

Agilent InfiniiVision 4000 X-Series Oscilloscopes

Programmer's Guide



Agilent Technologies

Notices

© Agilent Technologies, Inc. 2005-2013

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Trademarks

Microsoft®, Windows®, Windows NT®, MS Windows®, and Windows Vista® are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Manual Part Number

Version 03.20.0001

Edition

September 4, 2013

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 4000 X-Series oscilloscopes:

Table 1 InfiniiVision 4000 X-Series Oscilloscope Models, Bandwidths, Sample Rates

Bandwidth	200 MHz	350 MHz	500 MHz	1 GHz	1.5 GHz
Sample Rate (interleaved, non-interleaved)	5 GSa/s, 2.5 GSa/s				
2-Channel + 16 Logic Channels MSO	MSO-X 4022A	MSO-X 4032A	MSO-X 4052A		
4-Channel + 16 Logic Channels MSO	MSO-X 4024A	MSO-X 4034A	MSO-X 4054A	MSO-X 4104A	MSO-X 4154A
2-Channel DSO	DSO-X 4022A	DSO-X 4032A	DSO-X 4052A		
4-Channel DSO	DSO-X 4024A	DSO-X 4034A	DSO-X 4054A	DSO-X 4104A	DSO-X 4154A

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

- [Chapter 1](#), “What’s New,” starting on page 33, describes programming command changes in the latest version of oscilloscope software.
- [Chapter 2](#), “Setting Up,” starting on page 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 4000 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 165, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- [Chapter 6](#), “Root (:) Commands,” starting on page 191, 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 231, describes commands for setting the parameters used when acquiring and storing data.
- [Chapter 8](#), “:BUS<n> Commands,” starting on page 247, describes commands that control all oscilloscope functions associated with the digital channels bus display.

- [Chapter 9](#), “`:CALibrate Commands`,” starting on page 257, describes utility commands for determining the state of the calibration factor protection button.
- [Chapter 10](#), “`:CHANnel<n> Commands`,” starting on page 267, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- [Chapter 12](#), “`:DEMO Commands`,” starting on page 299, 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 13](#), “`:DIGital<d> Commands`,” starting on page 307, describes commands that control all oscilloscope functions associated with individual digital channels.
- [Chapter 14](#), “`:DISPlay Commands`,” starting on page 315, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- [Chapter 15](#), “`:DVM Commands`,” starting on page 329, describes commands that control the optional DSOXDVM digital voltmeter analysis feature.
- [Chapter 16](#), “`:EXTernal Trigger Commands`,” starting on page 337, describes commands that control the input characteristics of the external trigger input.
- [Chapter 17](#), “`:FUNCtion<m> Commands`,” starting on page 343, describes commands that control math waveforms.
- [Chapter 18](#), “`:HARDcopy Commands`,” starting on page 375, describes commands that set and query the selection of hardcopy device and formatting options.
- [Chapter 19](#), “`:LISTER Commands`,” starting on page 393, describes commands that turn on/off the Lister display for decoded serial data and get the Lister data.
- [Chapter 20](#), “`:MARKer Commands`,” starting on page 397, describes commands that set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
- [Chapter 21](#), “`:MEASure Commands`,” starting on page 415, describes commands that select automatic measurements (and control markers).
- [Chapter 22](#), “`:MEASure Power Commands`,” starting on page 495, describes measurement commands that are available when the DSOX4PWR power measurements and analysis application is licensed and enabled.
- [Chapter 23](#), “`:MTEST Commands`,” starting on page 517, describes commands that control the mask test features provided with Option LMT.

- [Chapter 24](#), “:POD Commands,” starting on page 551, describes commands that control all oscilloscope functions associated with groups of digital channels.
- [Chapter 25](#), “:POWer Commands,” starting on page 557, describes commands that control the DSOX4PWR power measurement application.
- [Chapter 26](#), “:RECall Commands,” starting on page 623, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- [Chapter 27](#), “:SAVE Commands,” starting on page 633, describes commands that save oscilloscope setups, screen images, and data.
- [Chapter 28](#), “:SBUS<n> Commands,” starting on page 657, describes commands that control oscilloscope functions associated with the serial decode bus and serial triggering.
- [Chapter 29](#), “:SEARch Commands,” starting on page 835, describes commands that control oscilloscope functions associated with searching for waveform events.
- [Chapter 30](#), “:SYSTem Commands,” starting on page 933, describes commands that control basic system functions of the oscilloscope.
- [Chapter 31](#), “:TIMEbase Commands,” starting on page 947, describes commands that control all horizontal sweep functions.
- [Chapter 32](#), “:TRIGger Commands,” starting on page 959, describes commands that control the trigger modes and parameters for each trigger type.
- [Chapter 33](#), “:WAVeform Commands,” starting on page 1043, describes commands that provide access to waveform data.
- [Chapter 34](#), “:WGEN<w> Commands,” starting on page 1079, describes commands that control waveform generator (Option WGN) functions and parameters.
- [Chapter 35](#), “:WMEMory<r> Commands,” starting on page 1123, describes commands that control reference waveforms.
- [Chapter 36](#), “Obsolete and Discontinued Commands,” starting on page 1133, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- [Chapter 37](#), “Error Messages,” starting on page 1191, 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 38](#), “Status Reporting,” starting on page 1199, describes the oscilloscope’s status registers and how to check the status of the instrument.
- [Chapter 39](#), “Synchronizing Acquisitions,” starting on page 1221, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- [Chapter 40](#), “More About Oscilloscope Commands,” starting on page 1231, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- [Chapter 41](#), “Programming Examples,” starting on page 1241.

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/4000X-Series-manual"](http://www.agilent.com/find/4000X-Series-manual)

Contents

In This Book 3

1 What's New

What's New in Version 3.20	34
What's New in Version 3.10	36
What's New in Version 3.01	37
Version 3.00 at Introduction	38
Command Differences From 3000 X-Series Oscilloscopes	39

2 Setting Up

Step 1. Install Agilent IO Libraries Suite software	46
Step 2. Connect and set up the oscilloscope	47
Using the USB (Device) Interface	47
Using the LAN Interface	47
Step 3. Verify the oscilloscope connection	49

3 Getting Started

Basic Oscilloscope Program Structure	56
Initializing	56
Capturing Data	56
Analyzing Captured Data	57

Programming the Oscilloscope	58
Referencing the IO Library	58
Opening the Oscilloscope Connection via the IO Library	59
Initializing the Interface and the Oscilloscope	59
Using :AUToscale to Automate Oscilloscope Setup	60
Using Other Oscilloscope Setup Commands	60
Capturing Data with the :DIGITIZE Command	61
Reading Query Responses from the Oscilloscope	63
Reading Query Results into String Variables	64
Reading Query Results into Numeric Variables	64
Reading Definite-Length Block Query Response Data	64
Sending Multiple Queries and Reading Results	65
Checking Instrument Status	66
Other Ways of Sending Commands	67
Telnet Sockets	67
Sending SCPI Commands Using Browser Web Control	67

4 Commands Quick Reference

Command Summary	70
Syntax Elements	161
Number Format	161
<NL> (Line Terminator)	161
[] (Optional Syntax Terms)	161
{ } (Braces)	161
::= (Defined As)	161
< > (Angle Brackets)	162
... (Ellipsis)	162
n,...p (Value Ranges)	162
d (Digits)	162
Quoted ASCII String	162
Definite-Length Block Response Data	162

5 Common (*) Commands

*CLS (Clear Status)	169
*ESE (Standard Event Status Enable)	170
*ESR (Standard Event Status Register)	172
*IDN (Identification Number)	174
*LRN (Learn Device Setup)	175
*OPC (Operation Complete)	176
*OPT (Option Identification)	177

*RCL (Recall)	179
*RST (Reset)	180
*SAV (Save)	183
*SRE (Service Request Enable)	184
*STB (Read Status Byte)	186
*TRG (Trigger)	188
*TST (Self Test)	189
*WAI (Wait To Continue)	190

6 Root (:) Commands

:ACTivity	195
:AER (Arm Event Register)	196
:AUToscale	197
:AUToscale:AMODE	199
:AUToscale:CHANnels	200
:AUToscale:FDEBug	201
:BLANk	202
:DIGItize	203
:HWEnable (Hardware Event Enable Register)	205
:HWERegister:CONDition (Hardware Event Condition Register)	207
:HWERegister[:EVENT] (Hardware Event Event Register)	208
:MTEnable (Mask Test Event Enable Register)	209
:MTERegister[:EVENT] (Mask Test Event Event Register)	211
:OPEE (Operation Status Enable Register)	213
:OPERegister:CONDition (Operation Status Condition Register)	215
:OPERegister[:EVENT] (Operation Status Event Register)	217
:OVLenable (Overload Event Enable Register)	219
:OVLRegister (Overload Event Register)	221
:PRINt	223
:RUN	224
:SERial	225
:SINGle	226
:STATus	227
:STOP	228
:TER (Trigger Event Register)	229
:VIEW	230

7 :ACQuire Commands

:ACQuire:COMPLETE	233
:ACQuire:COUNt	234
:ACQuire:MODE	235

:ACQuire:POINTs 236
:ACQuire:RSIGnal 237
:ACQuire:SEGmented:ANALyze 238
:ACQuire:SEGmented:COUNt 239
:ACQuire:SEGmented:INDex 240
:ACQuire:SRATe 243
:ACQuire:TYPE 244

8 :BUS<n> Commands

:BUS<n>:BIT<m> 249
:BUS<n>:BITS 250
:BUS<n>:CLEar 252
:BUS<n>:DISPlay 253
:BUS<n>:LABel 254
:BUS<n>:MASK 255

9 :CALibrate Commands

:CALibrate:DATE 259
:CALibrate:LABel 260
:CALibrate:OUTPut 261
:CALibrate:PROTected 262
:CALibrate:STARt 263
:CALibrate:STATus 264
:CALibrate:TEMPerature 265
:CALibrate:TIME 266

10 :CHANnel<n> Commands

:CHANnel<n>:BWLimit 270
:CHANnel<n>:COUPLing 271
:CHANnel<n>:DISPlay 272
:CHANnel<n>:IMPedance 273
:CHANnel<n>:INVert 274
:CHANnel<n>:LABel 275
:CHANnel<n>:OFFSet 276
:CHANnel<n>:PROBe 277
:CHANnel<n>:PROBe:HEAD[:TYPE] 278
:CHANnel<n>:PROBe:ID 279
:CHANnel<n>:PROBe:SKEW 280
:CHANnel<n>:PROBe:STYPe 281
:CHANnel<n>:PROTection 282
:CHANnel<n>:RANGE 283

:CHANnel<n>:SCALE [284](#)
:CHANnel<n>:UNITs [285](#)
:CHANnel<n>:VERNier [286](#)

11 :COMPliance Commands

:COMPliance:USB:AUTosetup [288](#)
:COMPliance:USB:HUBS [289](#)
:COMPliance:USB:RUN [290](#)
:COMPliance:USB:SOURce:ADJacent [291](#)
:COMPliance:USB:SOURce:DIFFerential [292](#)
:COMPliance:USB:SOURce:DMINus [293](#)
:COMPliance:USB:SOURce:DPLus [294](#)
:COMPliance:USB:TEST [295](#)
:COMPliance:USB:TEST:CONNnection [296](#)
:COMPliance:USB:TEST:TYPE [297](#)

12 :DEMO Commands

:DEMO:FUNCtion [300](#)
:DEMO:FUNCtion:PHASE:PHASE [304](#)
:DEMO:OUTPut [305](#)

13 :DIGItal<d> Commands

:DIGItal<d>:DISPlay [309](#)
:DIGItal<d>:LABel [310](#)
:DIGItal<d>:POSition [311](#)
:DIGItal<d>:SIZE [312](#)
:DIGItal<d>:THReShold [313](#)

14 :DISPlay Commands

:DISPlay:ANNotation [317](#)
:DISPlay:ANNotation:BACKground [318](#)
:DISPlay:ANNotation:COLor [319](#)
:DISPlay:ANNotation:TEXT [320](#)
:DISPlay:CLEar [321](#)
:DISPlay:DATA [322](#)
:DISPlay:LABel [323](#)
:DISPlay:LABList [324](#)
:DISPlay:MENU [325](#)
:DISPlay:SIDebar [326](#)
:DISPlay:PERSistence [327](#)
:DISPlay:VECTors [328](#)

15 :DVM Commands

:DVM:ARAnge 330
:DVM:CURREnt 331
:DVM:ENABLE 332
:DVM:FREQuency 333
:DVM:MODE 334
:DVM:SOURce 335

16 :EXTernal Trigger Commands

:EXTernal:BWLimit 338
:EXTernal:PROBe 339
:EXTernal:RANGE 340
:EXTernal:UNITS 341

17 :FUNCtion<m> Commands

:FUNCtion<m>:AVERage:COUNT 347
:FUNCtion<m>:BUS:CLOCK 348
:FUNCtion<m>:BUS:SLOPe 349
:FUNCtion<m>:BUS:YINCrement 350
:FUNCtion<m>:BUS:YORigin 351
:FUNCtion<m>:BUS:YUNits 352
:FUNCtion<m>:DISPlay 353
:FUNCtion<m>[:FFT]:CENTer 354
:FUNCtion<m>[:FFT]:SPAN 355
:FUNCtion<m>[:FFT]:VTPe 356
:FUNCtion<m>[:FFT]:WINDOW 357
:FUNCtion<m>:FREQuency:HIGHpass 358
:FUNCtion<m>:FREQuency:LOWPass 359
:FUNCtion<m>:INTegrate:IOFFset 360
:FUNCtion<m>:LINEar:GAIN 361
:FUNCtion<m>:LINEar:OFFSet 362
:FUNCtion<m>:OFFSet 363
:FUNCtion<m>:OPERation 364
:FUNCtion<m>:RANGE 367
:FUNCtion<m>:REFerence 368
:FUNCtion<m>:SCALE 369
:FUNCtion<m>:SOURce1 370
:FUNCtion<m>:SOURce2 372
:FUNCtion<m>:TRENd:MEASurement 373

18 :HARDcopy Commands

:HARDcopy:AREA 377
:HARDcopy:APRinter 378
:HARDcopy:FACTors 379
:HARDcopy:FFEd 380
:HARDcopy:INKSaver 381
:HARDcopy:LAYout 382
:HARDcopy:NETWork:ADDRess 383
:HARDcopy:NETWork:APPLy 384
:HARDcopy:NETWork:DOMain 385
:HARDcopy:NETWork:PASSword 386
:HARDcopy:NETWork:SLOT 387
:HARDcopy:NETWork:USERname 388
:HARDcopy:PAlette 389
:HARDcopy:PRINter:LIST 390
:HARDcopy:STARt 391

19 :LISTer Commands

:LISTer:DATA 394
:LISTer:DISPlay 395
:LISTer:REFerence 396

20 :MARKer Commands

:MARKer:DYDX 400
:MARKer:MODE 401
:MARKer:X1Position 402
:MARKer:X1Y1source 403
:MARKer:X2Position 404
:MARKer:X2Y2source 405
:MARKer:XDELta 406
:MARKer:XUNits 407
:MARKer:XUNits:USE 408
:MARKer:Y1Position 409
:MARKer:Y2Position 410
:MARKer:YDELta 411
:MARKer:YUNits 412
:MARKer:YUNits:USE 413

21 :MEASure Commands

:MEASure:ALL 430
:MEASure:AREa 431

:MEASure:BWIDth 432
:MEASure:CLEar 433
:MEASure:COUNter 434
:MEASure:DEFine 436
:MEASure:DELay 439
:MEASure:DUAL:CHARge 441
:MEASure:DUAL:VAMPlitude 442
:MEASure:DUAL:VAverage 443
:MEASure:DUAL:VBASe 444
:MEASure:DUAL:VPP 445
:MEASure:DUAL:VRMS 446
:MEASure:DUTYcycle 447
:MEASure:FALLtime 448
:MEASure:FREQuency 449
:MEASure:NDUTy 450
:MEASure:NEDGes 451
:MEASure:NPULses 452
:MEASure:NWIDth 453
:MEASure:OVERshoot 454
:MEASure:PEDGes 456
:MEASure:PERiod 457
:MEASure:PHASe 458
:MEASure:PPULses 459
:MEASure:PRESHoot 460
:MEASure:PWIDth 461
:MEASure:RESults 462
:MEASure:RISetime 465
:MEASure:SDEViation 466
:MEASure:SHOW 467
:MEASure:SOURce 468
:MEASure:STATistics 470
:MEASure:STATistics:DISPLAY 471
:MEASure:STATistics:INCRement 472
:MEASure:STATistics:MCOUNT 473
:MEASure:STATistics:RESET 474
:MEASure:STATistics:RSDeviation 475
:MEASure:TEDGE 476
:MEASure:TVALUE 479
:MEASure:VAMPlitude 481
:MEASure:VAverage 482
:MEASure:VBASe 483
:MEASure:VMAX 484

:MEASure:VMIN 485
:MEASure:VPP 486
:MEASure:VRATio 487
:MEASure:VRMS 488
:MEASure:VTIMe 489
:MEASure:VTOP 490
:MEASure:WINDOW 491
:MEASure:XMAX 492
:MEASure:XMIN 493

22 :MEASure Power Commands

:MEASure:ANGLE 499
:MEASure:APPARENT 500
:MEASure:CPLoss 501
:MEASure:CRESt 502
:MEASure:EFFiciency 503
:MEASure:ELOSSs 504
:MEASure:FACTOr 505
:MEASure:IPower 506
:MEASure:OFFTime 507
:MEASure:ONTIME 508
:MEASure:OPower 509
:MEASure:PCURrent 510
:MEASure:PLOSSs 511
:MEASure:REACtive 512
:MEASure:REAL 513
:MEASure:RIPPLE 514
:MEASure:TRESPonse 515

23 :MTEST Commands

:MTEST:ALL 522
:MTEST:AMASK:CREate 523
:MTEST:AMASK:SOURce 524
:MTEST:AMASK:UNITS 525
:MTEST:AMASK:XDELta 526
:MTEST:AMASK:YDELta 527
:MTEST:COUNT:FWAVEforms 528
:MTEST:COUNT:RESet 529
:MTEST:COUNT:TIME 530
:MTEST:COUNT:WAVEforms 531
:MTEST:DATA 532

:MTESt:DELete 533
:MTESt:ENABLE 534
:MTESt:LOCK 535
:MTESt:RMODE 536
:MTESt:RMODE:FACTion:MEASure 537
:MTESt:RMODE:FACTion:PRINT 538
:MTESt:RMODE:FACTion:SAVE 539
:MTESt:RMODE:FACTion:STOP 540
:MTESt:RMODE:SIGMa 541
:MTESt:RMODE:TIME 542
:MTESt:RMODE:WAveforms 543
:MTESt:SCALe:BIND 544
:MTESt:SCALe:X1 545
:MTESt:SCALe:XDELta 546
:MTESt:SCALe:Y1 547
:MTESt:SCALe:Y2 548
:MTESt:SOURce 549
:MTESt:TITLe 550

