibility to previous oscilloscopes. This command is only available on the MSO models.

Return Format `<order><NL>`

`<order> ::= Unquoted ASCII string`

NOTE A return value is included for each digital channel. A return value of NONE indicates that a channel is turned off.

See Also

- ":DIGital<d>:POSition" on page 285

Example Code

```
' DISP_ORDER - Set the order the channels are displayed on the
' analyzer.  You can enter between 1 and 32 channels at one time.
' If you leave out channels, they will not be displayed.

' Display ONLY channel 0 and channel 10 in that order.
myScope.WriteString ":DISPLAY:ORDER 0,10"
```

See complete example programs at: Chapter 39, “Programming Examples,” starting on page 1097

:ERASe

 (see page 1088)

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 295) instead.

:EXTernal:PMODe

O (see page 1088)

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:SOURce

 (see page 1088)

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 336) 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 Commands" on page 310
- ":FUNCtion:OPERation" on page 331

:FUNCtion:VIEW

 (see page 1088)

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 317) instead.

Query Syntax `:FUNCtion:VIEW?`

The :FUNCtion:VIEW? query returns the current state of the selected function.

Return Format `<view><NL>`

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

:HARDcopy:DESTination

 (see page 1088)

Command Syntax `:HARDcopy:DESTination <destination>`

`<destination> ::= {CENTronics | FLOPpy}`

The :HARDcopy:DESTination command sets the hardcopy destination.

NOTE The :HARDcopy:DESTination command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FILename command (see page 1013) instead.

Query Syntax `:HARDcopy:DESTination?`

The :HARDcopy:DESTination? query returns the selected hardcopy destination.

Return Format `<destination><NL>`

`<destination> ::= {CENT | FLOP}`

See Also

- "Introduction to :HARDcopy Commands" on page 342

:HARDcopy:FILename

O (see page 1088)

Command Syntax

```
:HARDcopy:FILename <string>

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

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

NOTE

The :HARDcopy:FILename command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:FILename command (see page 577) and :RECall:FILename command (see page 568) instead.

Query Syntax

```
:HARDcopy:FILename?
```

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

Return Format

```
<string><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342

:HARDcopy:GRAYscale

 (see page 1088)

Command Syntax

```
:HARDcopy:GRAYscale <gray>

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

The :HARDcopy:GRAYscale command controls whether grayscaling is performed in the hardcopy dump.

NOTE

The :HARDcopy:GRAYscale command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:PALette command (see page 355) instead. (":HARDcopy:GRAYscale ON" is the same as ":HARDcopy:PALette GRAYscale" and ":HARDcopy:GRAYscale OFF" is the same as ":HARDcopy:PALette COLor".)

Query Syntax

```
:HARDcopy:GRAYscale?
```

The :HARDcopy:GRAYscale? query returns a flag indicating whether grayscaling is performed in the hardcopy dump.

Return Format

```
<gray><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342

:HARDcopy:IGColors

 (see page 1088)

Command Syntax

```
:HARDcopy:IGColors <value>

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

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

NOTE

The :HARDcopy:IGColors command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:INKSaver (see page 347) command instead.

Query Syntax

```
:HARDcopy:IGColors?
```

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

Return Format

```
<value><NL>

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

See Also

- "Introduction to :HARDcopy Commands" on page 342

:HARDcopy:PDRiver

O (see page 1088)

Command Syntax

```
:HARDcopy:PDRiver <driver>

<driver> ::= {AP2Xxx | AP21xx | {AP2560 | AP25} | {DJ350 | DJ35} |
              DJ6xx | {DJ630 | DJ63} | DJ6Special | DJ6Photo |
              DJ8Special | DJ8xx | DJ9Vip | OJPRokx50 | DJ9xx | GVIP |
              DJ55xx | {PS470 | PS47} {PS100 | PS10} | CLASer |
              MLASer | LJFastraster | POSTscript}
```

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

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

NOTE The :HARDcopy:PDRiver command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:APRinter (see page 344) command instead.