24 :POD Commands

:POD<n>:DISPlay 553
:POD<n>:SIZE 554
:POD<n>:THReShold 555

25 :POWer Commands

:POWer:DESKew 562
:POWer:EFFiciency:APPLy 563
:POWer:ENABLE 564
:POWer:HARMonics:APPLy 565
:POWer:HARMonics:DATA 566
:POWer:HARMonics:DISPlay 567
:POWer:HARMonics:FAILcount 568
:POWer:HARMonics:LINE 569
:POWer:HARMonics:POWerfactor 570
:POWer:HARMonics:RUNCount 571
:POWer:HARMonics:STANDARD 572
:POWer:HARMonics:STATus 573
:POWer:HARMonics:THD 574
:POWer:INRush:APPLy 575
:POWer:INRush:EXIT 576
:POWer:INRush:NEXT 577

:POWER:MODulation:APPLy 578
:POWER:MODulation:SOURce 579
:POWER:MODulation:TYPE 580
:POWER:ONOFF:APPLy 581
:POWER:ONOFF:EXIT 582
:POWER:ONOFF:NEXT 583
:POWER:ONOFF:TEST 584
:POWER:PSRR:APPLy 585
:POWER:PSRR:FREQuency:MAXimum 586
:POWER:PSRR:FREQuency:MINimum 587
:POWER:PSRR:RMAXimum 588
:POWER:QUALity:APPLy 589
:POWER:RIPPLE:APPLy 590
:POWER:SIGNals:AUTosetup 591
:POWER:SIGNals:CYCLeS:HARMonics 592
:POWER:SIGNals:CYCLeS:QUALity 593
:POWER:SIGNals:DURation:EFFiciency 594
:POWER:SIGNals:DURation:MODulation 595
:POWER:SIGNals:DURation:ONOFF:OFF 596
:POWER:SIGNals:DURation:ONOFF:ON 597
:POWER:SIGNals:DURation:RIPPLE 598
:POWER:SIGNals:DURation:TRANsient 599
:POWER:SIGNals:IEXPected 600
:POWER:SIGNals:OVERshoot 601
:POWER:SIGNals:VMAXimum:INRush 602
:POWER:SIGNals:VMAXimum:ONOFF:OFF 603
:POWER:SIGNals:VMAXimum:ONOFF:ON 604
:POWER:SIGNals:VSteady:ONOFF:OFF 605
:POWER:SIGNals:VSteady:ONOFF:ON 606
:POWER:SIGNals:VSteady:TRANsient 607
:POWER:SIGNals:SOURce:CURREnt<i> 608
:POWER:SIGNals:SOURce:VOLTage<i> 609
:POWER:SLEW:APPLy 610
:POWER:SLEW:SOURce 611
:POWER:SWITch:APPLy 612
:POWER:SWITch:CONDuction 613
:POWER:SWITch:IREFerence 614
:POWER:SWITch:RDS 615
:POWER:SWITch:VCE 616
:POWER:SWITch:VREFerence 617
:POWER:TRANsient:APPLy 618
:POWER:TRANsient:EXIT 619

:POWer:TRANsient:IINitial 620
:POWer:TRANsient:INEW 621
:POWer:TRANsient:NEXT 622

26 :RECall Commands

:RECall:ARBitrary[:STARt] 625
:RECall:DBC[:STARt] 626
:RECall:FILEname 627
:RECall:MASK[:STARt] 628
:RECall:PWD 629
:RECall:SETUp[:STARt] 630
:RECall:WMEMory<r>[:STARt] 631

27 :SAVE Commands

:SAVE:ARBitrary[:STARt] 636
:SAVE:COMPliance:USB[:STARt] 637
:SAVE:FILEname 638
:SAVE:IMAGe[:STARt] 639
:SAVE:IMAGe:FACTors 640
:SAVE:IMAGe:FORMAT 641
:SAVE:IMAGe:INKSaver 642
:SAVE:IMAGe:PAlette 643
:SAVE:LISTER[:STARt] 644
:SAVE:MASK[:STARt] 645
:SAVE:MULTi[:STARt] 646
:SAVE:POWER[:STARt] 647
:SAVE:PWD 648
:SAVE:SETUp[:STARt] 649
:SAVE:WAVEform[:STARt] 650
:SAVE:WAVEform:FORMAT 651
:SAVE:WAVEform:LENGTH 652
:SAVE:WAVEform:LENGTH:MAX 653
:SAVE:WAVEform:SEGmented 654
:SAVE:WMEMory:SOURce 655
:SAVE:WMEMory[:STARt] 656

28 :SBUS<n> Commands

General :SBUS<n> Commands 659
:SBUS<n>:DISPlay 660
:SBUS<n>:MODE 661
:SBUS<n>:A429 Commands 662

:SBUS<n>:A429:AUTosetup 664
:SBUS<n>:A429:BASE 665
:SBUS<n>:A429:COUNt:ERRor 666
:SBUS<n>:A429:COUNt:RESet 667
:SBUS<n>:A429:COUNt:WORD 668
:SBUS<n>:A429:FORMat 669
:SBUS<n>:A429:SIGNal 670
:SBUS<n>:A429:SOURce 671
:SBUS<n>:A429:SPEed 672
:SBUS<n>:A429:TRIGger:LABEL 673
:SBUS<n>:A429:TRIGger:PATTern:DATA 674
:SBUS<n>:A429:TRIGger:PATTern:SDI 675
:SBUS<n>:A429:TRIGger:PATTern:SSM 676
:SBUS<n>:A429:TRIGger:RANGe 677
:SBUS<n>:A429:TRIGger:TYPE 678

:SBUS<n>:CAN Commands 680
:SBUS<n>:CAN:COUNt:ERRor 682
:SBUS<n>:CAN:COUNt:OVERload 683
:SBUS<n>:CAN:COUNt:RESet 684
:SBUS<n>:CAN:COUNt:TOTal 685
:SBUS<n>:CAN:COUNt:UTILization 686
:SBUS<n>:CAN:DISPlay 687
:SBUS<n>:CAN:SAMPLEpoint 688
:SBUS<n>:CAN:SIGNal:BAUDrate 689
:SBUS<n>:CAN:SIGNal:DEFinition 690
:SBUS<n>:CAN:SOURce 691
:SBUS<n>:CAN:TRIGger 692
:SBUS<n>:CAN:TRIGger:PATTern:DATA 695
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth 696
:SBUS<n>:CAN:TRIGger:PATTern:ID 697
:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE 698
:SBUS<n>:CAN:TRIGger:SYMBOLic:MESSage 699
:SBUS<n>:CAN:TRIGger:SYMBOLic:SIGNal 700
:SBUS<n>:CAN:TRIGger:SYMBOLic:VALue 701

:SBUS<n>:FLEXray Commands 702
:SBUS<n>:FLEXray:AUTosetup 704
:SBUS<n>:FLEXray:BAUDrate 705
:SBUS<n>:FLEXray:CHANnel 706
:SBUS<n>:FLEXray:COUNt:NULL 707
:SBUS<n>:FLEXray:COUNt:RESet 708
:SBUS<n>:FLEXray:COUNt:SYNC 709

:SBUS<n>:FLEXray:COUNt:TOTal 710
:SBUS<n>:FLEXray:SOURce 711
:SBUS<n>:FLEXray:TRIGger 712
:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE 713
:SBUS<n>:FLEXray:TRIGger:EVENt:AUToset 714
:SBUS<n>:FLEXray:TRIGger:EVENt:BSS:ID 715
:SBUS<n>:FLEXray:TRIGger:EVENt:TYPE 716
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase 717
:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition 718
:SBUS<n>:FLEXray:TRIGger:FRAMe:ID 719
:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE 720

:SBUS<n>:I2S Commands 721
 :SBUS<n>:I2S:ALIGNment 723
 :SBUS<n>:I2S:BASE 724
 :SBUS<n>:I2S:CLOCK:SLOPe 725
 :SBUS<n>:I2S:RWIDth 726
 :SBUS<n>:I2S:SOURce:CLOCK 727
 :SBUS<n>:I2S:SOURce:DATA 728
 :SBUS<n>:I2S:SOURce:WSELect 729
 :SBUS<n>:I2S:TRIGger 730
 :SBUS<n>:I2S:TRIGger:AUDIO 732
 :SBUS<n>:I2S:TRIGger:PATTERn:DATA 733
 :SBUS<n>:I2S:TRIGger:PATTERn:FORMAT 735
 :SBUS<n>:I2S:TRIGger:RANGe 736
 :SBUS<n>:I2S:TVIDth 738
 :SBUS<n>:I2S:WSLow 739

:SBUS<n>:IIC Commands 740
 :SBUS<n>:IIC:ASIZe 741
 :SBUS<n>:IIC[:SOURce]:CLOCK 742
 :SBUS<n>:IIC[:SOURce]:DATA 743
 :SBUS<n>:IIC:TRIGger:PATTERn:ADDResS 744
 :SBUS<n>:IIC:TRIGger:PATTERn:DATA 745
 :SBUS<n>:IIC:TRIGger:PATTERn:DATA2 746
 :SBUS<n>:IIC:TRIGger:QUALifier 747
 :SBUS<n>:IIC:TRIGger[:TYPE] 748

:SBUS<n>:LIN Commands 750
 :SBUS<n>:LIN:PARity 752
 :SBUS<n>:LIN:SAMPLEpoint 753
 :SBUS<n>:LIN:SIGNal:BAUDrate 754
 :SBUS<n>:LIN:SOURce 755

:SBUS<n>:LIN:STANDARD 756
:SBUS<n>:LIN:SYNCbreak 757
:SBUS<n>:LIN:TRIGger 758
:SBUS<n>:LIN:TRIGger:ID 759
:SBUS<n>:LIN:TRIGger:PATTERn:DATA 760
:SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGTH 762
:SBUS<n>:LIN:TRIGger:PATTERn:FORMAT 763

:SBUS<n>:M1553 Commands 764
:SBUS<n>:M1553:AUTosetup 765
:SBUS<n>:M1553:BASE 766
:SBUS<n>:M1553:SOURce 767
:SBUS<n>:M1553:TRIGger:PATTERn:DATA 768
:SBUS<n>:M1553:TRIGger:RTA 769
:SBUS<n>:M1553:TRIGger:TYPE 770

:SBUS<n>:SPI Commands 771
:SBUS<n>:SPI:BITorder 773
:SBUS<n>:SPI:CLOCK:SLOPe 774
:SBUS<n>:SPI:CLOCK:TIMEout 775
:SBUS<n>:SPI:FRAMing 776
:SBUS<n>:SPI:SOURce:CLOCK 777
:SBUS<n>:SPI:SOURce:FRAME 778
:SBUS<n>:SPI:SOURce:MISO 779
:SBUS<n>:SPI:SOURce:MOStI 780
:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA 781
:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTH 782
:SBUS<n>:SPI:TRIGger:PATTERn:MOStI:DATA 783
:SBUS<n>:SPI:TRIGger:PATTERn:MOStI:WIDTH 784
:SBUS<n>:SPI:TRIGger:TYPE 785
:SBUS<n>:SPI:WIDTH 786

:SBUS<n>:UART Commands 787
:SBUS<n>:UART:BASE 790
:SBUS<n>:UART:BAUDrate 791
:SBUS<n>:UART:BITorder 792
:SBUS<n>:UART:COUNT:ERRor 793
:SBUS<n>:UART:COUNT:RESET 794
:SBUS<n>:UART:COUNT:RXFRAMES 795
:SBUS<n>:UART:COUNT:TXFRAMES 796
:SBUS<n>:UART:FRAMing 797
:SBUS<n>:UART:PARity 798
:SBUS<n>:UART:POLarity 799

:SBUS<n>:UART:SOURce:RX 800
:SBUS<n>:UART:SOURce:TX 801
:SBUS<n>:UART:TRIGger:BASE 802
:SBUS<n>:UART:TRIGger:BURSt 803
:SBUS<n>:UART:TRIGger:DATA 804
:SBUS<n>:UART:TRIGger:IDLE 805
:SBUS<n>:UART:TRIGger:QUALifier 806
:SBUS<n>:UART:TRIGger:TYPE 807
:SBUS<n>:UART:WIDTh 808

.SBUS<n>:USB Commands 809
:SBUS<n>:USB:BASE 812
:SBUS<n>:USB:SOURce:DMINus 813
:SBUS<n>:USB:SOURce:DPLus 814
:SBUS<n>:USB:SOURce:DIFFerential 815
:SBUS<n>:USB:SPEEd 816
:SBUS<n>:USB:TRIGger 817
:SBUS<n>:USB:TRIGger:ADDRess 818
:SBUS<n>:USB:TRIGger:CRC 819
:SBUS<n>:USB:TRIGger:DATA 820
:SBUS<n>:USB:TRIGger:DATA:LENgth 821
:SBUS<n>:USB:TRIGger:ENDPoint 822
:SBUS<n>:USB:TRIGger:ET 823
:SBUS<n>:USB:TRIGger:FRAMe 824
:SBUS<n>:USB:TRIGger:HADDress 825
:SBUS<n>:USB:TRIGger:PCHeck 826
:SBUS<n>:USB:TRIGger:PID:DATA 827
:SBUS<n>:USB:TRIGger:PID:HANDshake 828
:SBUS<n>:USB:TRIGger:PID:SPECial 829
:SBUS<n>:USB:TRIGger:PID:TOKen 830
:SBUS<n>:USB:TRIGger:PORT 831
:SBUS<n>:USB:TRIGger:SC 832
:SBUS<n>:USB:TRIGger:SEU 833

29 :SEARch Commands

General :SEARch Commands 836
:SEARch:COUNT 837
:SEARch:EVENT 838
:SEARch:MODE 839
:SEARch:STATe 840

.SEARch:EDGE Commands 841

:SEARch:EDGE:SLOPe	842
:SEARch:EDGE:SOURce	843
:SEARch:GLITch Commands	844
:SEARch:GLITch:GREaterthan	845
:SEARch:GLITch:LESSthan	846
:SEARch:GLITch:POLarity	847
:SEARch:GLITch:QUALifier	848
:SEARch:GLITch:RANGE	849
:SEARch:GLITch:SOURce	850
:SEARch:RUNT Commands	851
:SEARch:RUNT:POLarity	852
:SEARch:RUNT:QUALifier	853
:SEARch:RUNT:SOURce	854
:SEARch:RUNT:TIME	855
:SEARch:TRANSition Commands	856
:SEARch:TRANSition:QUALifier	857
:SEARch:TRANSition:SLOPe	858
:SEARch:TRANSition:SOURce	859
:SEARch:TRANSition:TIME	860
:SEARch:SERial:A429 Commands	861
:SEARch:SERial:A429:LABEL	862
:SEARch:SERial:A429:MODE	863
:SEARch:SERial:A429:PATTern:DATA	864
:SEARch:SERial:A429:PATTern:SDI	865
:SEARch:SERial:A429:PATTern:SSM	866
:SEARch:SERial:CAN Commands	867
:SEARch:SERial:CAN:MODE	868
:SEARch:SERial:CAN:PATTern:DATA	870
:SEARch:SERial:CAN:PATTern:DATA:LENGTH	871
:SEARch:SERial:CAN:PATTern:ID	872
:SEARch:SERial:CAN:PATTern:ID:MODE	873
:SEARch:SERial:CAN:SYMBolic:MESSAge	874
:SEARch:SERial:CAN:SYMBolic:SIGNAl	875
:SEARch:SERial:CAN:SYMBolic:VALUe	876
:SEARch:SERial:FLEXray Commands	877
:SEARch:SERial:FLEXray:CYCLE	878
:SEARch:SERial:FLEXray:DATA	879
:SEARch:SERial:FLEXray:DATA:LENGTH	880
:SEARch:SERial:FLEXray:FRAMe	881

:SEARch:SERial:FLEXray:MODE	882
:SEARch:SERial:I2S Commands	883
:SEARch:SERial:I2S:AUDIO	884
:SEARch:SERial:I2S:MODE	885
:SEARch:SERial:I2S:PATTERn:DATA	886
:SEARch:SERial:I2S:PATTERn:FORMAT	887
:SEARch:SERial:I2S:RANGE	888
:SEARch:SERial:IIC Commands	889
:SEARch:SERial:IIC:MODE	890
:SEARch:SERial:IIC:PATTERn:ADDReSS	892
:SEARch:SERial:IIC:PATTERn:DATA	893
:SEARch:SERial:IIC:PATTERn:DATA2	894
:SEARch:SERial:IIC:QUALifier	895
:SEARch:SERial:LIN Commands	896
:SEARch:SERial:LIN:ID	897
:SEARch:SERial:LIN:MODE	898
:SEARch:SERial:LIN:PATTERn:DATA	899
:SEARch:SERial:LIN:PATTERn:DATA:LENGTH	900
:SEARch:SERial:LIN:PATTERn:FORMAT	901
:SEARch:SERial:M1553 Commands	902
:SEARch:SERial:M1553:MODE	903
:SEARch:SERial:M1553:PATTERn:DATA	904
:SEARch:SERial:M1553:RTA	905
:SEARch:SERial:SPI Commands	906
:SEARch:SERial:SPI:MODE	907
:SEARch:SERial:SPI:PATTERn:DATA	908
:SEARch:SERial:SPI:PATTERn:WIDTH	909
:SEARch:SERial:UART Commands	910
:SEARch:SERial:UART:DATA	911
:SEARch:SERial:UART:MODE	912
:SEARch:SERial:UART:QUALifier	913
:SEARch:SERial:USB Commands	914
:SEARch:SERial:USB:MODE	916
:SEARch:SERial:USB:ADDReSS	917
:SEARch:SERial:USB:CRC	918
:SEARch:SERial:USB:DATA	919
:SEARch:SERial:USB:DATA:LENGTH	920
:SEARch:SERial:USB:ENDPoint	921
:SEARch:SERial:USB:ET	922

:SEARch:SERial:USB:FRAMe 923
:SEARch:SERial:USB:HADDress 924
:SEARch:SERial:USB:PID:DATA 925
:SEARch:SERial:USB:PID:HANDshake 926
:SEARch:SERial:USB:PID:SPECial 927
:SEARch:SERial:USB:PID:TOKen 928
:SEARch:SERial:USB:PORT 929
:SEARch:SERial:USB:SC 930
:SEARch:SERial:USB:SEU 931

30 :SYSTem Commands

:SYSTem:DATE 935
:SYSTem:DSP 936
:SYSTem:ERRor 937
:SYSTem:LOCK 938
:SYSTem:PRESet 939
:SYSTem:PROTection:LOCK 942
:SYSTem:SETup 943
:SYSTem:TIME 945
:SYSTem:TOUCH 946

31 :TIMEbase Commands

:TIMEbase:MODE 949
:TIMEbase:POSIon 950
:TIMEbase:RANGe 951
:TIMEbase:REFClock 952
:TIMEbase:REFerence 953
:TIMEbase:SCALE 954
:TIMEbase:VERNier 955
:TIMEbase:WINDOW:POSIon 956
:TIMEbase:WINDOW:RANGe 957
:TIMEbase:WINDOW:SCALE 958

32 :TRIGger Commands

General :TRIGger Commands 961
:TRIGger:FORCe 962
:TRIGger:HFReject 963
:TRIGger:HOLDoff 964
:TRIGger:LEVel:ASETup 965
:TRIGger:LEVel:HIGH 966
:TRIGger:LEVel:LOW 967

:TRIGger:MODE 968
:TRIGger:NREject 969
:TRIGger:SWEep 970

:TRIGger:DELay Commands 971
:TRIGger:DELay:ARM:SLOPe 972
:TRIGger:DELay:ARM:SOURce 973
:TRIGger:DELay:TDELay:TIME 974
:TRIGger:DELay:TRIGger:COUNt 975
:TRIGger:DELay:TRIGger:SLOPe 976
:TRIGger:DELay:TRIGger:SOURce 977

:TRIGger:EBURst Commands 978
:TRIGger:EBURst:COUNt 979
:TRIGger:EBURst:IDLE 980
:TRIGger:EBURst:SLOPe 981
:TRIGger:EBURst:SOURce 982

:TRIGger[:EDGE] Commands 983
:TRIGger[:EDGE]:COUpling 985
:TRIGger[:EDGE]:LEVel 986
:TRIGger[:EDGE]:REJect 987
:TRIGger[:EDGE]:SLOPe 988
:TRIGger[:EDGE]:SOURce 989

:TRIGger:GLITch Commands 990
:TRIGger:GLITch:GREaterthan 992
:TRIGger:GLITch:LESSthan 993
:TRIGger:GLITch:LEVel 994
:TRIGger:GLITch:POLarity 995
:TRIGger:GLITch:QUALifier 996
:TRIGger:GLITch:RANGE 997
:TRIGger:GLITch:SOURce 998

:TRIGger:OR Commands 999
:TRIGger:OR 1000

:TRIGger:PATTERn Commands 1001
:TRIGger:PATTERn 1002
:TRIGger:PATTERn:FORMAT 1004
:TRIGger:PATTERn:GREaterthan 1005
:TRIGger:PATTERn:LESSthan 1006
:TRIGger:PATTERn:QUALifier 1007
:TRIGger:PATTERn:RANGE 1009

:TRIGger:RUNT Commands 1010

:TRIGger:RUNT:POLarity 1011
:TRIGger:RUNT:QUALifier 1012
:TRIGger:RUNT:SOURce 1013
:TRIGger:RUNT:TIME 1014

:TRIGger:SHOLd Commands 1015
:TRIGger:SHOLd:SLOPe 1016
:TRIGger:SHOLd:SOURce:CLOCK 1017
:TRIGger:SHOLd:SOURce:DATA 1018
:TRIGger:SHOLd:TIME:HOLD 1019
:TRIGger:SHOLd:TIME:SETup 1020

:TRIGger:TRANSition Commands 1021
:TRIGger:TRANSition:QUALifier 1022
:TRIGger:TRANSition:SLOPe 1023
:TRIGger:TRANSition:SOURce 1024
:TRIGger:TRANSition:TIME 1025

:TRIGger:TV Commands 1026
:TRIGger:TV:LINE 1027
:TRIGger:TV:MODE 1028
:TRIGger:TV:POLarity 1029
:TRIGger:TV:SOURce 1030
:TRIGger:TV:STANDARD 1031
:TRIGger:TV:UDTV:ENUMber 1032
:TRIGger:TV:UDTV:HSync 1033
:TRIGger:TV:UDTV:HTime 1034
:TRIGger:TV:UDTV:PGTHan 1035

:TRIGger:ZONE Commands 1036
:TRIGger:ZONE:SOURce 1037
:TRIGger:ZONE:STATe 1038
:TRIGger:ZONE<n>:MODE 1039
:TRIGger:ZONE<n>:PLACement 1040
:TRIGger:ZONE<n>:VALidity 1041
:TRIGger:ZONE<n>:STATe 1042

33 :WAVeform Commands

:WAVeform:BYTeorder 1051
:WAVeform:COUNT 1052
:WAVeform:DATA 1053
:WAVeform:FORMAT 1055
:WAVeform:POINts 1056
:WAVeform:POINts:MODE 1058

:WAVEform:PREamble 1060
:WAVEform:SEGmented:COUNt 1063
:WAVEform:SEGmented:TTAG 1064
:WAVEform:SOURce 1065
:WAVEform:SOURce:SUBSource 1069
:WAVEform:TYPE 1070
:WAVEform:UNSIGNED 1071
:WAVEform:VIEW 1072
:WAVEform:XINCrement 1073
:WAVEform:XORigin 1074
:WAVEform:XREFerence 1075
:WAVEform:YINCrement 1076
:WAVEform:YORigin 1077
:WAVEform:YREFerence 1078

34 :WGEN<w> Commands

:WGEN<w>:ARBitrary:BYTeorder 1083
:WGEN<w>:ARBitrary:DATA 1084
:WGEN<w>:ARBitrary:DATA:ATTRibute:POINTs 1085
:WGEN<w>:ARBitrary:DATA:CLEar 1086
:WGEN<w>:ARBitrary:DATA:DAC 1087
:WGEN<w>:ARBitrary:INTERpolate 1088
:WGEN<w>:ARBitrary:STORe 1089
:WGEN<w>:FREQuency 1090
:WGEN<w>:FUNCTION 1091
:WGEN<w>:FUNCTION:PULSe:WIDTH 1094
:WGEN<w>:FUNCTION:RAMP:SYMMetry 1095
:WGEN<w>:FUNCTION:SQUare:DCYCLE 1096
:WGEN<w>:MODulation:AM:DEPTH 1097
:WGEN<w>:MODulation:AM:FREQuency 1098
:WGEN<w>:MODulation:FM:DEViation 1099
:WGEN<w>:MODulation:FM:FREQuency 1100
:WGEN<w>:MODulation:FSKey:FREQuency 1101
:WGEN<w>:MODulation:FSKey:RATE 1102
:WGEN<w>:MODulation:FUNCTION 1103
:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry 1104
:WGEN<w>:MODulation:NOISE 1105
:WGEN<w>:MODulation:STATe 1106
:WGEN<w>:MODulation:TYPE 1107
:WGEN<w>:OUTPut 1109
:WGEN<w>:OUTPut:LOAD 1110

:WGEN<w>:OUTPut:POLarity 1111
:WGEN<w>:PERiod 1112
:WGEN<w>:RST 1113
:WGEN<w>:TRACK 1114
:WGEN<w>:TRACK:AMPLitude 1115
:WGEN<w>:TRACK:CSIGnal 1116
:WGEN<w>:TRACK:FREQuency 1117
:WGEN<w>:VOLTage 1118
:WGEN<w>:VOLTage:HIGH 1119
:WGEN<w>:VOLTage:LOW 1120
:WGEN<w>:VOLTage:OFFSet 1121

35 :WMEMory<r> Commands

:WMEMory<r>:CLEar 1125
:WMEMory<r>:DISPlay 1126
:WMEMory<r>:LABel 1127
:WMEMory<r>:SAVE 1128
:WMEMory<r>:SKEW 1129
:WMEMory<r>:YOFFset 1130
:WMEMory<r>:YRANge 1131
:WMEMory<r>:YScale 1132

36 Obsolete and Discontinued Commands

:CHANnel:ACTivity 1140
:CHANnel:LABEL 1141
:CHANnel:THRehold 1142
:CHANnel2:SKEW 1143
:CHANnel<n>:INPut 1144
:CHANnel<n>:PMODE 1145
:DISPlay:CONNect 1146
:DISPlay:ORDer 1147
:ERASE 1148
:EXternal:PMODE 1149
:FUNCTION:GOFT:OPERation 1150
:FUNCTION:GOFT:SOURce1 1151
:FUNCTION:GOFT:SOURce2 1152
:FUNCTION:SOURce 1153
:FUNCTION:VIEW 1154
:HARDcopy:DESTination 1155
:HARDcopy:FILEname 1156
:HARDcopy:GRAYscale 1157

:HARDcopy:IGColors 1158
:HARDcopy:PDRiver 1159
:MEASure:LOWER 1160
:MEASure:SCRatch 1161
:MEASure:TDELta 1162
:MEASure:THResholds 1163
:MEASure:TMAX 1164
:MEASure:TMIN 1165
:MEASure:TSTArt 1166
:MEASure:TSTOp 1167
:MEASure:TVOLT 1168
:MEASure:UPPer 1170
:MEASure:VDELta 1171
:MEASure:VSTArt 1172
:MEASure:VSTOp 1173
:MTEST:AMASK:{SAVE | STORe} 1174
:MTEST:AVERage 1175
:MTEST:AVERage:COUNt 1176
:MTEST:LOAD 1177
:MTEST:RUMode 1178
:MTEST:RUMode:SOFailure 1179
:MTEST:{STARt | STOP} 1180
:MTEST:TRIGger:SOURce 1181
:PRINt? 1182
:SAVE:IMAGE:AREA 1184
:SBUS<n>:LIN:SIGNal:DEFinition 1185
:SBUS<n>:SPI:SOURce:DATA 1186
:SYSTem:MENU 1187
:TIMEbase:DELay 1188
:TRIGger:THReshold 1189
:TRIGger:TV:TVMode 1190

37 Error Messages

38 Status Reporting

Status Reporting Data Structures 1201
Status Byte Register (STB) 1203
Service Request Enable Register (SRE) 1205
Trigger Event Register (TER) 1206
Output Queue 1207

Message Queue	1208
(Standard) Event Status Register (ESR)	1209
(Standard) Event Status Enable Register (ESE)	1210
Error Queue	1211
Operation Status Event Register (:OPERegister[:EVENT])	1212
Operation Status Condition Register (:OPERegister:CONDITION)	1213
Arm Event Register (AER)	1214
Overload Event Register (:OVLRegister)	1215
Hardware Event Event Register (:HWERegister[:EVENT])	1216
Hardware Event Condition Register (:HWERegister:CONDITION)	1217
Mask Test Event Event Register (:MTERegister[:EVENT])	1218
Clearing Registers and Queues	1219
Status Reporting Decision Chart	1220

39 Synchronizing Acquisitions

Synchronization in the Programming Flow	1222
Set Up the Oscilloscope	1222
Acquire a Waveform	1222
Retrieve Results	1222
Blocking Synchronization	1223
Polling Synchronization With Timeout	1224
Synchronizing with a Single-Shot Device Under Test (DUT)	1226
Synchronization with an Averaging Acquisition	1228

40 More About Oscilloscope Commands

Command Classifications	1232
Core Commands	1232
Non-Core Commands	1232
Obsolete Commands	1232
Valid Command/Query Strings	1233
Program Message Syntax	1233
Duplicate Mnemonics	1237
Tree Traversal Rules and Multiple Commands	1237
Query Return Values	1239
All Oscilloscope Commands Are Sequential	1240

41 Programming Examples

VISA COM Examples	1242
VISA COM Example in Visual Basic	1242
VISA COM Example in C#	1251
VISA COM Example in Visual Basic .NET	1260
VISA COM Example in Python	1268
VISA Examples	1275
VISA Example in C	1275
VISA Example in Visual Basic	1284
VISA Example in C#	1294
VISA Example in Visual Basic .NET	1305
VISA Example in Python	1315
SICL Examples	1322
SICL Example in C	1322
SICL Example in Visual Basic	1331
SCPI.NET Examples	1342
SCPI.NET Example in C#	1342
SCPI.NET Example in Visual Basic .NET	1348
SCPI.NET Example in IronPython	1354

Index

1 **What's New**

- What's New in Version 3.20 [34](#)
- What's New in Version 3.10 [36](#)
- What's New in Version 3.01 [37](#)
- Version 3.00 at Introduction [38](#)
- Command Differences From 3000 X-Series Oscilloscopes [39](#)



What's New in Version 3.20

New features in version 3.20 of the InfiniiVision 4000 X-Series oscilloscope software are:

- Being able to load CAN symbolic data from a *.dbc file into the oscilloscope, display it in the decode, and use it to set up triggers and protocol decode searches.
- LIN triggering and decode updates.

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

New Commands

Command	Description
:COMPliance:USB:HUBS (see page 289)	Specifies the number of internal hubs between the host and the test point.
:MARKer:DYDX? (see page 400)	Returns the cursor ΔY/ΔX value.
:MEASure:NDUTy (see page 450)	Negative duty cycle measurement.
:RECall:DBC[:STARt] (see page 626)	Loads a CAN DBC (communication database) symbolic data file into the oscilloscope.
:SBUS<n>:CAN:DISPlay (see page 687)	Specifies whether symbolic values (from a loaded DBC file) or hexadecimal values are displayed in the decode waveform and the Lister window
:SBUS<n>:CAN:TRIGger:SYMBOLic:MESSage (see page 699)	Specifies the message to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MESSage or MSIGnal.
:SBUS<n>:CAN:TRIGger:SYMBOLic:SIGNal (see page 700)	Specifies signal to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.
:SBUS<n>:CAN:TRIGger:SYMBOLic:VALue (see page 701)	Specifies signal value to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.
:SEARch:EVENT (see page 838)	Lets you navigate to one of the found search events.
:SEARch:SERial:CAN:SYMBOLic:MESSage (see page 874)	Specifies the message to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MESSage or MSIGnal.

Command	Description
:SEARch:SERial:CAN:SYMBolic:SIGNal (see page 875)	Specifies signal to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.
:SEARch:SERial:CAN:SYMBolic:VALue (see page 876)	Specifies signal value to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

Changed Commands

Command	Differences
:SBUS<n>:CAN:TRIGger (see page 692)	You can now trigger on symbolic messages (with parameter MESSage) or messages and signal values (with parameter MSIGnal).
:SBUS<n>:LIN:TRIGger (see page 758)	You can now trigger on parity errors (with parameter PARityerror) or checksum errors (with parameter CSUMerror).
:SEARch:SERial:CAN:MODE (see page 868)	You can now search for symbolic messages (with parameter MESSage) or messages and signal values (with parameter MSIGnal).

What's New in Version 3.10

New features in version 3.10 of the InfiniiVision 4000 X-Series oscilloscope software are:

- The DSOX4USBSQ USB 2.0 signal quality analysis application.
- Support for the N2820A high-sensitivity current probe.
- Saving Multi Channel Waveform data (*.h5) format files that can be opened by the N8900A InfiniiView oscilloscope analysis software.

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

New Commands

Command	Description
:COMPliance Commands (see page 287)	Commands for controlling the optional DSOX4USBSQ USB 2.0 signal quality analysis feature.
:MEASure:DUAL:CHARge (see page 441)	These are special measurements available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used.
:MEASure:DUAL:VAMplitude (see page 442)	These measurements join the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.
:MEASure:DUAL:VAverage (see page 443)	
:MEASure:DUAL:VBASe (see page 444)	
:MEASure:DUAL:VPP (see page 445)	
:MEASure:DUAL:VRMS (see page 446)	
:SAVE:COMPliance:USB[:STA Rt] (see page 637)	Lets you save DSOX4USBSQ USB 2.0 signal quality test results.
:SAVE:MULTi[:STARt] (see page 646)	Lets you save Multi Channel Waveform data (*.h5) format files that can be opened by the N8900A InfiniiView oscilloscope analysis software.
:TRIGger:LEVel:ASETup (see page 965)	Sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

What's New in Version 3.01

New features in version 3.01 of the InfiniiVision 4000 X-Series oscilloscope software are:

- Ability to turn reference waveform locations on or off and view their status using the :VIEW, :BLANK, and :STATus commands.

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

Changed Commands

Command	Differences
:BLANK (see page 202)	You can now use the WMEMOrY<r> source parameter to turn off the display of a reference waveform location.
:STATus (see page 227)	You can now use the WMEMOrY<r> source parameter to view the display status of a reference waveform location.
:VIEW (see page 230)	You can now use the WMEMOrY<r> source parameter to turn on the display of a reference waveform location.

Version 3.00 at Introduction

The Agilent InfiniiVision 4000 X-Series oscilloscopes were introduced with version 3.00 of oscilloscope operating software.

The command set is most closely related to the InfiniiVision 3000 X-Series oscilloscopes (and the 7000A/B Series, 6000 Series, and 54620/54640 Series oscilloscopes before them). For more information, see ["Command Differences From 3000 X-Series Oscilloscopes" on page 39](#).

Command Differences From 3000 X-Series Oscilloscopes

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

The main differences between the version 3.00 programming command set for the InfiniiVision 4000 X-Series oscilloscopes and the 2.10 programming command set for the InfiniiVision 3000 X-Series oscilloscopes are related to:

- Zone qualified trigger feature.
- USB 2.0 triggering and serial decode options (which replace the old :TRIGger:USB commands).
- Two-channel built-in waveform generator (with Option WGN license) instead of one channel.
- Support for modulation of the waveform generator output on WaveGen1.
- A couple new built-in demo signals (with Option EDU license that comes with the N6455A Education Kit).
- Four reference waveforms (instead of two).
- Four math functions (instead of g(t) and f(t)).
- Touchscreen, sidebar, and display related command additions.
- Measure cursors mode.
- Support for controlling the optional DSOXDVM digital voltmeter analysis feature.
- Power measurements application modifications.

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

New Commands

Command	Description
:ACQuire:RSIGnal (see page 237)	Lets you select the 10 MHz reference signal mode.
:DISPlay:MENU (see page 325)	Changes the front panel softkey menu.
:DISPlay:SIDebar (see page 326)	Specifies the sidebar dialog to display on the screen.
:DVM Commands (see page 329)	Commands for controlling the optional DSOXDVM digital voltmeter analysis feature.
:FUNCtion<m>:AVERage:COU Nt (see page 347)	Specifies the count for the Averaged Value math function.