Query Syntax

```
:HARDcopy:PDRiver?
```

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

Return Format

```
<driver><NL>

<driver> ::= {AP2X | AP21 | AP25 | DJ35 | DJ6 | DJ63 | DJ6S | DJ6P |
              DJ8S | DJ8 | DJ9V | OJPR | DJ9 | GVIP | DJ55 | PS10 |
              PS47 | CLAS | MLAS | LJF | POST}
```

See Also

- "Introduction to :HARDcopy Commands" on page 342

:MEASure:LOWer

(see page 1088)

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 396) 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 388
- ":MEASure:THResholds" on page 1020
- ":MEASure:UPPer" on page 1027

:MEASure:SCRatch

(see page 1088)

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 394) instead.

:MEASure:TDELta

 (see page 1088)

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 370) instead.

Return Format `<value><NL>`

`<value> ::= time difference between start and stop markers in NR3 format`

See Also

- "Introduction to :MARKer Commands" on page 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:XDELta" on page 370
- ":MEASure:TSTArt" on page 1023
- ":MEASure:TSTOp" on page 1024

:MEASure:THResholds

 (see page 1088)

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 396) 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 388
- ":MEASure:LOWer" on page 1017
- ":MEASure:UPPer" on page 1027

:MEASure:TMAX

 (see page 1088)

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 444) 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 388
- ":MEASure:TMIN" on page 1022
- ":MEASure:XMAX" on page 444
- ":MEASure:XMIN" on page 445

:MEASure:TMIN

 (see page 1088)

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 445) 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 388
- ":MEASure:TMAX" on page 1021
- ":MEASure:XMAX" on page 444
- ":MEASure:XMIN" on page 445

:MEASure:TSTArt

O (see page 1088)

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 1090). 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 366) 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 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:XDELta" on page 370
- ":MEASure:TDELta" on page 1019
- ":MEASure:TSTOp" on page 1024

:MEASure:TSTOp

(see page 1088)

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 1090). 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 368) 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 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:X1Position" on page 366
- ":MARKer:X2Position" on page 368
- ":MARKer:XDELta" on page 370
- ":MEASure:TDELta" on page 1019
- ":MEASure:TSTArt" on page 1023

:MEASure:TVOLt

0 (see page 1088)

Query Syntax

```
:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform.  A rising slope is indicated by
            a plus sign (+).  A falling edge is indicated by a minus
            sign (-).

<occurrence> ::= the transition to be reported.  If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

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

<digital channels> ::= {DIGital<d>} for the MSO models

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

<d> ::= 0 to (# digital channels - 1) 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 431).

Return Format

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

```
<value> ::= time in seconds of the specified voltage crossing
            in NR3 format
```

:MEASure:UPPer

(see page 1088)

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 396) 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 388
- ":MEASure:LOWer" on page 1017
- ":MEASure:THResholds" on page 1020

:MEASure:VDELta

O (see page 1088)

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 375) instead.

Return Format `<value><NL>`

`<value> ::= delta V value in NR1 format`

See Also

- "Introduction to :MARKer Commands" on page 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YDELta" on page 375
- ":MEASure:TDELta" on page 1019
- ":MEASure:TSTArt" on page 1023

:MEASure:VSTArt

(see page 1088)

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 1090). 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 373) 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 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YDELta" on page 375
- ":MARKer:X1Y1source" on page 367
- ":MEASure:SOURce" on page 421
- ":MEASure:TDELta" on page 1019
- ":MEASure:TSTArt" on page 1023

:MEASure:VSTOp

 (see page 1088)

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 1090). 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 374) 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 364
- "Introduction to :MEASure Commands" on page 388
- ":MARKer:Y1Position" on page 373
- ":MARKer:Y2Position" on page 374
- ":MARKer:YDELta" on page 375
- ":MARKer:X2Y2source" on page 369
- ":MEASure:SOURce" on page 421
- ":MEASure:TDELta" on page 1019
- ":MEASure:TSTArt" on page 1023

:MTESt:AMASk:{SAVE | STORe}

 (see page 1088)

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 584) instead.

See Also

- "Introduction to :MTESt Commands" on page 469

:MTESt:AVERage

 (see page 1088)

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 231) 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 469
- ":MTESt:AVERage:COUNt" on page 1033

:MTESt:AVERage:COUNt

 (see page 1088)

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 222) 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 469
- ":MTESt:AVERage" on page 1032

:MTESt:LOAD

 (see page 1088)

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 569) instead.

See Also

- "Introduction to :MTESt Commands" on page 469
- ":MTESt:AMASk:{SAVE | STORe}" on page 1031

:MTESt:RUMode

O (see page 1088)

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 486) 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 469
- ":MTESt:RUMode:SOFailure" on page 1036

:MTESt:RUMode:SOFailure

 (see page 1088)

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 490) 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 469
- ":MTESt:RUMode" on page 1035

:MTESt:{STARt | STOP}

 (see page 1088)

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 211) and :STOP command (see page 215) instead.

See Also

- "Introduction to :MTESt Commands" on page 469

:MTESt:TRIGger:SOURce

O (see page 1088)

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 843) 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 469

:PRINt?

 (see page 1088)

Query Syntax