Command	Description
:MEASure:CPLoss (see page 501)	Installs a power loss per cycle measurement on screen or returns the measured value.
:POWer:SIGNals:CYCles:HARMonics (see page 592)	Specifies the number of cycles to include in the current harmonics analysis.
:POWer:SIGNals:CYCles:QUALity (see page 593)	Specifies the number of cycles to include in the power quality analysis.
:POWer:SIGNals:DURation:EFFiciency (see page 594)	Specifies the duration of the efficiency analysis.
:POWer:SIGNals:DURation:MODulation (see page 595)	Specifies the duration of the modulation analysis.
:POWer:SIGNals:DURation:ONOFF:OFF (see page 596)	Specifies the duration of the turn off analysis.
:POWer:SIGNals:DURation:ONOFF:ON (see page 597)	Specifies the duration of the turn on analysis.
:POWer:SIGNals:DURation:RIPPLE (see page 598)	Specifies the duration of the output ripple analysis.
:POWer:SIGNals:DURation:TRANSient (see page 599)	Specifies the duration of the transient response analysis.
:POWer:SIGNals:VMAXimum:INRush (see page 602)	Specifies the maximum expected input voltage for inrush current analysis.
:POWer:SIGNals:VMAXimum:ONOFF:OFF (see page 603)	Specifies the maximum expected input voltage for turn off analysis.
:POWer:SIGNals:VMAXimum:ONOFF:ON (see page 604)	Specifies the maximum expected input voltage for turn on analysis.
:POWer:SIGNals:VSTeady:ONOFF:OFF (see page 605)	Specifies the expected steady state output DC voltage of the power supply for turn off analysis.
:POWer:SIGNals:VSTeady:ONOFF:ON (see page 606)	Specifies the expected steady state output DC voltage of the power supply for turn on analysis.
:POWer:SIGNals:VSTeady:TRANSIENT (see page 607)	Specifies the expected steady state output DC voltage of the power supply for transient response analysis.
:SBUS<n>:USB Commands (see page 809)	Commands for using the USB 2.0 triggering and serial decode feature.
:SEARch:SERial:USB Commands (see page 914)	Commands for searching USB 2.0 serial decode events.
:SYSTem:TOUCH (see page 946)	Disables or enables the touchscreen.
:TIMEbase:REFClock (see page 952)	Lets you enable or disable the use of a 10 MHz reference input signal.

Command	Description
:TRIGger:ZONE Commands (see page 1036)	Commands for setting up zone qualified triggers.
:WGEN<w>:OUTPut:POLarity (see page 1111)	Lets you invert the waveform generator output.
:WGEN<w>:MODulation:AM: DEPTh (see page 1097)	Specifies the amount of amplitude modulation.
:WGEN<w>:MODulation:AM: FREQuency (see page 1098)	Specifies the frequency of the modulating signal.
:WGEN<w>:MODulation:FM: DEViation (see page 1099)	Specifies the frequency deviation from the original carrier signal frequency.
:WGEN<w>:MODulation:FM:F REQuency (see page 1100)	Specifies the frequency of the modulating signal.
:WGEN<w>:MODulation:FSKe y:FREQuency (see page 1101)	Specifies the "hop frequency".
:WGEN<w>:MODulation:FSKe y:RATE (see page 1102)	Specifies the rate at which the output frequency "shifts".
:WGEN<w>:MODulation:FUN Ction (see page 1103)	Specifies the shape of the modulating signal.
:WGEN<w>:MODulation:FUN Ction:RAMP:SYMMetry (see page 1104)	Specifies the amount of time per cycle that the ramp waveform is rising.
:WGEN<w>:MODulation:STAT e (see page 1106)	Enables or disables modulated waveform generator output.
:WGEN<w>:MODulation:TYP E (see page 1107)	Selects the modulation type: Amplitude Modulation (AM), Frequency Modulation (FM), or Frequency-Shift Keying Modulation (FSK).
:WGEN<w>:TRACk (see page 1114)	Causes frequency, amplitude, offset, and duty cycle adjustments to this waveform generator output signal to be tracked by the other waveform generator output.
:WGEN<w>:TRACk:AMPLitud e (see page 1115)	Causes amplitude and offset adjustments to this waveform generator output signal to be tracked by the other waveform generator output.
:WGEN<w>:TRACk:CSIGnal (see page 1116)	Copies the signal settings from one waveform generator to the other.
:WGEN<w>:TRACk:FREQuenc y (see page 1117)	Causes frequency adjustments to this waveform generator output signal to be tracked by the other waveform generator output.

Changed Commands

Command	Differences From InfiniiVision 3000 X-Series Oscilloscopes
:ACQuire:MODE (see page 235)	The ETIMe mode is available with the 1 GHz or 1.5 GHz bandwidth models.
:CALibrate:OUTPut (see page 261)	The TRIG OUT signal can be a trigger output, mask test failure, or waveform generator sync pulse from either WaveGen1 or WaveGen2.
:DEMO:FUNCTION (see page 300)	The USB and NMONotonic functions are now available with the DSOXEDK educator's kit license.
:DISPlay:VECTors (see page 328)	The OFF parameter is available with the 1 GHz or 1.5 GHz bandwidth models.
:FUNCTION<m>:OPERation (see page 364)	The AVERage parameter is available for the Averaged Value math function.
:MARKer:MODE (see page 401)	You can now select the BINary or HEX marker modes.
:MEASure:SHOW (see page 467)	This now has an OFF state when markers are not in measurement mode.
:MEASURE:WINDOW (see page 491)	This now has a GATE option for gated measurements between the X1 and X2 cursors.
:RECall:ARbitrary:[STARt] (see page 625)	There is now an option to specify which waveform generator to recall the arbitrary waveform into.
:SAVE:ARbitrary:[STARt] (see page 636)	There is now an option to specify which waveform generator to save the arbitrary waveform from.
:SAVE:WAVeform:FORMAT (see page 651)	The ALB Agilent module binary format is not supported by the 4000 X-Series oscilloscopes.
:SBUS<n>:MODE (see page 661)	The USB mode is now available with the DSOX4USBFL (USB 2.0 Full/Low Speed) or DSOX4USBH (USB 2.0 High Speed) serial decode and triggering licenses.
:TRIGger[:EDGE]:SOURce (see page 989)	You can now trigger on the WGEN1 or WGEN2 outputs or, when FM or FSK modulation is used, you can trigger on the WMOD modulating signal output.
:TRIGger:MODE (see page 968)	The USB mode has been superseded by commands that are available with the DSOX4USBFL (USB 2.0 Full/Low Speed) or DSOX4USBH (USB 2.0 High Speed) serial decode and triggering licenses. See " :SBUS<n>:USB Commands " on page 809.
:WGEN<w> Commands (see page 1079)	With the built-in waveform generator's (with Option WGN license) two outputs, you can use :WGEN1 and :WGEN2 to specify which output you are controlling. :WGEN is the same as :WGEN1. Maximum waveform generator amplitudes and offsets are also different.
:WMEMory<r> Commands (see page 1123)	<r> is now from 1 to 4 with four reference waveform memories.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:FUNCtion Commands	:FUNCtion2 Commands (see page 343)	:FUNCtion commands (with no <m> number) map to :FUNCtion2. This allows legacy programs to work without change.
:FUNCtion:GOFT:OPERation (see page 1150)	:FUNCtion1:OPERation (see page 364)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce1 (see page 1151)	:FUNCtion1:SOURce1 (see page 370)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce2 (see page 1152)	:FUNCtion1:SOURce2 (see page 372)	GOFT maps to FUNCtion1.
:SYSTem:MENU (see page 1187)	:DISPlay:MENU (see page 325)	No change in behavior.

Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:POWer:SIGNals:CYCLes	:POWer:SIGNals:CYCLes:HARMonics (see page 592) :POWer:SIGNals:CYCLes:QUALity (see page 593)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:DURation	:POWer:SIGNals:DURation:EFFiciency (see page 594) :POWer:SIGNals:DURation:MODulation (see page 595) :POWer:SIGNals:DURation:ONOFF:OFF (see page 596) :POWer:SIGNals:DURation:ONOFF:ON (see page 597) :POWer:SIGNals:DURation:RIPPLE (see page 598) :POWer:SIGNals:DURation:TRANsient (see page 599)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:VMAXimum	:POWer:SIGNals:VMAXimum:INRush (see page 602) :POWer:SIGNals:VMAXimum:ONOFF:OFF (see page 603) :POWer:SIGNals:VMAXimum:ONOFF:ON (see page 604)	This command was separated into several other commands for specific types of power analysis.

Discontinued Command	Current Command Equivalent	Comments
:POWer:SIGNals:VSTeady	:POWer:SIGNals:VSTeady:ON OFF:OFF (see page 605) :POWer:SIGNals:VSTeady:ON OFF:ON (see page 606) :POWer:SIGNals:VSTeady:TRA Nsient (see page 607)	This command was separated into several other commands for specific types of power analysis.
:POWer:SLEW:VALUe	none	Slew rate values are now displayed using max and min measurements of a differentiate math function signal.
:PWRenable	none	The Power Event Enable Register does not exist in the 4000 X-Series oscilloscopes.
:PWRRegister[:EVENT]	none	The Power Event Event Register does not exist in the 4000 X-Series oscilloscopes.
:TRIGger:USB:SOURce:DMINu s	none	These USB trigger commands are superseded by commands that are available with the DSOX4USBFL (USB 2.0 Full/Low Speed) or DSOX4USBH (USB 2.0 High Speed) serial decode and triggering licenses. See " :SBUS<n>:USB Commands " on page 809.
:TRIGger:USB:DPLus	none	
:TRIGger:USB:SPEed	none	
:TRIGger:USB:TRIGger	none	

2 **Setting Up**

Step 1. Install Agilent IO Libraries Suite software [46](#)

Step 2. Connect and set up the oscilloscope [47](#)

Step 3. Verify the oscilloscope connection [49](#)

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 4000 X-Series oscilloscope has two different interfaces you can use for programming:

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

These interfaces are always active.

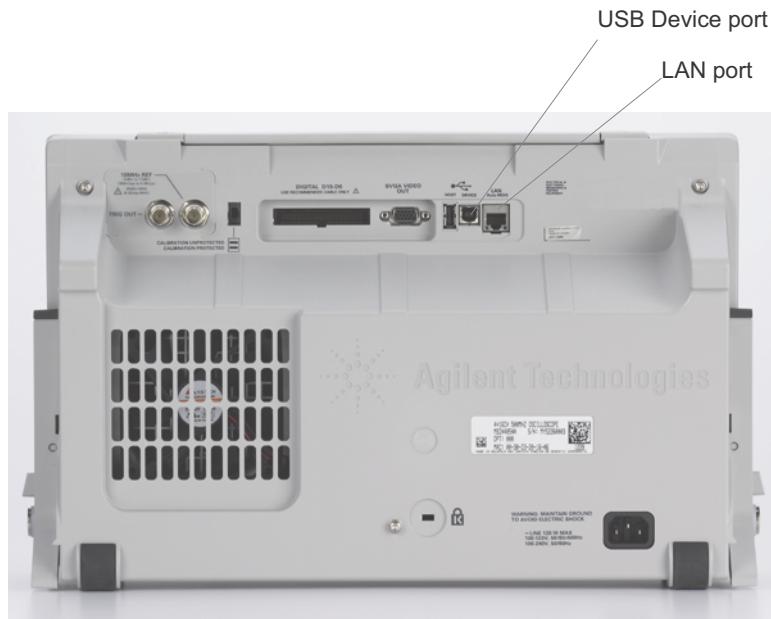


Figure 1 Control Connectors on Rear Panel

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 back of the oscilloscope.
- 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 **Hostname** softkey. Use the softkeys and the Entry knob to enter the Host name.

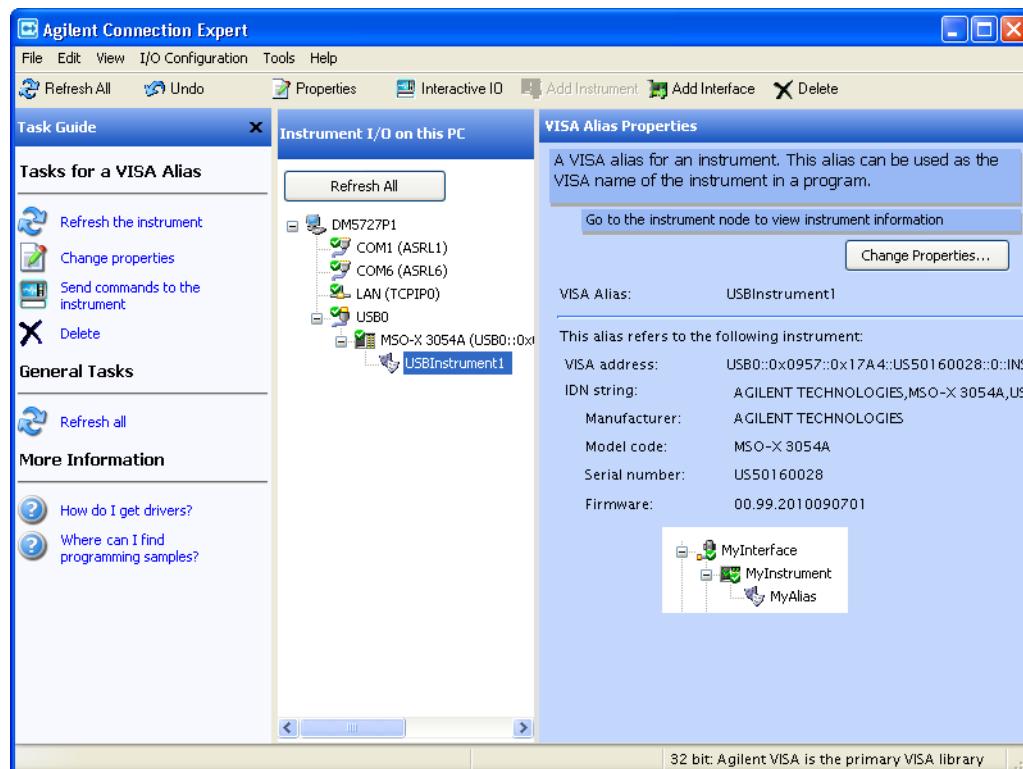
When you are done, press the **[Back up]** key.

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.



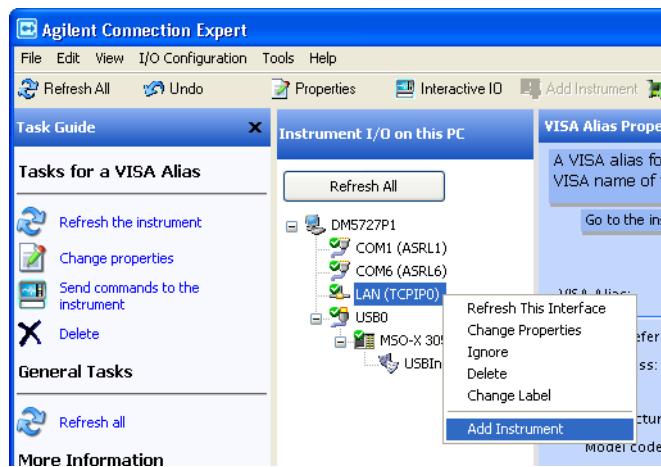
- 2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



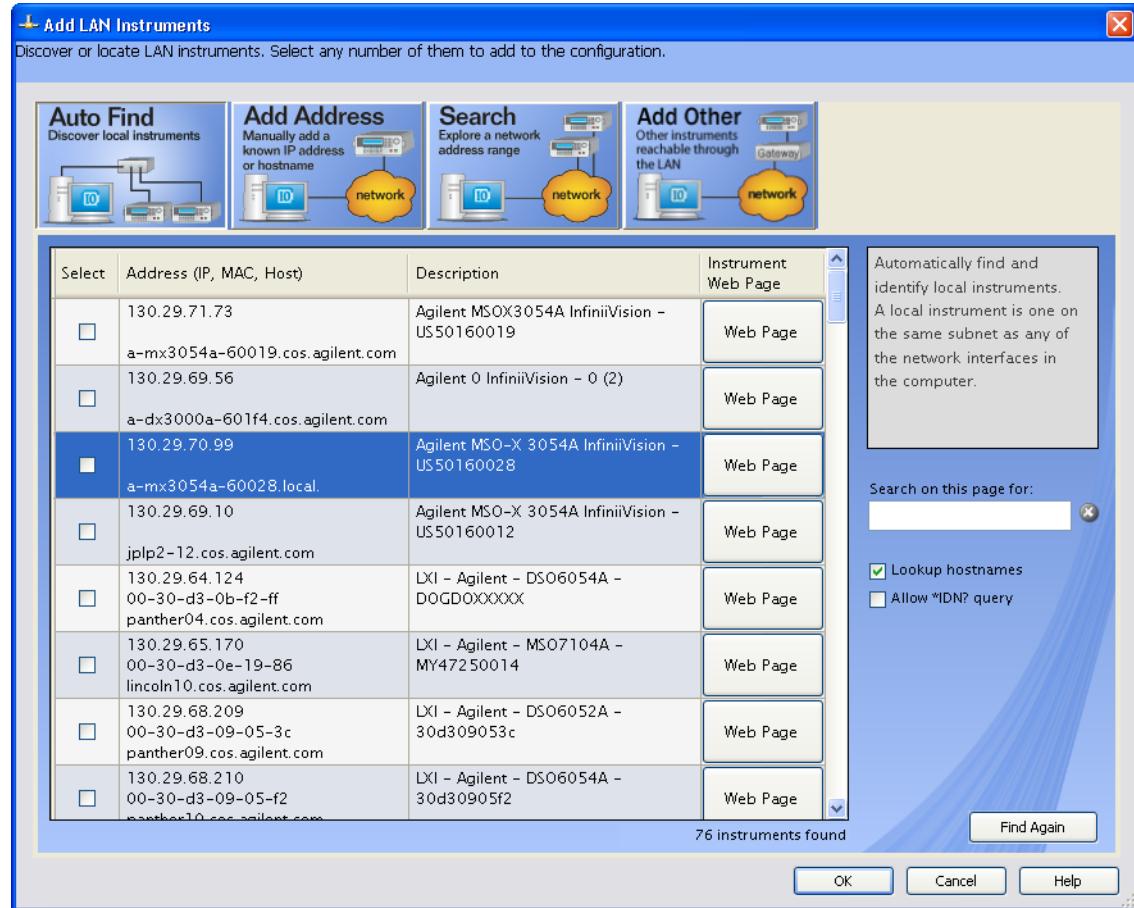
2 Setting Up

You must manually add instruments on LAN interfaces:

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



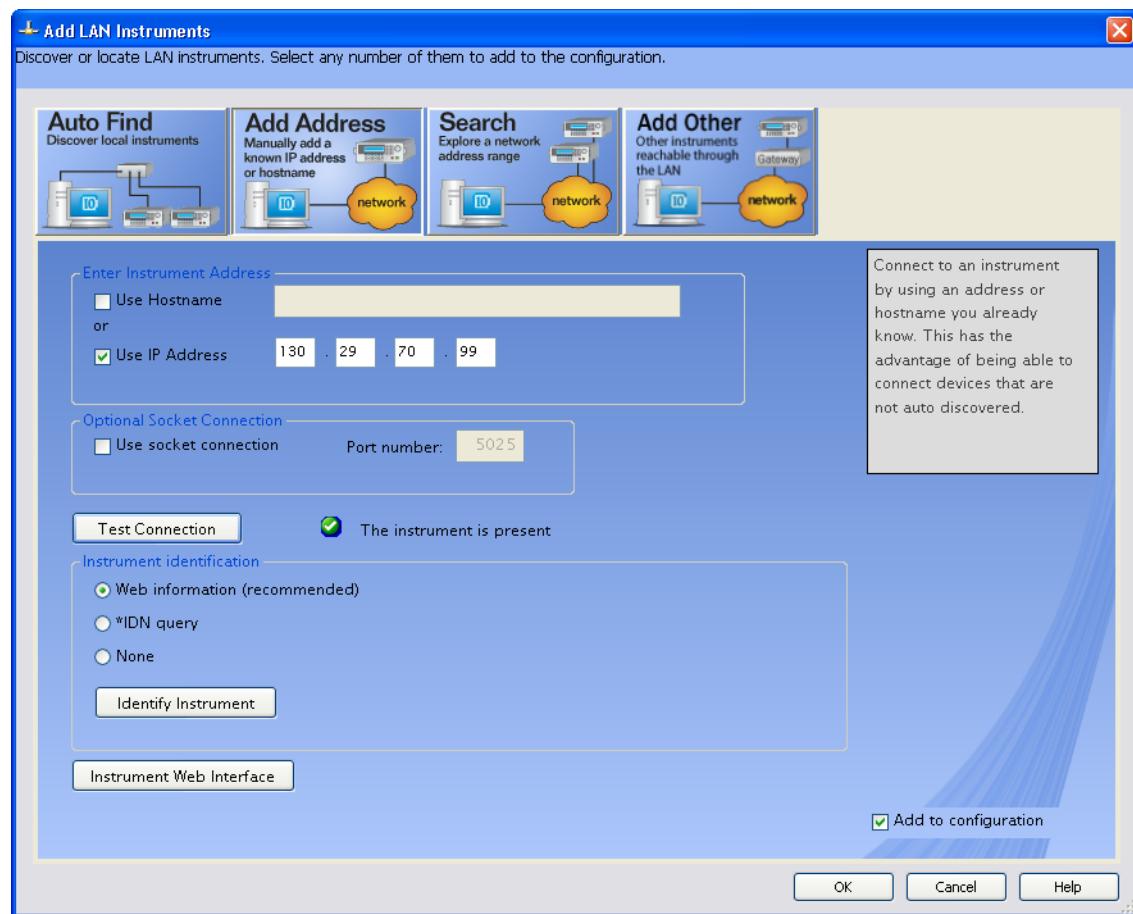
- b If the oscilloscope is on the same subnet, select it, and click **OK**.



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**.

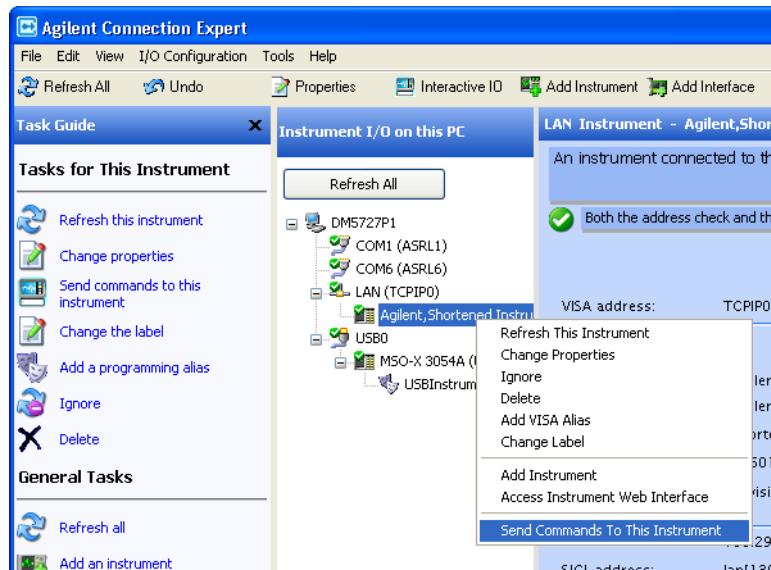
2 Setting Up



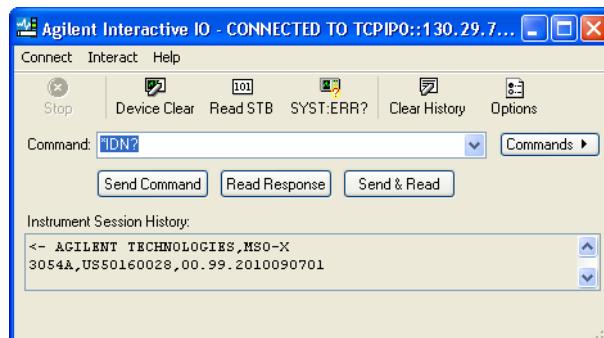
- iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

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.

2 Setting Up

3 Getting Started

Basic Oscilloscope Program Structure 56

Programming the Oscilloscope 58

Other Ways of Sending Commands 67

This chapter gives you an overview of programming the 4000 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

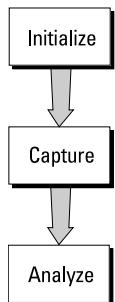
Language for Program Examples

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" on page 58
- "Opening the Oscilloscope Connection via the IO Library" on page 59
- "Using :AUToscale to Automate Oscilloscope Setup" on page 60
- "Using Other Oscilloscope Setup Commands" on page 60
- "Capturing Data with the :DIGitize Command" on page 61
- "Reading Query Responses from the Oscilloscope" on page 63
- "Reading Query Results into String Variables" on page 64
- "Reading Query Results into Numeric Variables" on page 64
- "Reading Definite-Length Block Query Response Data" on page 64
- "Sending Multiple Queries and Reading Results" on page 65
- "Checking Instrument Status" on page 66

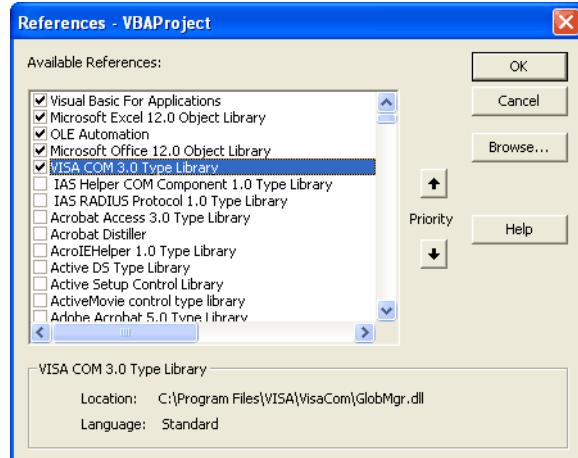
Referencing the IO Library

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 1233.

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 clear 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 165.

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 μ s.

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 33](#), “:WAVeform Commands,” starting on page 1043.

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:

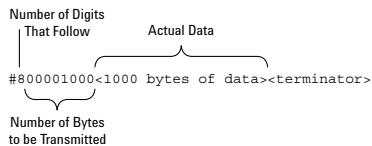


Figure 2 Definite-length block response data

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 38](#), “Status Reporting,” starting on page 1199 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" on page 67
- "Sending SCPI Commands Using Browser Web Control" on page 67

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 4000 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.

3 Getting Started

4

Commands Quick Reference

Command Summary [70](#)

Syntax Elements [161](#)



Command Summary

- Common (*) Commands Summary (see [page 72](#))
- Root (:) Commands Summary (see [page 75](#))
- :ACQuire Commands Summary (see [page 78](#))
- :BUS<n> Commands Summary (see [page 79](#))
- :CALibrate Commands Summary (see [page 80](#))
- :CHANnel<n> Commands Summary (see [page 81](#))
- :DEMO Commands Summary (see [page 83](#))
- :DIGItal<n> Commands Summary (see [page 83](#))
- :DISPlay Commands Summary (see [page 84](#))
- :DVM Commands Summary (see [page 85](#))
- :EXTernal Trigger Commands Summary (see [page 86](#))
- :FUNCtion Commands Summary (see [page 86](#))
- :HARDcopy Commands Summary (see [page 89](#))
- :LISTer Commands Summary (see [page 90](#))
- :MARKer Commands Summary (see [page 91](#))
- :MEASure Commands Summary (see [page 92](#))
- :MTEST Commands Summary (see [page 106](#))
- :POD<n> Commands Summary (see [page 109](#))
- :POWER Commands Summary (see [page 109](#))
- :RECall Commands Summary (see [page 114](#))
- :SAVE Commands Summary (see [page 115](#))
- General :SBUS<n> Commands Summary (see [page 117](#))
- :SBUS<n>:A429 Commands Summary (see [page 117](#))
- :SBUS<n>:CAN Commands Summary (see [page 119](#))
- :SBUS<n>:FLEXray Commands Summary (see [page 120](#))
- :SBUS<n>:I2S Commands Summary (see [page 122](#))
- :SBUS<n>:IIC Commands Summary (see [page 124](#))
- :SBUS<n>:LIN Commands Summary (see [page 125](#))
- :SBUS<n>:M1553 Commands Summary (see [page 126](#))
- :SBUS<n>:SPI Commands Summary (see [page 127](#))
- :SBUS<n>:UART Commands Summary (see [page 129](#))
- :SBUS<n>:USB Commands Summary (see [page 131](#))
- General :SEARch Commands Summary (see [page 133](#))

- :SEARch:EDGE Commands Summary (see [page 133](#))
- :SEARch:GLITch Commands Summary (see [page 134](#))
- :SEARch:RUNT Commands Summary (see [page 134](#))
- :SEARch:TRANSition Commands Summary (see [page 135](#))
- :SEARch:SERial:A429 Commands Summary (see [page 135](#))
- :SEARch:SERial:CAN Commands Summary (see [page 136](#))
- :SEARch:SERial:FLEXray Commands Summary (see [page 137](#))
- :SEARch:SERial:I2S Commands Summary (see [page 137](#))
- :SEARch:SERial:IIC Commands Summary (see [page 138](#))
- :SEARch:SERial:LIN Commands Summary (see [page 139](#))
- :SEARch:SERial:M1553 Commands Summary (see [page 139](#))
- :SEARch:SERial:SPI Commands Summary (see [page 140](#))
- :SEARch:SERial:UART Commands Summary (see [page 141](#))
- :SEARch:SERial:UART Commands Summary (see [page 141](#))
- :SYSTem Commands Summary (see [page 143](#))
- :TIMEbase Commands Summary (see [page 143](#))
- General :TRIGger Commands Summary (see [page 144](#))
- :TRIGger:DELay Commands Summary (see [page 145](#))
- :TRIGger:EBURst Commands Summary (see [page 146](#))
- :TRIGger[:EDGE] Commands Summary (see [page 147](#))
- :TRIGger:GLITch Commands Summary (see [page 148](#))
- :TRIGger:OR Commands Summary (see [page 149](#))
- :TRIGger:PATTern Commands Summary (see [page 150](#))
- :TRIGger:RUNT Commands Summary (see [page 151](#))
- :TRIGger:SHOLD Commands Summary (see [page 151](#))
- :TRIGger:TRANSition Commands Summary (see [page 152](#))
- :TRIGger:TV Commands Summary (see [page 152](#))
- :TRIGger:ZONE Commands Summary (see [page 153](#))
- :WAVEform Commands Summary (see [page 154](#))
- :WGEN Commands Summary (see [page 156](#))
- :WMEMory<r> Commands Summary (see [page 160](#))

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 169)	n/a	n/a
*ESE <mask> (see page 170)	*ESE? (see page 170)	<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 172)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 172)	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 175)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 176)	*OPC? (see page 176)	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 177)	<pre> <return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <Xilinx FPGA Probe>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <FlexRay Serial>, <Power Measurements>, <RS-232/UART Serial>, <reserved>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <I2S Serial>, <reserved>, <Educator's Kit>, <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>, <reserved>, <reserved>, <reserved>, <reserved>, <Digital Voltmeter>, <Spectrum Visualizer>, <reserved>, <reserved>, <USB 2.0 Low/Full Speed>, <USB 2.0 High Speed> <All field> ::= {0 All} <reserved> ::= 0 <MSO> ::= {0 MSO} <Xilinx FPGA Probe> ::= {0 FPGAX} <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} <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} <Digital Voltmeter> ::= {0 DVM} <Spectrum Visualizer> ::= {0 ASV} <USB 2.0 Low/Full Speed> ::= {0 USF} <USB 2.0 High Speed> ::= {0 HUF} </pre>

4 Commands Quick Reference

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																													
*RCL <value> (see page 179)	n/a	<value> ::= { 0 1 4 5 6 7 8 9 }																																													
*RST (see page 180)	n/a	See *RST (Reset) (see page 180)																																													
*SAV <value> (see page 183)	n/a	<value> ::= { 0 1 4 5 6 7 8 9 }																																													
*SRE <mask> (see page 184)	*SRE? (see page 185)	<p><mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>Enables</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>Operation Status Reg</td> </tr> <tr> <td>6</td> <td>64</td> <td>---</td> <td>(Not used.)</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td>Event Status Bit</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td>Message Available</td> </tr> <tr> <td>3</td> <td>8</td> <td>---</td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td>Message</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td>User</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td>Trigger</td> </tr> </tbody> </table>	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									
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 186)	<p><value> ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>"1"</th> <th>Indicates</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>---</td> <td>Operation status condition occurred.</td> </tr> <tr> <td>6</td> <td>64</td> <td>RQS/</td> <td>MSS</td> <td>Instrument is requesting service.</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td></td> <td>Enabled event status condition occurred.</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td></td> <td>Message available.</td> </tr> <tr> <td>3</td> <td>8</td> <td>---</td> <td></td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td></td> <td>Message displayed.</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td></td> <td>User event condition occurred.</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td></td> <td>A trigger occurred.</td> </tr> </tbody> </table>	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.
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 188)	n/a	n/a																																													
n/a	*TST? (see page 189)	<result> ::= 0 or non-zero value; an integer in NR1 format																																													
*WAI (see page 190)	n/a	n/a																																													

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 195)	:ACTivity? (see page 195)	<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 196)	{0 1}; an integer in NR1 format
:AUToscale [<source>[,...<source>]] (see page 197)	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 199)	:AUToscale:AMODE? (see page 199)	<value> ::= {NORMAl CURRent}
:AUToscale:CHANnels <value> (see page 200)	:AUToscale:CHANnels? (see page 200)	<value> ::= {ALL DISPlayed}
:AUToscale:FDEBug {{0 OFF} {1 ON}} (see page 201)	:AUToscale:FDEBug? (see page 201)	{0 1}
:BLANK [<source>] (see page 202)	n/a	<source> ::= {CHANnel<n>} FUNCtion<m> MATH<m> SBUS{1 2} WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCtion<m> MATH<m> SBUS{1 2} WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGItize [<source/> [,...,source>]] (see page 203)	n/a	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> SBUS{1 2}} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCtion<m> MATH<m> SBUS{1 2}} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p>
:HWEenable <n> (see page 205)	:HWEenable? (see page 205)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDiti on? (see page 207)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister[:EVENT]? (see page 208)	<n> ::= 16-bit integer in NR1 format
:MTEenable <n> (see page 209)	:MTEenable? (see page 209)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENT]? (see page 211)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 213)	:OPEE? (see page 214)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDiti on? (see page 215)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister[:EVENT]? (see page 217)	<n> ::= 15-bit integer in NR1 format

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:OVLenable <mask> (see page 219)	:OVLenable? (see page 220)	<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 221)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINT [<options>] (see page 223)	n/a	<options> ::= [<print option>] [,...,<print option>] <print option> ::= {COLOR GRAYscale PRINTER0 BMP8bit BMP PNG NOFactors FACTors} <print option> can be repeated up to 5 times.
:RUN (see page 224)	n/a	n/a
n/a	:SERial (see page 225)	<return value> ::= unquoted string containing serial number
:SINGle (see page 226)	n/a	n/a
n/a	:STATus? <display> (see page 227)	{0 1} <display> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCTion<m> MATH<m> SBUS{1 2} WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 228)	n/a	n/a

4 Commands Quick Reference

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 229)	{0 1}
:VIEW <source> (see page 230)	n/a	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> SBUS{1 2} WMEMORY<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCtion<m> MATH<m> SBUS{1 2} WMEMORY<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p>

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMplete <complete> (see page 233)	:ACQuire:COMplete? (see page 233)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see page 234)	:ACQuire:COUNT? (see page 234)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 235)	:ACQuire:MODE? (see page 235)	<mode> ::= {RTIMe ETIMe SEGmented}
n/a	:ACQuire:POINTS? (see page 236)	<# points> ::= an integer in NR1 format
:ACQuire:RSIGnal <ref_signal_mode> (see page 237)	:ACQuire:RSIGnal? (see page 237)	<ref_signal_mode> ::= {OFF OUT IN}
:ACQuire:SEGmented:AN ALyze (see page 238)	n/a	n/a (with Option SGM)
:ACQuire:SEGmented:CO UNT <count> (see page 239)	:ACQuire:SEGmented:CO UNT? (see page 239)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
:ACQuire:SEGmented:IN Dex <index> (see page 240)	:ACQuire:SEGmented:IN Dex? (see page 240)	<index> ::= an integer from 1 to 1000 in NR1 format (with Option SGM)

Table 4 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:ACQuire:SRATE? (see page 243)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 244)	:ACQuire:TYPE? (see page 244)	<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 249)	:BUS<n>:BIT<m>? (see page 249)	{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 250)	:BUS<n>:BITS? (see page 250)	<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 252)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 253)	:BUS<n>:DISPlay? (see page 253)	{0 1} <n> ::= 1 or 2; an integer in NR1 format

4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 254)	:BUS<n>:LABel? (see page 254)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 255)	:BUS<n>:MASK? (see page 255)	<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 259)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABEL <string> (see page 260)	:CALibrate:LABEL? (see page 260)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 261)	:CALibrate:OUTPut? (see page 261)	<signal> ::= {TRIGgers MASK WAVEgen WGEN1 WGEN2} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.
n/a	:CALibrate:PROTected? (see page 262)	{PROTected UNPROTected}
:CALibrate:START (see page 263)	n/a	n/a