```
:PRINt? [<options>]
<options> ::= [<print option>][,..,<print option>]
<print option> ::= {COLor | GRAYscale | BMP8bit | BMP}
```

The :PRINt? query pulls image data back over the bus for storage.

NOTE The :PRINT command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:DATA command (see page 296) instead.

Print Option	:PRINt command	:PRINt? query	Query Default
COLor	Sets palette=COLor		
GRAYscale	Sets palette=GRAYscale		palette=COLor
PRINter0,1	Causes the USB printer #0,1 to be selected as destination (if connected)	Not used	N/A
BMP8bit	Sets print format to 8-bit BMP	Selects 8-bit BMP formatting for query	N/A
BMP	Sets print format to BMP	Selects BMP formatting for query	N/A
FACTors	Selects outputting of additional settings information for :PRINT	Not used	N/A
NOFactors	Deselects outputting of additional settings information for :PRINT	Not used	N/A

Old Print Option:	Is Now:
HIRes	COLor
LORes	GRAYscale
PARallel	PRINter0

Old Print Option:	Is Now:
DISK	invalid
PCL	invalid

NOTE The PRINt? query is not a core command.

See Also

- "Introduction to Root (:) Commands" on page 182
- "Introduction to :HARDcopy Commands" on page 342
- ":HARDcopy:FACTors" on page 345
- ":HARDcopy:GRAYscale" on page 1014
- ":DISPlay:DATA" on page 296

:SAVE:IMAGe:AREA

 (see page 1088)

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 575
- ":SAVE:IMAGe[:STARt]" on page 578
- ":SAVE:IMAGe:FACTors" on page 579
- ":SAVE:IMAGe:FORMat" on page 580
- ":SAVE:IMAGe:INKSaver" on page 581
- ":SAVE:IMAGe:PALette" on page 582

:SBUS<n>:LIN:SIGNal:DEFinition

 (see page 1088)

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 843
- ":TRIGger:MODE" on page 851
- ":SBUS<n>:LIN:SIGNal:BAUDrate" on page 687
- ":SBUS<n>:LIN:SOURce" on page 688

:TIMebase:DELay

O (see page 1088)

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 836).

NOTE The :TIMebase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase:POSition command (see page 834) 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 39, “Programming Examples,” starting on page 1097

:TRIGger:THReshold

 (see page 1088)

Command Syntax

```
:TRIGger:THReshold <channel group>, <threshold type> [, <value>]

<channel group> ::= {POD1 | POD2}

<threshold type> ::= {CMOS | ECL | TTL | USERdef}

<value>::= voltage for USERdef (floating-point number) [Volt type]

[Volt type] ::= {V | mV | uV}
```

The :TRIGger:THReshold command sets the threshold (trigger level) for a pod of 8 digital channels (either digital channels 0 through 7 or 8 through 15). The threshold can be set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is not required.

NOTE This command is only available on the MSO models.

NOTE The :TRIGger:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold command (see page 505), :DIGital<d>:THReshold command (see page 287), or :TRIGger[:EDGE]:LEVel command (see page 868).

Query Syntax

```
:TRIGger:THReshold? <channel group>
```

The :TRIGger:THReshold? query returns the voltage and threshold text for analog channel 1 or 2, or POD1 or POD2.

Return Format

```
<threshold type>[, <value>]<NL>

<threshold type> ::= {CMOS | ECL | TTL | USER}

CMOS ::= 2.5V

TTL ::= 1.5V

ECL ::= -1.3V

USERdef ::= range from -8.0V to +8.0V.

<value> ::= voltage for USERdef (a floating-point number in NR1.
```

:TRIGger:TV:TVMode

O (see page 1088)

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 913).

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 910) instead.

Query Syntax

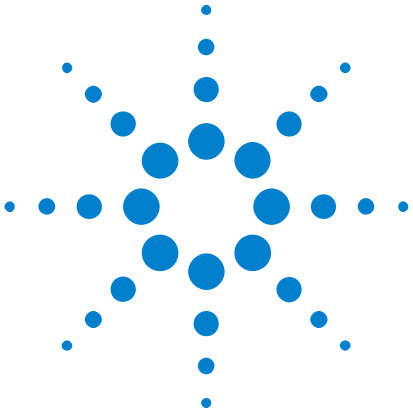
```
:TRIGger:TV:TVMode?
```

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}
```



35 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.

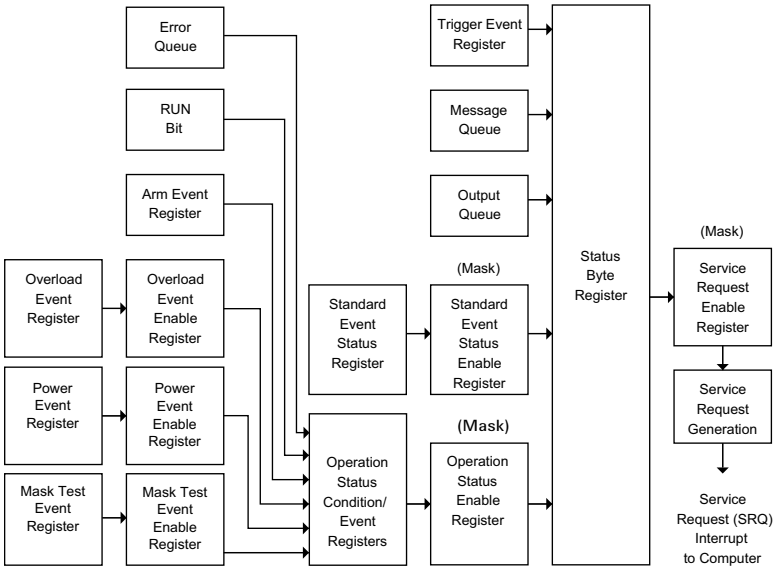
+112, Unknown File Type

+113, Directory Not Supported

36 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.