Table 6 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:STATUS? (see page 264)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see page 265)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 266)	<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 270)	:CHANnel<n>:BWLimit? (see page 270)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 271)	:CHANnel<n>:COUPling? (see page 271)	<coupling> ::= {AC DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay { {0 OFF} {1 ON} } (see page 272)	:CHANnel<n>:DISPlay? (see page 272)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 273)	:CHANnel<n>:IMPedance? (see page 273)	<impedance> ::= {ONEMeg FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert { {0 OFF} {1 ON} } (see page 274)	:CHANnel<n>:INVert? (see page 274)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see page 275)	:CHANnel<n>:LABEL? (see page 275)	<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 276)	:CHANnel<n>:OFFSet? (see page 276)	<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 277)	:CHANnel<n>:PROBe? (see page 277)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:PROBe:HEA D [:TYPE] <head_param> (see page 278)	:CHANnel<n>:PROBe:HEA D [:TYPE]? (see page 278)	<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 279)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKE W <skew_value> (see page 280)	:CHANnel<n>:PROBe:SKE W? (see page 280)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STY Pe <signal type> (see page 281)	:CHANnel<n>:PROBe:STY Pe? (see page 281)	<signal type> ::= {DIFFerential SINGLE} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see page 282)	:CHANnel<n>:PROTection? (see page 282)	{NORM TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 283)	:CHANnel<n>:RANGE? (see page 283)	<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 284)	:CHANnel<n>:SCALe? (see page 284)	<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 285)	:CHANnel<n>:UNITS? (see page 285)	<units> ::= {VOLT AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier { {0 OFF} {1 ON} } (see page 286)	:CHANnel<n>:VERNier? (see page 286)	{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 300)	:DEMO:FUNCTION? (see page 303)	<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 USB NMONotonic}
:DEMO:FUNCTION:PHASE: PHASE <angle> (see page 304)	:DEMO:FUNCTION:PHASE: PHASE? (see page 304)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 OFF} {1 ON}} (see page 305)	:DEMO:OUTPut? (see page 305)	{0 1}

Table 9 :DIGItal<d> Commands Summary

Command	Query	Options and Query Returns
:DIGItal<d>:DISPlay {{0 OFF} {1 ON}} (see page 309)	:DIGItal<d>:DISPlay? (see page 309)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 1}
:DIGItal<d>:LABel <string> (see page 310)	:DIGItal<d>:LABel? (see page 310)	<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 311)	:DIGItal<d>:POSITION? (see page 311)	<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 312)	:DIGItal<d>:SIZE? (see page 312)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALL MEDIUM LARGE}
:DIGItal<d>:THreshold <value>[suffix] (see page 313)	:DIGItal<d>:THreshold? ? (see page 313)	<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 317)	:DISPlay:ANNotation? (see page 317)	{0 1}
:DISPlay:ANNotation:B ACKground <mode> (see page 318)	:DISPlay:ANNotation:B ACKground? (see page 318)	<mode> ::= {OPAQUE INVERTed TRANSPARENT}
:DISPlay:ANNotation:C OLor <color> (see page 319)	:DISPlay:ANNotation:C OLor? (see page 319)	<color> ::= {CH1 CH2 CH3 CH4 DIG MATH REF MARKer WHITE RED}
:DISPlay:ANNotation:T EXT <string> (see page 320)	:DISPlay:ANNotation:T EXT? (see page 320)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 321)	n/a	n/a
n/a	:DISPlay:DATA? [<format>] [,] [<palett e>] (see page 322)	<format> ::= {BMP BMP8bit PNG} <palette> ::= {COLor GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel { {0 OFF} {1 ON} } (see page 323)	:DISPlay:LABel? (see page 323)	{0 1}
:DISPlay:LABList <binary block> (see page 324)	:DISPlay:LABList? (see page 324)	<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:MENU <menu> (see page 325)	n/a	<menu> ::= {MASK MEASure SEGmented LISTer POWER}
:DISPlay:SIDebar <sidebar> (see page 326)	n/a	<sidebar> ::= {SUMmary CURSors MEASurements DVM NAVigate CONTrols}
:DISPlay:PERSistence <value> (see page 327)	:DISPlay:PERSistence? (see page 327)	<value> ::= {MINimum INFinite <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {{0 OFF} {1 ON}} (see page 328)	:DISPlay:VECTors? (see page 328)	{0 1} OFF or 0 can only be selected with the 1 GHz or 1.5 GHz bandwidth models

Table 11 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARAnge {{0 OFF} {1 ON}} (see page 330)	:DVM:ARAnge? (see page 330)	{0 1}
n/a	:DVM:CURREnt? (see page 331)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABLE {{0 OFF} {1 ON}} (see page 332)	:DVM:ENABLE? (see page 332)	{0 1}
n/a	:DVM:FREQuency? (see page 331)	<freq_value> ::= floating-point number in NR3 format
:DVM:MODE <mode> (see page 334)	:DVM:MODE? (see page 334)	<dvm_mode> ::= {ACRMs DC DCRMs FREQuency}
:DVM:SOURce <source> (see page 335)	:DVM:SOURce? (see page 335)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format

Table 12 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLImIt <bwlImIt> (see page 338)	:EXTernal:BWLImIt? (see page 338)	<bwlImIt> ::= {0 OFF}
:EXTernal:PROBe <attenuation> (see page 339)	:EXTernal:PROBe? (see page 339)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGE <range>[<suffix>] (see page 340)	:EXTernal:RANGE? (see page 340)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXTernal:UNITS <units> (see page 341)	:EXTernal:UNITS? (see page 341)	<units> ::= {VOLT AMPere}

Table 13 :FUNCtion<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCtion<m>:AVERage: COUNT <count> (see page 347)	:FUNCtion<m>:AVERage: COUNT? (see page 347)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1-4 in NR1 format
:FUNCtion<m>:BUS:CLOCk <source> (see page 348)	:FUNCtion<m>:BUS:CLOCk? (see page 348)	<source> ::= {CHANnel<n> DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format
:FUNCtion<m>:BUS:SLOPe <slope> (see page 349)	:FUNCtion<m>:BUS:SLOPe? (see page 349)	<slope> ::= {NEGative POSitive EITHer} <m> ::= 1-4 in NR1 format
:FUNCtion<m>:BUS:YINC rement <value> (see page 350)	:FUNCtion<m>:BUS:YINC rement? (see page 350)	<value> ::= value per bus code, in NR3 format <m> ::= 1-4 in NR1 format
:FUNCtion<m>:BUS:YORigin <value> (see page 351)	:FUNCtion<m>:BUS:YORigin? (see page 351)	<value> ::= value at bus code = 0, in NR3 format <m> ::= 1-4 in NR1 format
:FUNCtion<m>:BUS:YUNI ts <units> (see page 352)	:FUNCtion<m>:BUS:YUNI ts? (see page 352)	<units> ::= {VOLT AMPere NONE} <m> ::= 1-4 in NR1 format

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

Command	Query	Options and Query Returns
:FUNCTION<m>:DISPLAY { {0 OFF} {1 ON}} (see page 353)	:FUNCTION<m>:DISPLAY? (see page 353)	{0 1} <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:CENTER <frequency> (see page 354)	:FUNCTION<m>[:FFT]:CENTER? (see page 354)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz. <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:SPAN (see page 355)	:FUNCTION<m>[:FFT]:SPAN? (see page 355)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:VTPe <units> (see page 356)	:FUNCTION<m>[:FFT]:VTPe? (see page 356)	<units> ::= {DECibel VRMS} <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:WINDOW <>window> (see page 357)	:FUNCTION<m>[:FFT]:WINDOW? (see page 357)	<window> ::= {RECTangular HANNing FLATtop BHARris} <m> ::= 1-4 in NR1 format
:FUNCTION<m>:FREQuency:HIGHpass <3dB_freq> (see page 358)	:FUNCTION<m>:FREQuency:HIGHpass? (see page 358)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:FREQuency:LOWPass <3dB_freq> (see page 359)	:FUNCTION<m>:FREQuency:LOWPass? (see page 359)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:INTEGRATE:IOFFset <input_offset> (see page 360)	:FUNCTION<m>:INTEGRATE:IOFFset? (see page 360)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1-4 in NR1 format
:FUNCTION<m>:LINEar:GAIN <value> (see page 361)	:FUNCTION<m>:LINEar:GAIN? (see page 361)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:LINEar:OFFSet <value> (see page 362)	:FUNCTION<m>:LINEar:OFFSet? (see page 362)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:OFFSET <offset> (see page 363)	:FUNCTION<m>:OFFSET? (see page 363)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1-4 in NR1 format

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

Command	Query	Options and Query Returns
:FUNCtion<m>:OPERatio n <operation> (see page 364)	:FUNCtion<m>:OPERatio n? (see page 365)	<operation> ::= {ADD SUBTract MULTiply DIVide INTegrate DIFF FFT SQRT MAGNify ABSolute SQUare LN LOG EXP TEN LOWPass HIGHpass AVERage LINEar TRENd BTIMing BSTate} <m> ::= 1-4 in NR1 format
:FUNCtion<m>:RANGE <range> (see page 367)	:FUNCtion<m>:RANGE? (see page 367)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV. <m> ::= 1-4 in NR1 format
:FUNCtion<m>:REFerenc e <level> (see page 368)	:FUNCtion<m>:REFerenc e? (see page 368)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1-4 in NR1 format
:FUNCtion<m>:SCALe <scale value>[<suffix>] (see page 369)	:FUNCtion<m>:SCALe? (see page 369)	<scale value> ::= integer in NR1 format <suffix> ::= {V dB} <m> ::= 1-4 in NR1 format
:FUNCtion<m>:SOURcel <source> (see page 370)	:FUNCtion<m>:SOURcel? (see page 370)	<source> ::= {CHANnel<n> FUNCtion<c> MATH<c> BUS} <n> ::= 1 to (# analog channels) in NR1 format <c> ::= {1 2 3}, must be lower than <m> ::= {1 2} <m> ::= 1-4 in NR1 format

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

Command	Query	Options and Query Returns
:FUNCTION<m>:SOURCE2 <source> (see page 372)	:FUNCTION<m>:SOURCE2? (see page 372)	<source> ::= {CHANnel<n> NONE} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:TREND:MEA surement <type> (see page 373)	:FUNCTION<m>:TREND:MEA surement? (see page 373)	<type> ::= {VAVerage ACRMs VRATio PERiod FREQuency PWIDth NWIDTH DUTYcycle RISetime FALLtime} <m> ::= 1-4 in NR1 format

Table 14 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 377)	:HARDcopy:AREA? (see page 377)	<area> ::= SCreen
:HARDcopy:APRinter <active_printer> (see page 378)	:HARDcopy:APRinter? (see page 378)	<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 379)	:HARDcopy:FACTors? (see page 379)	{0 1}
:HARDcopy:FFEed {{0 OFF} {1 ON}} (see page 380)	:HARDcopy:FFEed? (see page 380)	{0 1}
:HARDcopy:INKSaver { {0 OFF} {1 ON}} (see page 381)	:HARDcopy:INKSaver? (see page 381)	{0 1}
:HARDcopy:LAYOUT <layout> (see page 382)	:HARDcopy:LAYOUT? (see page 382)	<layout> ::= {LANDscape PORTRait}
:HARDcopy:NETWork:ADD Ress <address> (see page 383)	:HARDcopy:NETWork:ADD Ress? (see page 383)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see page 384)	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see page 385)	:HARDcopy:NETWork:DOM ain? (see page 385)	<domain> ::= quoted ASCII string

Table 14 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:NETWork:PAS Sword <password> (see page 386)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLO T <slot> (see page 387)	:HARDcopy:NETWork:SLO T? (see page 387)	<slot> ::= {NET0 NET1}
:HARDcopy:NETWork:USE Rname <username> (see page 388)	:HARDcopy:NETWork:USE Rname? (see page 388)	<username> ::= quoted ASCII string
:HARDcopy:PAlette <palette> (see page 389)	:HARDcopy:PAlette? (see page 389)	<palette> ::= {COLOR GRAYscale NONE}
n/a	:HARDcopy:PRINTER:LIS T? (see page 390)	<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 391)	n/a	n/a

Table 15 :LISTer Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTer:DATA? (see page 394)	<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 395)	:LISTer:DISPlay? (see page 395)	{OFF SBUS1 SBUS2 ALL}
:LISTer:REFerence <time_ref> (see page 396)	:LISTer:REFerence? (see page 396)	<time_ref> ::= {TRIGger PREVIOUS}

Table 16 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 400)	<return_value> ::= ΔY/ΔX value in NR3 format
:MARKer:MODE <mode> (see page 401)	:MARKer:MODE? (see page 401)	<mode> ::= {OFF MEASurement MANual WAVEform BINARY HEX}
:MARKer:X1Position <position>[suffix] (see page 402)	:MARKer:X1Position? (see page 402)	<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 403)	:MARKer:X1Y1source? (see page 403)	<source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 404)	:MARKer:X2Position? (see page 404)	<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 405)	:MARKer:X2Y2source? (see page 405)	<source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 406)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:XUNits <mode> (see page 407)	:MARKer:XUNits? (see page 407)	<units> ::= {SEConds HERTZ DEGREes PERCent}
:MARKer:XUNits:USE (see page 408)	n/a	n/a

4 Commands Quick Reference

Table 16 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:Y1Position <position>[suffix] (see page 409)	:MARKer:Y1Position? (see page 409)	<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 410)	:MARKer:Y2Position? (see page 410)	<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 411)	<return_value> ::= Y cursors delta value in NR3 format
:MARKer:YUNits <mode> (see page 412)	:MARKer:YUNits? (see page 412)	<units> ::= {BASE PERCent}
:MARKer:YUNits:USE (see page 413)	n/a	n/a

Table 17 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 430)	n/a	n/a
:MEASure:AREA [<interval>][,<source>] (see page 431)	:MEASure:AREa? [<interval>][,<source>] (see page 431)	<interval> ::= {CYCLE DISPLAY} <source> ::= {CHANNEL<n> FUNCTION<m> MATH<m> WMEMORY<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= area in volt-seconds, NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:BWIDth [<source>] (see page 432)	:MEASure:BWIDth? [<source>] (see page 432)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= burst width in seconds, NR3 format</p>
:MEASure:CLEar (see page 433)	n/a	n/a
:MEASure:COUNTER [<source>] (see page 434)	:MEASure:COUNTER? [<source>] (see page 434)	<p><source> ::= {CHANnel<n> EXTERNAL} for DSO models</p> <p><source> ::= {CHANnel<n> DIGITAL<d> EXTERNAL} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= counter frequency in Hertz in NR3 format</p>
:MEASure:DEFine DElAy, <delay spec> (see page 436)	:MEASure:DEFine? DElAy (see page 437)	<p><delay spec> ::=</p> <p><edge_spec1>, <edge_spec2></p> <p>edge_spec1 ::= [<slope>]<occurrence></p> <p>edge_spec2 ::= [<slope>]<occurrence></p> <p><slope> ::= {+ -}</p> <p><occurrence> ::= integer</p>
:MEASure:DEFine THReSholds, <threshold spec> (see page 436)	:MEASure:DEFine? THReSholds (see page 437)	<p><threshold spec> ::= {STANDARD} {<threshold mode>, <upper>, <middle>, <lower>}</p> <p><threshold mode> ::= {PERCent ABSolute}</p>

4 Commands Quick Reference

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DELay [<source1>] [,<source2>] (see page 439)	:MEASure:DELay? [<source1>] [,<source2>] (see page 439)	<source1,2> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUAL:CHARge [<interval>] [,<source1>][,<source2>] (see page 441)	:MEASure:DUAL:CHARge? [<interval>] [,<source1>][,<source2>] (see page 441)	<interval> ::= {CYCLE DISPLAY} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= area in Amp-hours, NR3 format
:MEASure:DUAL:VAMPtitude [<source1>][,<source2>] (see page 442)	:MEASure:DUAL:VAMPtitude? [<source1>][,<source2>] (see page 442)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:DUAL:VAVerage [<interval>] [,<source1>][,<source2>] (see page 443)	:MEASure:DUAL:VAVerage? [<interval>] [,<source1>][,<source2>] (see page 443)	<interval> ::= {CYCLE DISPLAY} <source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:DUAL:VBASe [<source1>][,<source2>] (see page 444)	:MEASure:DUAL:VBASe? [<source1>][,<source2>] (see page 444)	<source1>,<source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUAL:VPP [<source1>] [,<source2>] (see page 445)	:MEASure:DUAL:VPP? [<source1>] [,<source2>] (see page 445)	<source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:DUAL:VRMS [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 446)	:MEASure:DUAL:VRMS? [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 446)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated RMS voltage in NR3 format
:MEASure:DUTYcycle [<source>] (see page 447)	:MEASure:DUTYcycle? [<source>] (see page 447)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 448)	:MEASure:FALLtime? [<source>] (see page 448)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= time in seconds between the lower and upper thresholds in NR3 format</p>
:MEASure:FREQuency [<source>] (see page 449)	:MEASure:FREQuency? [<source>] (see page 449)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= frequency in Hertz in NR3 format</p>

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NDUTy [<source>] (see page 450)	:MEASure:NDUTy? [<source>] (see page 450)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= ratio of negative pulse width to period in NR3 format</p>
:MEASure:NEDGes [<source>] (see page 451)	:MEASure:NEDGes? [<source>] (see page 451)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the falling edge count in NR3 format</p>
:MEASure:NPULses [<source>] (see page 452)	:MEASure:NPULses? [<source>] (see page 452)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the falling pulse count in NR3 format</p>

4 Commands Quick Reference

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 453)	:MEASure:NWIDth? [<source>] (see page 453)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= negative pulse width in seconds-NR3 format</p>
:MEASure:OVERshoot [<source>] (see page 454)	:MEASure:OVERshoot? [<source>] (see page 454)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the percent of the overshoot of the selected waveform in NR3 format</p>
:MEASure:PEDGes [<source>] (see page 456)	:MEASure:PEDGes? [<source>] (see page 456)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the rising edge count in NR3 format</p>

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PERiod [<source>] (see page 457)	:MEASure:PERiod? [<source>] (see page 457)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= waveform period in seconds in NR3 format</p>
:MEASure:PHASE [<source1>] [,<source2>] (see page 458)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 458)	<p><source1,2> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the phase angle value in degrees in NR3 format</p>
:MEASure:PPULses [<source>] (see page 459)	:MEASure:PPULses? [<source>] (see page 459)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the rising pulse count in NR3 format</p>