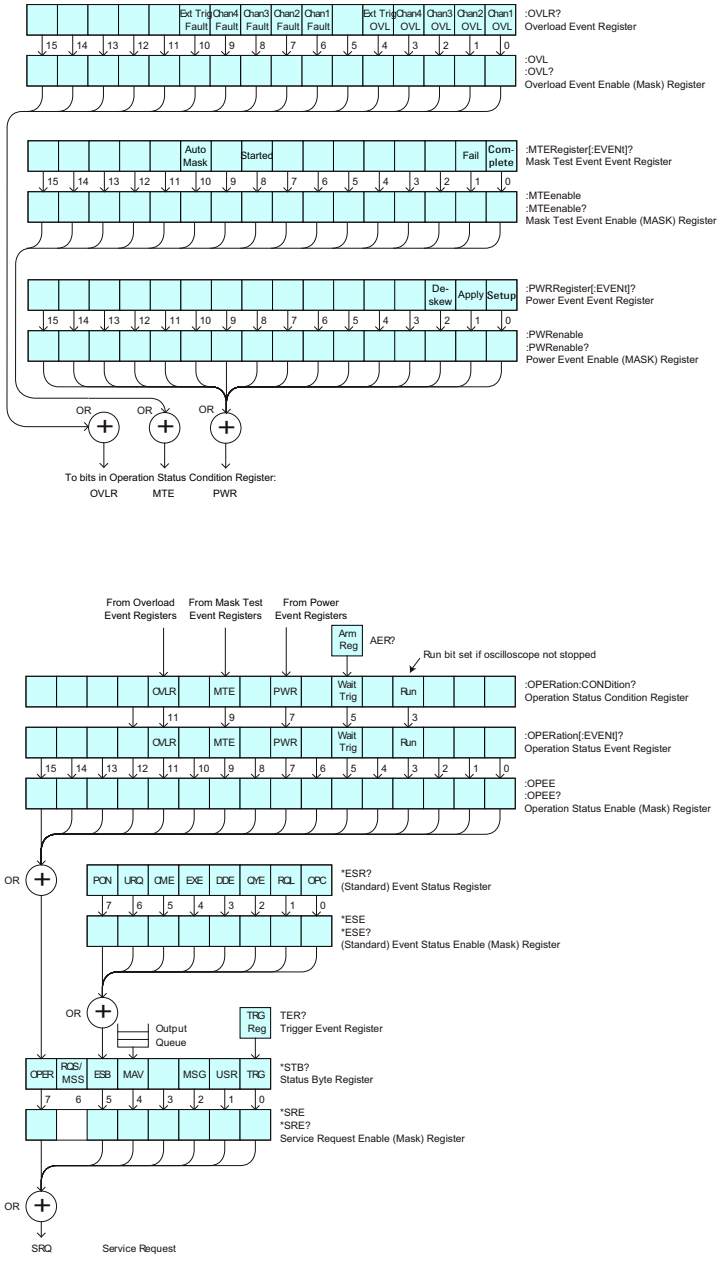
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the 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 67

- Table 65
- Table 72
- Table 73
- Table 75
- Table 70
- 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.

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).

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.

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

When using the Agilent VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

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.

(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.

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.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
PWR bit	bit 7	Comes from the Power Event Registers.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.

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

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.
WAIT TRIG bit	bit 5	Is set by the Trigger Armed Event Register and indicates that the trigger is armed.
PWR bit	bit 7	Comes from the Power Event Registers.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.

The :OPERegister:CONDition? query returns the value of the Operation Status Condition Register.

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.

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.

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.
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.

Power Event Event Register (:PWRRegister[:EVENt])

The Power Event Event Register register hosts these bits:

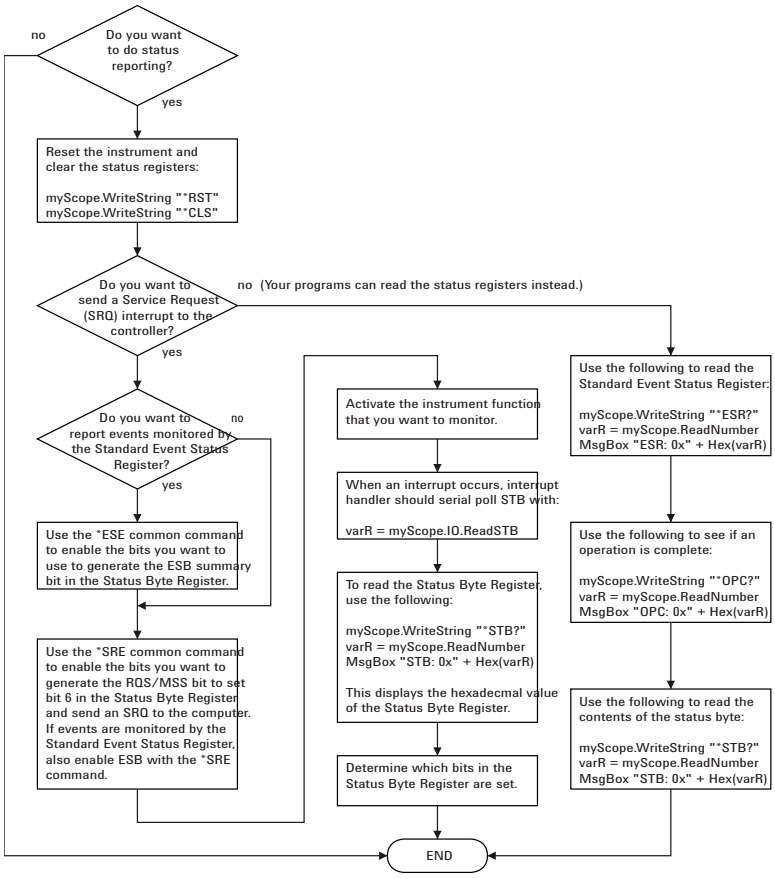
Name	Location	Description
Setup Complete	bit 0	Is set when the power analysis auto setup feature is complete.
Apply Complete	bit 1	Is set when the power analysis apply feature is complete.
Deskew Complete	bit 2	Is set when the power analysis deskew is complete.

The :PWRRegister[:EVENt]? query returns the value of, and clears, the Power Event Event Register.

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately after a program message terminator, the output queue is also cleared.

Status Reporting Decision Chart



37 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 1078).

2 Acquire a waveform (see page 1078).

3 Retrieve results (see page 1078).

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 1079.	See "Polling Synchronization With Timeout" on page 1080.

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 1079 and "Polling Synchronization With Timeout" on page 1080 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 1080 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
```

38
More About Oscilloscope Commands

Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of Agilent InfiniiVision oscilloscopes, commands are classified by the following categories:

- "Core Commands" on page 1088
- "Non-Core Commands" on page 1088
- "Obsolete Commands" on page 1088

C Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent 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.

N Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Agilent 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. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

O Obsolete Commands

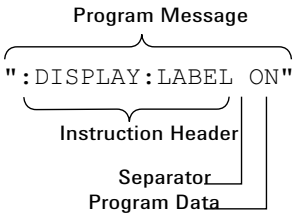
Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

- Chapter 34, “Obsolete and Discontinued Commands,” starting on page 995

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 1090), 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 1091
- "Compound Command Headers" on page 1091
- "Common Command Headers" on page 1091

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 1092 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
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 Agilent VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

```
:<subsystem>:<function><separator><data>;<function><separator><data><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 Agilent VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String
myScope.WriteString ":TIMebase:RANGe?"
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

NOTE **Read Query Results Before Sending Another Command**. Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

Infinity Representation

The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

All Oscilloscope Commands Are Sequential

IEEE 488.2 makes the distinction between sequential and overlapped commands:

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

All of the oscilloscope commands are sequential.

39 Programming Examples

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

VISA COM Examples

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Agilent VISA COM library:
   a. Choose **Tools>References...** from the main menu.
   b. In the References dialog, check the "VISA COM 3.0 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.

```
'
' Agilent VISA COM Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent 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("USB0::0x0957::0x17A6::US50210029::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 "Vertial 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 3.0 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 3.0 Type Library**; then click **OK**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA COM Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
```

```
            myScope = new
              VisaComInstrument("USB0::0x0957::0x17A6::US50210029::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("Vertial 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 addres 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 3.0 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 3.0 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 Agilent IO Libraries Suite 15.

```
'
' Agilent VISA COM Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
```

```
        myScope = New  _
          VisaComInstrument("USB0::0x0957::0x17A6::US50210029::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("Vertial 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 addres 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 for .NET or IronPython

You can also control Agilent oscilloscopes using the VISA COM library and Python programming language on the .NET platform using:

- The Python for .NET package ("http://pythonnet.sourceforge.net/") which integrates with the .NET Common Language Runtime (CLR).
- IronPython ("http://ironpython.codeplex.com/") which is an implementation of the Python programming language running under .NET.