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PREShoot [<source>] (see page 460)	:MEASure:PREShoot? [<source>] (see page 460)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 461)	:MEASure:PWIDth? [<source>] (see page 461)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMMemory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:REsults? <result_list> (see page 462)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISetime [<source>] (see page 465)	:MEASure:RISetime? [<source>] (see page 465)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SDEViation [<source>] (see page 466)	:MEASure:SDEViation? [<source>] (see page 466)	<source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 OFF} {1 ON}} (see page 467)	:MEASure:SHOW? (see page 467)	{0 1}
:MEASure:SOURce <source1> [,<source2>] (see page 468)	:MEASure:SOURce? (see page 468)	<source1,2> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMMemory<r> EXTERNAL} for DSO models <source1,2> ::= {CHANnel<n> DIGITAL<d> FUNCTION<m> MATH<m> WMMemory<r> EXTERNAL} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> NONE}
:MEASure:STATistics <type> (see page 470)	:MEASure:STATistics? (see page 470)	<type> ::= {{ON 1} CURRent MEAN MINimum MAXimum STDDev COUNT} ON ::= all statistics returned
:MEASure:STATistics:D ISPlay {{0 OFF} {1 ON}} (see page 471)	:MEASure:STATistics:D ISPlay? (see page 471)	{0 1}
:MEASure:STATistics:I NCREMENT (see page 472)	n/a	n/a

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:STATistics:M CCount <setting> (see page 473)	:MEASure:STATistics:M CCount? (see page 473)	<setting> ::= {INFinite <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:R ESet (see page 474)	n/a	n/a
:MEASure:STATistics:R SDeviation {{0 OFF} {1 ON}} (see page 475)	:MEASure:STATistics:R SDeviation? (see page 475)	{0 1}
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 476)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TValue? <value>, [<slope>]<occurrence> [,<source>] (see page 479)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMplitude [<source>] (see page 481)	:MEASure:VAMplitude? [<source>] (see page 481)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<interval>] [,<source >] (see page 482)	:MEASure:VAverage? [<interval>] [,<source >] (see page 482)	<interval> ::= {CYCLE DISPLAY} <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VBASe [<source>] (see page 483)	:MEASure:VBASe? [<source>] (see page 483)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 484)	:MEASure:VMAX? [<source>] (see page 484)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 485)	:MEASure:VMIN? [<source>] (see page 485)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 486)	:MEASure:VPP? [<source>] (see page 486)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRATio [<interval>] [,<source 1>] [,<source2>] (see page 487)	:MEASure:VRATio? [<interval>] [,<source 1>] [,<source2>] (see page 487)	<interval> ::= {CYCLE DISPLAY} <source1,2> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>] [,<type>] [,<source>] (see page 488)	:MEASure:VRMS? [<interval>] [,<type>] [,<source>] (see page 488)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtim>[,<source>] (see page 489)	<vtim> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

4 Commands Quick Reference

Table 17 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 490)	:MEASure:VTOP? [<source>] (see page 490)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDOW <type> (see page 491)	:MEASure:WINDOW? (see page 491)	<type> ::= {MAIN ZOOM AUTO GATE}
:MEASure:XMAX [<source>] (see page 492)	:MEASure:XMAX? [<source>] (see page 492)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 493)	:MEASure:XMIN? [<source>] (see page 493)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format

Table 18 :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0 OFF} {1 ON}} (see page 522)	:MTESt:ALL? (see page 522)	{0 1}
:MTESt:AMASK:CREate (see page 523)	n/a	n/a

Table 18 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:AMASK:SOURCE <source> (see page 524)	:MTEST:AMASK:SOURce? (see page 524)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:MTEST:AMASK:UNITS <units> (see page 525)	:MTEST:AMASK:UNITS? (see page 525)	<units> ::= {CURREnt DIVisions}
:MTEST:AMASK:XDELta <value> (see page 526)	:MTEST:AMASK:XDELta? (see page 526)	<value> ::= X delta value in NR3 format
:MTEST:AMASK:YDELta <value> (see page 527)	:MTEST:AMASK:YDELta? (see page 527)	<value> ::= Y delta value in NR3 format
n/a	:MTEST:COUNT:FWAVEforms? [CHANnel<n>] (see page 528)	<failed> ::= number of failed waveforms in NR1 format
:MTEST:COUNT:RESet (see page 529)	n/a	n/a
n/a	:MTEST:COUNT:TIME? (see page 530)	<time> ::= elapsed seconds in NR3 format
n/a	:MTEST:COUNT:WAVEforms? (see page 531)	<count> ::= number of waveforms in NR1 format
:MTEST:DATA <mask> (see page 532)	:MTEST:DATA? (see page 532)	<mask> ::= data in IEEE 488.2 # format.
:MTEST:DELETE (see page 533)	n/a	n/a
:MTEST:ENABLE {{0 OFF} {1 ON}} (see page 534)	:MTEST:ENABLE? (see page 534)	{0 1}
:MTEST:LOCK {{0 OFF} {1 ON}} (see page 535)	:MTEST:LOCK? (see page 535)	{0 1}
:MTEST:RMODE <rmode> (see page 536)	:MTEST:RMODE? (see page 536)	<rmode> ::= {FORever TIME SIGMa WAVEforms}
:MTEST:RMODE:FACTion:MEASure {{0 OFF} {1 ON}} (see page 537)	:MTEST:RMODE:FACTion:MEASure? (see page 537)	{0 1}

Table 18 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:RMODE:FACTion: PRINT {{0 OFF} {1 ON}} (see page 538)	:MTEST:RMODE:FACTion: PRINT? (see page 538)	{0 1}
:MTEST:RMODE:FACTion: SAVE {{0 OFF} {1 ON}} (see page 539)	:MTEST:RMODE:FACTion: SAVE? (see page 539)	{0 1}
:MTEST:RMODE:FACTion: STOP {{0 OFF} {1 ON}} (see page 540)	:MTEST:RMODE:FACTion: STOP? (see page 540)	{0 1}
:MTEST:RMODE:SIGMa <level> (see page 541)	:MTEST:RMODE:SIGMa? (see page 541)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTEST:RMODE:TIME <seconds> (see page 542)	:MTEST:RMODE:TIME? (see page 542)	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAVeform s <count> (see page 543)	:MTEST:RMODE:WAVeform s? (see page 543)	<count> ::= number of waveforms in NR1 format
:MTEST:SCALe:BIND {{0 OFF} {1 ON}} (see page 544)	:MTEST:SCALe:BIND? (see page 544)	{0 1}
:MTEST:SCALe:X1 <x1_value> (see page 545)	:MTEST:SCALe:X1? (see page 545)	<x1_value> ::= X1 value in NR3 format
:MTEST:SCALe:XDELta <xdelta_value> (see page 546)	:MTEST:SCALe:XDELta? (see page 546)	<xdelta_value> ::= X delta value in NR3 format
:MTEST:SCALe:Y1 <y1_value> (see page 547)	:MTEST:SCALe:Y1? (see page 547)	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALe:Y2 <y2_value> (see page 548)	:MTEST:SCALe:Y2? (see page 548)	<y2_value> ::= Y2 value in NR3 format
:MTEST:SOURce <source> (see page 549)	:MTEST:SOURce? (see page 549)	<source> ::= {CHANnel<n> NONE} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
n/a	:MTEST:TITLe? (see page 550)	<title> ::= a string of up to 128 ASCII characters

Table 19 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 OFF} {1 ON}} (see page 553)	:POD<n>:DISPlay? (see page 553)	{0 1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see page 554)	:POD<n>:SIZE? (see page 554)	<value> ::= {SMALL MEDium LARGe}
:POD<n>:THreshold <type>[suffix] (see page 555)	:POD<n>:THreshold? (see page 555)	<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 20 :POWer Commands Summary

Command	Query	Options and Query Returns
:POWer:DESKew (see page 562)	n/a	n/a
:POWer:EFFiciency:APP Ly (see page 563)	n/a	n/a
:POWer:ENABLE {{0 OFF} {1 ON}} (see page 564)	:POWer:ENABLE? (see page 564)	{0 1}
:POWer:HARMonics:APPL y (see page 565)	n/a	n/a
n/a	:POWer:HARMonics:DATA ? (see page 566)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISP lay <display> (see page 567)	:POWer:HARMonics:DISP lay? (see page 567)	<display> ::= {TABLE BAR OFF}
n/a	:POWer:HARMonics:FAIL count? (see page 568)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 569)	:POWer:HARMonics:LINE ? (see page 569)	<frequency> ::= {F50 F60 F400}
n/a	:POWer:HARMonics:POWe rfactor? (see page 570)	<value> ::= Class C power factor in NR3 format

4 Commands Quick Reference

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:POWer:HARMonics:RUNCOUNT? (see page 571)	<count> ::= integer in NR1 format
:POWer:HARMonics:STANDARD <class> (see page 572)	:POWer:HARMonics:STANDARD? (see page 572)	<class> ::= {A B C D}
n/a	:POWer:HARMonics:STATUS? (see page 573)	<status> ::= {PASS FAIL UNTested}
n/a	:POWer:HARMonics:THD? (see page 574)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLY (see page 575)	n/a	n/a
:POWer:INRush:EXIT (see page 576)	n/a	n/a
:POWer:INRush:NEXT (see page 577)	n/a	n/a
:POWer:MODulation:APPly (see page 578)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 579)	:POWer:MODulation:SOURce? (see page 579)	<source> ::= {V I}
:POWer:MODulation:TYPe <modulation> (see page 580)	:POWer:MODulation:TYPe? (see page 580)	<modulation> ::= {VAverage ACRMs VRATio PERiod FREQuency PWIDith NWIDth DUTYcycle RISetime FALLtime}
:POWer:ONOOff:APPLY (see page 581)	n/a	n/a
:POWer:ONOOff:EXIT (see page 582)	n/a	n/a
:POWer:ONOOff:NEXT (see page 583)	n/a	n/a
:POWer:ONOOff:TEST {{0 OFF} {1 ON}} (see page 584)	:POWer:ONOOff:TEST? (see page 584)	{0 1}
:POWer:PSRR:APPLY (see page 585)	n/a	n/a
:POWer:PSRR:FREQuency :MAXimum <value>[suffix] (see page 586)	:POWer:PSRR:FREQuency :MAXimum? (see page 586)	<value> ::= {10 100 1000 10000 100000 1000000 10000000 20000000} [suffix] ::= {Hz kHz MHz}

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:FREQuency :MINimum <value>[suffix] (see page 587)	:POWer:PSRR:FREQuency :MINimum? (see page 587)	<value> ::= {1 10 100 1000 10000 100000 1000000} [suffix] ::= {Hz kHz MHz}
:POWer:PSRR:RMAXimum <value> (see page 588)	:POWer:PSRR:RMAXimum?	<value> ::= Maximum ratio value in NR1 format
:POWer:QUALity:APPLy (see page 589)	n/a	n/a
:POWer:RIPPle:APPLy (see page 590)	n/a	n/a
:POWer:SIGNals:AUTose tup <analysis> (see page 591)	n/a	<analysis> ::= {HARMonics EFFiciency RIPPLE MODulation QUALity SLEW SWITch}
:POWer:SIGNals:CYCLES :HARMonics <count> (see page 592)	:POWer:SIGNals:CYCLES :HARMonics? (see page 592)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:CYCLES :QUALity <count> (see page 593)	:POWer:SIGNals:CYCLES :QUALity? (see page 593)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURati on:EFFiciency <value>[suffix] (see page 594)	:POWer:SIGNals:DURati on:EFFiciency? (see page 594)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:MODulation <value>[suffix] (see page 595)	:POWer:SIGNals:DURati on:MODulation? (see page 595)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:ONOFF:OFF <value>[suffix] (see page 596)	:POWer:SIGNals:DURati on:ONOFF:OFF? (see page 596)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:ONOFF:ON <value>[suffix] (see page 597)	:POWer:SIGNals:DURati on:ONOFF:ON? (see page 597)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:RIPPLE <value>[suffix] (see page 598)	:POWer:SIGNals:DURati on:RIPPLE? (see page 598)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:DURati on:TRANsient <value>[suffix] (see page 599)	:POWer:SIGNals:DURati on:TRANsient? (see page 599)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:IEXPec ted <value>[suffix] (see page 600)	:POWer:SIGNals:IEXPec ted? (see page 600)	<value> ::= Expected current value in NR3 format [suffix] ::= {A mA}
:POWer:SIGNals:OVERsh oot <percent> (see page 601)	:POWer:SIGNals:OVERsh oot? (see page 601)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V mV}}
:POWer:SIGNals:VMAXim um:INRush <value>[suffix] (see page 602)	:POWer:SIGNals:VMAXim um:INRush? (see page 602)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VMAXim um:ONOFF:OFF <value>[suffix] (see page 603)	:POWer:SIGNals:VMAXim um:ONOFF:OFF? (see page 603)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VMAXim um:ONOFF:ON <value>[suffix] (see page 604)	:POWer:SIGNals:VMAXim um:ONOFF:ON? (see page 604)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTead y:ONOOff:OFF <value>[suffix] (see page 605)	:POWer:SIGNals:VSTead y:ONOOff:OFF? (see page 605)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTead y:ONOOff:ON <value>[suffix] (see page 606)	:POWer:SIGNals:VSTead y:ONOOff:ON? (see page 606)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTead y:TRANsient <value>[suffix] (see page 607)	:POWer:SIGNals:VSTead y:TRANsient? (see page 607)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:SOURce :CURRent<i> <source> (see page 608)	:POWer:SIGNals:SOURce :CURRent<i>? (see page 608)	<i> ::= 1, 2 in NR1 format <source> ::= CHANNEL<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 20 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:SOURce :VOLTage<i> <source> (see page 609)	:POWer:SIGNals:SOURce :VOLTage<i>? (see page 609)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SLEW:APPLy (see page 610)	n/a	n/a
:POWer:SLEW:SOURce <source> (see page 611)	:POWer:SLEW:SOURce? (see page 611)	<source> ::= {V I}
:POWer:SWITch:APPLy (see page 612)	n/a	n/a
:POWer:SWITch:CONDuction <conduction> (see page 613)	:POWer:SWITch:CONDuction? (see page 613)	<conduction> ::= {WAVEform RDS VCE}
:POWer:SWITch:IREFere nce <percent> (see page 614)	:POWer:SWITch:IREFere nce? (see page 614)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 615)	:POWer:SWITch:RDS? (see page 615)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 616)	:POWer:SWITch:VCE? (see page 616)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V mV}
:POWer:SWITch:VREFere nce <percent> (see page 617)	:POWer:SWITch:VREFere nce? (see page 617)	<percent> ::= percent in NR1 format
:POWer:TRANSient:APPLy (see page 618)	n/a	n/a
:POWer:TRANSient:EXIT (see page 619)	n/a	n/a
:POWer:TRANSient:IINI tial <value>[suffix] (see page 620)	:POWer:TRANSient:IINI tial? (see page 620)	<value> ::= Initial current value in NR3 format [suffix] ::= {A mA}
:POWer:TRANSient:INEW <value>[suffix] (see page 621)	:POWer:TRANSient:INEW ? (see page 621)	<value> ::= New current value in NR3 format [suffix] ::= {A mA}
:POWer:TRANSient:NEXT (see page 622)	n/a	n/a

Table 21 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARbitrary[:START] [<file_spec>] [, <column>] [, <wavegen_id>] (see page 625)	n/a	<p><file_spec> ::= {<internal_loc> <file_name>}</p> <p><column> ::= Column in CSV file to load. Column number starts from 1.</p> <p><internal_loc> ::= 0-3; an integer in NR1 format</p> <p><file_name> ::= quoted ASCII string</p> <p><wavegen_id> ::= {WGEN1 WGEN2}</p>
:RECall:DBC[:START] [<file_name>] [, <serialbus>] (see page 626)	n/a	<p><file_name> ::= quoted ASCII string</p> <p>If extension included in file name, it must be ".dbc".</p> <p><serialbus> ::= {SBUS<n>}</p> <p><n> ::= 1 to (# of serial bus) in NR1 format</p>
:RECall:FILEname <base_name> (see page 627)	:RECall:FILEname? (see page 627)	<p><base_name> ::= quoted ASCII string</p>
:RECall:MASK[:START] [<file_spec>] (see page 628)	n/a	<p><file_spec> ::= {<internal_loc> <file_name>}</p> <p><internal_loc> ::= 0-3; an integer in NR1 format</p> <p><file_name> ::= quoted ASCII string</p>
:RECall:PWD <path_name> (see page 629)	:RECall:PWD? (see page 629)	<p><path_name> ::= quoted ASCII string</p>
:RECall:SETup[:START] [<file_spec>] (see page 630)	n/a	<p><file_spec> ::= {<internal_loc> <file_name>}</p> <p><internal_loc> ::= 0-9; an integer in NR1 format</p> <p><file_name> ::= quoted ASCII string</p>
:RECall:WMEMory<r>[:START] [<file_name>] (see page 631)	n/a	<p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><file_name> ::= quoted ASCII string</p> <p>If extension included in file name, it must be ".h5".</p>

Table 22 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary[:STARt] [<file_spec>] [, <wavegen_id>] (see page 636)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= {WGEN1 WGEN2}
:SAVE:COMpliance:USB[:START] [<file_name>] (see page 637)	n/a	<file_name> ::= quoted ASCII string
:SAVE:FIlename <base_name> (see page 638)	:SAVE:FIlename? (see page 638)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:START] [<file_name>] (see page 639)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGe:FACTOrs {{0 OFF} {1 ON}} (see page 640)	:SAVE:IMAGe:FACTOrs? (see page 640)	{0 1}
:SAVE:IMAGe:FORMAT <format> (see page 641)	:SAVE:IMAGe:FORMAT? (see page 641)	<format> ::= {{BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGe:INKSaver {{0 OFF} {1 ON}} (see page 642)	:SAVE:IMAGe:INKSaver? (see page 642)	{0 1}
:SAVE:IMAGe:PALETTE <palette> (see page 643)	:SAVE:IMAGe:PALETTE? (see page 643)	<palette> ::= {COLOR GRAYscale}
:SAVE:LISTER[:START] [<file_name>] (see page 644)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:START] [<file_spec>] (see page 645)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:MULTi[:START] [<file_name>] (see page 646)	n/a	<file_name> ::= quoted ASCII string

Table 22 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:POWer[:START] [<file_name>] (see page 647)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 648)	:SAVE:PWD? (see page 648)	<path_name> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see page 649)	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 650)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMAT <format> (see page 651)	:SAVE:WAVeform:FORMAT? (see page 651)	<format> ::= {ASCIixy CSV BINary NONE}
:SAVE:WAVeform:LENGTH <length> (see page 652)	:SAVE:WAVeform:LENGTH? (see page 652)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGTH :MAX {{0 OFF} {1 ON}} (see page 653)	:SAVE:WAVeform:LENGTH :MAX? (see page 653)	{0 1}
:SAVE:WAVeform:SEGMen ted <option> (see page 654)	:SAVE:WAVeform:SEGMen ted? (see page 654)	<option> ::= {ALL CURRent}
:SAVE:WMEMory:SOURce <source> (see page 655)	:SAVE:WMEMory:SOURce? (see page 655)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:STARt] [<file_name>] (see page 656)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