To run this example with Python for .NET or IronPython:

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 the Python for .NET "python.exe" and other files are placed in the current directory, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
.\python.exe example.py
```

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
```

```
#
# Agilent VISA COM Example in Python for .NET or IronPython
# ********************************************************
# This program illustrates a few commonly used programming
# features of your Agilent oscilloscope.
# ********************************************************

# Import Python modules.
# --------------------------------------------------------
import sys
sys.path.append("C:\Python26\Lib")   # Python Standard Library.
sys.path.append("C:\Program Files\IVI Foundation\VISA\VisaCom\
Primary Interop Assemblies")
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("Ivi.Visa.Interop")
from Ivi.Visa.Interop import *


# ========================================================
# 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_UI1(":SYSTem:SETup?")
nLength = len(setup_bytes)
fStream = File.Open("setup.stp", FileMode.Create)
fStream.Write(setup_bytes, 0, nLength)
fStream.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, set up oscilloscope by loading a previously saved setup.
setup_bytes = File.ReadAllBytes("setup.stp")
```

```
 do_command_ieee_block(":SYSTem:SETup", setup_bytes)
 print "Setup bytes restored: %d" % len(setup_bytes)

 # Capture an acquisition using :DIGitize.
 do_command(":DIGitize CHANnel1")


# ============================================================
# Capture:
# ============================================================
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_UI1(":DISPlay:DATA? 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.
 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.
data_bytes = do_query_ieee_block_UI1(":WAVeform:DATA?")
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


# ========================================================
# Send a command and check for errors:
# ========================================================
def do_command(command):
 InfiniiVision.WriteString("%s" % command, True)
 check_instrument_errors(command)


# ========================================================
# Send a command and check for errors:
# ========================================================
def do_command_ieee_block(command, data):
 InfiniiVision.WriteIEEEBlock(command, data, True)
 check_instrument_errors(command)


# ========================================================
# Send a query, check for errors, return string:
# ========================================================
def do_query_string(query):
 InfiniiVision.WriteString("%s" % query, True)
 result = InfiniiVision.ReadString()
 check_instrument_errors(query)
 return result


# ========================================================
# Send a query, check for errors, return string:
# ========================================================
def do_query_ieee_block_UI1(query):
 InfiniiVision.WriteString("%s" % query, True)
 result = \
   InfiniiVision.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, \
   False, True)
 check_instrument_errors(query)
 return result


# ========================================================
# Send a query, check for errors, return values:
# ========================================================
def do_query_number(query):
 InfiniiVision.WriteString("%s" % query, True)
 result = InfiniiVision.ReadNumber(IEEEASCIIType.ASCIIType_R8, True)
 check_instrument_errors(query)
 return result


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

 while True:
```

```
  InfiniiVision.WriteString(":SYSTem:ERRor?", True)
  error_string = InfiniiVision.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 = ResourceManagerClass()
InfiniiVision = FormattedIO488Class()
InfiniiVision.IO = rm.Open("TCPIP0::130.29.70.139::inst0::INSTR", \
  AccessMode.NO_LOCK, 0, "Timeout = 15000;")

# Clear the interface.
IMessage.Clear(InfiniiVision.IO)
print "Interface cleared."

# 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)
```

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, select **VC++ Directories** under Projects and Solutions.

   c Show directories for **Include files**, and add the include directory (for example, Program Files\IVI Foundation\VISA\WinNT\include).

   d Show directories for **Library files**, and add the library files directory (for example, Program Files\IVI Foundation\VISA\WinNT\lib\msc).

   e Click **OK** to close the Options dialog.

9 Build and run the program.

```
/*
 * Agilent VISA Example in C
```

```
 * -------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Agilent oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <visa.h>             /* Agilent VISA routines. */

#define VISA_ADDRESS "USB0::0x0957::0x17A6::US50210029::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.bmp.\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 Agilent IO Libraries Suite and found in the Program Files\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.

```
'
' Agilent VISA Example in Visual Basic
' ----------------------------------------------------------------------
```

```
' This program illustrates a few commonly-used programming
' features of your Agilent 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, _
      "USB0::0x0957::0x17A6::US50210029::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 "Vertial 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 Agilent'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 Agilent IO Libraries Suite and found in the Program Files\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 Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
```

```
using System.Text;

namespace InfiniiVision
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaInstrument("USB0::0x0957::0x17A6::US50210029::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("Vertial 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 addres 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 Agilent'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 Agilent IO Libraries Suite and found in the Program Files\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 Agilent IO Libraries Suite 15.

```
'
' Agilent VISA Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
```

```
            New VisaInstrument("USB0::0x0957::0x17A6::US50210029::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("Vertial 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 addres 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

You can use the Python programming language with the PyVISA package to control Agilent 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 Agilent 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::130.29.70.139::inst0::INSTR")
InfiniiVision.timeout = 15
InfiniiVision.term_chars = ""
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.

```
/*
 * Agilent SICL Example in C
 * ----------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Agilent oscilloscope.
 */
```

```
#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <sicl.h>             /* Agilent SICL routines. */

#define SICL_ADDRESS      "usb0[2391::6054::US50210029::0]"
#define TIMEOUT           5000
#define IEEEBLOCK_SPACE   100000

/* 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 Agilent 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.

```
'
' Agilent SICL Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public id As Integer   ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/iread.
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 "Vertial 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 Agilent'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, Agilent VEE, and Agilent SystemVue.

For more information on Agilent Command Expert, and to download the software, see: "http://www.agilent.com/find/commandexpert"

- "SCPI.NET Example in C#" on page 1198
- "SCPI.NET Example in Visual Basic .NET" on page 1204
- "SCPI.NET Example in IronPython" on page 1210

SCPI.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Install the Agilent 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\Agilent\Command Expert\ScpiNetDrivers
- Windows 7: C:\ProgramData\Agilent\Command Expert\ScpiNetDrivers

d Select the .dll file for your oscilloscope, for example **AgInfiniiVision3000X_01_20.dll**; then, click **OK**.

7 Build and run the program.

For more information, see the SCPI.NET driver help that comes with Agilent Command Expert.

```
/*
 * Agilent SCPI.NET Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Agilent.CommandExpert.ScpiNet.AgInfiniiVision3000X_01_20;

namespace InfiniiVision
{
  class ScpiNetInstrumentApp
  {
    private static AgInfiniiVision3000X myScope;

    static void Main(string[] args)
    {
      try
      {
        string strScopeAddress;
        //strScopeAddress = "a-mx3054a-60028.cos.agilent.com";
        strScopeAddress =
          "TCPIP0::a-mx3054a-60028.cos.agilent.com::inst0::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("Vertial 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 Agilent 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\Agilent\Command Expert\ScpiNetDrivers
      - Windows 7: C:\ProgramData\Agilent\Command Expert\ScpiNetDrivers

   d Select the .dll file for your oscilloscope, for example **AgInfiniiVision3000X_01_20.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 Agilent Command Expert.

```
'
' Agilent SCPI.NET Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Agilent.CommandExpert.ScpiNet.AgInfiniiVision3000X_01_20

Namespace InfiniiVision
  Class ScpiNetInstrumentApp
    Private Shared myScope As AgInfiniiVision3000X

    Public Shared Sub Main(ByVal args As String())
      Try
        Dim strScopeAddress As String
        'strScopeAddress = "a-mx3054a-60028.cos.agilent.com";
        strScopeAddress = _
          "TCPIP0::a-mx3054a-60028.cos.agilent.com::inst0::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("Vertial 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 Agilent 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 Agilent 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
```

```
#
# Agilent SCPI.NET Example in IronPython
# ********************************************************
# This program illustrates a few commonly used programming
# features of your Agilent oscilloscope.
# ********************************************************

# Import Python modules.
# --------------------------------------------------------
import sys
sys.path.append("C:\Python26\Lib")    # Python Standard Library.
sys.path.append("C:\ProgramData\Agilent\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("AgInfiniiVision3000X_01_20")
from Agilent.CommandExpert.ScpiNet.AgInfiniiVision3000X_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 = "a-mx3054a-60028.cos.agilent.com"
addr = "TCPIP0::a-mx3054a-60028.cos.agilent.com::inst0::INSTR"
scope = AgInfiniiVision3000X(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