Table 23 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 660)	:SBUS<n>:DISPlay? (see page 660)	{0 1}
:SBUS<n>:MODE <mode> (see page 661)	:SBUS<n>:MODE? (see page 661)	<mode> ::= {A429 CAN FLEXray I2S IIC LIN M1553 SPI UART USB}

Table 24 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUTOset up (see page 664)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 665)	:SBUS<n>:A429:BASE? (see page 665)	<base> ::= {BINary HEX}
n/a	:SBUS<n>:A429:COUNT:E RRor? (see page 666)	<error_count> ::= integer in NR1 format
:SBUS<n>:A429:COUNT:R ESet (see page 667)	n/a	n/a
n/a	:SBUS<n>:A429:COUNT:W ORD? (see page 668)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMAT <format> (see page 669)	:SBUS<n>:A429:FORMAT? (see page 669)	<format> ::= {LDSDi LDSSm LDAData}
:SBUS<n>:A429:SIGNAl <signal> (see page 670)	:SBUS<n>:A429:SIGNAl? (see page 670)	<signal> ::= {A B DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 671)	:SBUS<n>:A429:SOURce? (see page 671)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEEd <speed> (see page 672)	:SBUS<n>:A429:SPEEd? (see page 672)	<speed> ::= {LOW HIGH}

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

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :LABel <value> (see page 673)	:SBUS<n>:A429:TRIGger :LABel? (see page 673)	<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 674)	:SBUS<n>:A429:TRIGger :PATTern:DATA? (see page 674)	<string> ::= "nn...n" where n ::= {0 1 X}, length depends on FORMat
:SBUS<n>:A429:TRIGger :PATTern:SDI <string> (see page 675)	:SBUS<n>:A429:TRIGger :PATTern:SDI? (see page 675)	<string> ::= "nn" where n ::= {0 1 X}, length always 2 bits
:SBUS<n>:A429:TRIGger :PATTern:SSM <string> (see page 676)	:SBUS<n>:A429:TRIGger :PATTern:SSM? (see page 676)	<string> ::= "nn" where n ::= {0 1 X}, length always 2 bits
:SBUS<n>:A429:TRIGger :RANGE <min>,<max> (see page 677)	:SBUS<n>:A429:TRIGger :RANGE? (see page 677)	<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 678)	:SBUS<n>:A429:TRIGger :TYPE? (see page 678)	<condition> ::= {WSTArt WSTOp LABel LBITS PERRor WERRor GERRor WGERRors ALLerrors LRANge ABITs AOBits AZBits}

Table 25 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNT:ER Ror? (see page 682)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:OV ERload? (see page 683)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNT:RE Set (see page 684)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNT:TO Tal? (see page 685)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:UT ILization? (see page 686)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPLAY <type> (see page 687)	:SBUS<n>:CAN:DISPLAY? (see page 687)	<type> ::= {HEXAdecimal SYMBOLic}
:SBUS<n>:CAN:SAMPLEpo int <value> (see page 688)	:SBUS<n>:CAN:SAMPLEpo int? (see page 688)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:CAN:SIGNAl:B AUDrate <baudrate> (see page 689)	:SBUS<n>:CAN:SIGNAl:B AUDrate? (see page 689)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNAl:D EFinition <value> (see page 690)	:SBUS<n>:CAN:SIGNAl:D EFinition? (see page 690)	<value> ::= {CANH CANL RX TX DIFFerential DIFL DIFH}
:SBUS<n>:CAN:SOURce <source> (see page 691)	:SBUS<n>:CAN:SOURce? (see page 691)	<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 692)	:SBUS<n>:CAN:TRIGger? (see page 693)	<condition> ::= {SOF DATA ERRor IDData IDEither IDRemote ALLerrors OVERload ACKerror MESSage MSIGnal}
:SBUS<n>:CAN:TRIGger: PATTERn:DATA <string> (see page 695)	:SBUS<n>:CAN:TRIGger: PATTERn:DATA? (see page 695)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}

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

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGTH <length> (see page 696)	:SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGTH? (see page 696)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger:PATTern:ID <string> (see page 697)	:SBUS<n>:CAN:TRIGger:PATTern:ID? (see page 697)	<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 698)	:SBUS<n>:CAN:TRIGger:PATTern:ID:MODE? (see page 698)	<value> ::= {STANDARD EXTENDED}
:SBUS<n>:CAN:TRIGger:SYMBOLic:MESSAge <name> (see page 699)	:SBUS<n>:CAN:TRIGger:SYMBOLic:MESSAge? (see page 699)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBOLic:SIGNAl <name> (see page 700)	:SBUS<n>:CAN:TRIGger:SYMBOLic:SIGNAl? (see page 700)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger:SYMBOLic:VALue <data> (see page 701)	:SBUS<n>:CAN:TRIGger:SYMBOLic:VALue? (see page 701)	<data> ::= value in NR3 format

Table 26 :SBUS<n>:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:AUTO setup (see page 704)	n/a	n/a
:SBUS<n>:FLEXray:BAUD rate <baudrate> (see page 705)	:SBUS<n>:FLEXray:BAUD rate? (see page 705)	<baudrate> ::= {2500000 5000000 10000000}
:SBUS<n>:FLEXray:CHAN nel <channel> (see page 706)	:SBUS<n>:FLEXray:CHAN nel? (see page 706)	<channel> ::= {A B}
n/a	:SBUS<n>:FLEXray:COUN t:NULL? (see page 707)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:COUN t:RESET (see page 708)	n/a	n/a

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

Command	Query	Options and Query Returns
n/a	:SBUS<n>:FLEXray:COUNT:SYNC? (see page 709)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:FLEXray:COUNT:TOTal? (see page 710)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:SOURCE <source> (see page 711)	:SBUS<n>:FLEXray:SOURCE? (see page 711)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format
:SBUS<n>:FLEXray:TRIGGER <condition> (see page 712)	:SBUS<n>:FLEXray:TRIGGER? (see page 712)	<condition> ::= {FRAMe ERRor EVENT}
:SBUS<n>:FLEXray:TRIGGER:ERRor:TYPE <error_type> (see page 713)	:SBUS<n>:FLEXray:TRIGGER:ERRor:TYPE? (see page 713)	<error_type> ::= {ALL HCRC FCRC}
:SBUS<n>:FLEXray:TRIGGER:EVENT:AUToset (see page 714)	n/a	n/a
:SBUS<n>:FLEXray:TRIGGER:EVENT:BSS:ID <frame_id> (see page 715)	:SBUS<n>:FLEXray:TRIGGER:EVENT:BSS:ID? (see page 715)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIGGER:EVENT:TYPE <event> (see page 716)	:SBUS<n>:FLEXray:TRIGGER:EVENT:TYPE? (see page 716)	<event> ::= {WAKEup TSS {FES DTS} BSS}
:SBUS<n>:FLEXray:TRIGGER:FRAMe:CCBase <cycle_count_base> (see page 717)	:SBUS<n>:FLEXray:TRIGGER:FRAMe:CCBase? (see page 717)	<cycle_count_base> ::= integer from 0-63
:SBUS<n>:FLEXray:TRIGGER:FRAMe:CCRepetition <cycle_count_repetition> (see page 718)	:SBUS<n>:FLEXray:TRIGGER:FRAMe:CCRepetition? (see page 718)	<cycle_count_repetition> ::= {ALL <rep #>} <rep #> ::= integer values 2, 4, 8, 16, 32, or 64

4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:TRIG ger:FRAMe:ID <frame_id> (see page 719)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:ID? (see page 719)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIG ger:FRAMe:TYPE <frame_type> (see page 720)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:TYPE? (see page 720)	<frame_type> ::= {NORMAl STARtup NULL SYNC NSTArtup NNULl NSYNC ALL}

Table 27 :SBUS<n>:I2S Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:I2S:ALIGnmen t <setting> (see page 723)	:SBUS<n>:I2S:ALIGnmen t? (see page 723)	<setting> ::= {I2S LJ RJ}
:SBUS<n>:I2S:BASE <base> (see page 724)	:SBUS<n>:I2S:BASE?	<base> ::= {DECimal HEX}
:SBUS<n>:I2S:CLOCK:SL OPe <slope> (see page 725)	:SBUS<n>:I2S:CLOCK:SL OPe? (see page 725)	<slope> ::= {NEGative POSitive}
:SBUS<n>:I2S:RWIDth <receiver> (see page 726)	:SBUS<n>:I2S:RWIDth?	<receiver> ::= 4-32 in NR1 format
:SBUS<n>:I2S:SOURce:C LOCK <source> (see page 727)	:SBUS<n>:I2S:SOURce:C LOCK? (see page 727)	<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:D ATA <source> (see page 728)	:SBUS<n>:I2S:SOURce:D ATA? (see page 728)	<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 27 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:SOURCE:W SElect <source> (see page 729)	:SBUS<n>:I2S:SOURce:W SElect? (see page 729)	<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 730)	:SBUS<n>:I2S:TRIGger? (see page 730)	<operator> ::= {EQUAL NOTequal LESSthan GREaterthan INRange OUTRange INCReasing DECReasing}
:SBUS<n>:I2S:TRIGger: AUDIO <audio_ch> (see page 732)	:SBUS<n>:I2S:TRIGger: AUDIO? (see page 732)	<audio_ch> ::= {RIGHT LEFT EITHer}
:SBUS<n>:I2S:TRIGger: PATtern:DATA <string> (see page 733)	:SBUS<n>:I2S:TRIGger: PATtern:DATA? (see page 734)	<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 735)	:SBUS<n>:I2S:TRIGger: PATtern:FORMAT? (see page 735)	<base> ::= {BINary HEX DECimal}
:SBUS<n>:I2S:TRIGger: RANGE <lower>,<upper> (see page 736)	:SBUS<n>:I2S:TRIGger: RANGE? (see page 736)	<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

4 Commands Quick Reference

Table 27 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:TWIDth <word_size> (see page 738)	:SBUS<n>:I2S:TWIDth? (see page 738)	<word_size> ::= 4-32 in NR1 format
:SBUS<n>:I2S:WSLow <low_def> (see page 739)	:SBUS<n>:I2S:WSLow? (see page 739)	<low_def> ::= {LEFT RIGHT}

Table 28 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZE <size> (see page 741)	:SBUS<n>:IIC:ASIZE? (see page 741)	<size> ::= {BIT7 BIT8}
:SBUS<n>:IIC[:SOURce] :CLOCk <source> (see page 742)	:SBUS<n>:IIC[:SOURce] :CLOCK? (see page 742)	<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 743)	:SBUS<n>:IIC[:SOURce] :DATA? (see page 743)	<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 744)	:SBUS<n>:IIC:TRIGger: PATtern:ADDRess? (see page 744)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATtern:DATA <value> (see page 745)	:SBUS<n>:IIC:TRIGger: PATtern:DATA? (see page 745)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATtern:DATA2 <value> (see page 746)	:SBUS<n>:IIC:TRIGger: PATtern:DATA2? (see page 746)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}

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

Command	Query	Options and Query Returns
:SBUS<n>:IIC:TRIGger: QUALifier <value> (see page 747)	:SBUS<n>:IIC:TRIGger: QUALifier? (see page 747)	<value> ::= {EQUAL NOTequal LESSthan GREaterthan}
:SBUS<n>:IIC:TRIGger[: :TYPE] <type> (see page 748)	:SBUS<n>:IIC:TRIGger[: :TYPE]? (see page 748)	<type> ::= {START STOP READ7 READEprom WRITE7 WRITE10 NACKnowledge ANACK R7Data2 W7Data2 RESTart}

Table 29 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity { {0 OFF} {1 ON} } (see page 752)	:SBUS<n>:LIN:PARity? (see page 752)	{0 1}
:SBUS<n>:LIN:SAMPLEpo int <value> (see page 753)	:SBUS<n>:LIN:SAMPLEpo int? (see page 753)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:LIN:SIGNAl:B AUDrate <baudrate> (see page 754)	:SBUS<n>:LIN:SIGNAl:B AUDrate? (see page 754)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 755)	:SBUS<n>:LIN:SOURce? (see page 755)	<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 756)	:SBUS<n>:LIN:STANDARD ? (see page 756)	<std> ::= {LIN13 LIN20}
:SBUS<n>:LIN:SYNCbre ak <value> (see page 757)	:SBUS<n>:LIN:SYNCbre ak? (see page 757)	<value> ::= integer = {11 12 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 758)	:SBUS<n>:LIN:TRIGger? (see page 758)	<condition> ::= {SYNCbreak ID DATA}

4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger: ID <value> (see page 759)	:SBUS<n>:LIN:TRIGger: ID? (see page 759)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:SBUS<n>:LIN:TRIGger: PATtern:DATA <string> (see page 760)	:SBUS<n>:LIN:TRIGger: PATtern:DATA? (see page 760)	<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 762)	:SBUS<n>:LIN:TRIGGER: PATtern:DATA:LENGTH? (see page 762)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGGER: PATtern:FORMAT <base> (see page 763)	:SBUS<n>:LIN:TRIGGER: PATtern:FORMAT? (see page 763)	<base> ::= {BINary HEX DECimal}

Table 30 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTose tup (see page 765)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 766)	:SBUS<n>:M1553:BASE? (see page 766)	<base> ::= {BINary HEX}
:SBUS<n>:M1553:SOURce <source> (see page 767)	:SBUS<n>:M1553:SOURce? (see page 767)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGge r:PATTERn:DATA <string> (see page 768)	:SBUS<n>:M1553:TRIGge r:PATTERn:DATA? (see page 768)	<string> ::= "nn...n" where n ::= {0 1 X}

Table 30 :SBUS<n>:M1553 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:M1553:TRIGge r:RTA <value> (see page 769)	:SBUS<n>:M1553:TRIGge r:RTA? (see page 769)	<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:TRIGge r:TYPE <type> (see page 770)	:SBUS<n>:M1553:TRIGge r:TYPE? (see page 770)	<type> ::= {DSTArt DSTOp CSTArt CSTOP RTA PERROR SERROR MERROR RTA11}

Table 31 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 773)	:SBUS<n>:SPI:BITorder ? (see page 773)	<order> ::= {LSBFIRST MSBFIRST}
:SBUS<n>:SPI:CLOCK:SL OPe <slope> (see page 774)	:SBUS<n>:SPI:CLOCK:SL OPe? (see page 774)	<slope> ::= {NEGATIVE POSITIVE}
:SBUS<n>:SPI:CLOCK:TI Meout <time_value> (see page 775)	:SBUS<n>:SPI:CLOCK:TI Meout? (see page 775)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 776)	:SBUS<n>:SPI:FRAMing? (see page 776)	<value> ::= {CHIPselect NCHIPselect NOTC} TIMEout
:SBUS<n>:SPI:SOURce:C LOCK <source> (see page 777)	:SBUS<n>:SPI:SOURce:C LOCK? (see page 777)	<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:F RAMe <source> (see page 778)	:SBUS<n>:SPI:SOURce:F RAMe? (see page 778)	<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 31 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:SOURce:MISO <source> (see page 779)	:SBUS<n>:SPI:SOURce:MISO? (see page 779)	<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:MO SI <source> (see page 780)	:SBUS<n>:SPI:SOURce:MO SI? (see page 780)	<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 781)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 781)	<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 782)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTH? (see page 782)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MO SI:DATA <string> (see page 783)	:SBUS<n>:SPI:TRIGger:PATTern:MO SI:DATA? (see page 783)	<string> ::= "nn...n" where n ::= {0 1 X \$} <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:SPI:TRIGger:PATTern:MO SI:WIDTH <width> (see page 784)	:SBUS<n>:SPI:TRIGger:PATTern:MO SI:WIDTH? (see page 784)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 785)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 785)	<value> ::= {MOSI MISO}
:SBUS<n>:SPI:WIDTH <word_width> (see page 786)	:SBUS<n>:SPI:WIDTH? (see page 786)	<word_width> ::= integer 4-16 in NR1 format

Table 32 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 790)	:SBUS<n>:UART:BASE? (see page 790)	<base> ::= {ASCii BINary HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 791)	:SBUS<n>:UART:BAUDrate? (see page 791)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorder <bitorder> (see page 792)	:SBUS<n>:UART:BITorder? (see page 792)	<bitorder> ::= {LSBFirst MSBFFirst}
n/a	:SBUS<n>:UART:COUNT:ERRor? (see page 793)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNT:RESet (see page 794)	n/a	n/a
n/a	:SBUS<n>:UART:COUNT:RXFRAMES? (see page 795)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNT:TXFRAMES? (see page 796)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 797)	:SBUS<n>:UART:FRAMing? (see page 797)	<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 798)	:SBUS<n>:UART:PARity? (see page 798)	<parity> ::= {EVEN ODD NONE}
:SBUS<n>:UART:POLaritY <polarity> (see page 799)	:SBUS<n>:UART:POLaritY? (see page 799)	<polarity> ::= {HIGH LOW}
:SBUS<n>:UART:SOURce:RX <source> (see page 800)	:SBUS<n>:UART:SOURce:RX? (see page 800)	<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 32 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce: TX <source> (see page 801)	:SBUS<n>:UART:SOURce: TX? (see page 801)	<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 802)	:SBUS<n>:UART:TRIGger :BASE? (see page 802)	<base> ::= {ASCii HEX}
:SBUS<n>:UART:TRIGger :BURSt <value> (see page 803)	:SBUS<n>:UART:TRIGger :BURSt? (see page 803)	<value> ::= {OFF 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger :DATA <value> (see page 804)	:SBUS<n>:UART:TRIGger :DATA? (see page 804)	<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 805)	:SBUS<n>:UART:TRIGger :IDLE? (see page 805)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 806)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 806)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 807)	:SBUS<n>:UART:TRIGger :TYPE? (see page 807)	<value> ::= {RSTArt RSTOP RDATA RD1 RD0 RDX PARityerror TSTArt TSTOP TDATA TD1 TD0 TDX}
:SBUS<n>:UART:WIDTh <width> (see page 808)	:SBUS<n>:UART:WIDTh? (see page 808)	<width> ::= {5 6 7 8 9}

Table 33 :SBUS<n>:USB Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:USB:BASE <base> (see page 812)	:SBUS<n>:USB:BASE? (see page 812)	<base> ::= {ASCII BINARY DECIMAL HEX}
:SBUS<n>:USB:SOURce:D MINus <source> (see page 813)	:SBUS<n>:USB:SOURce:D MINus? (see page 813)	<source> ::= {CHANNEL<n>} 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>:USB:SOURce:D PLus <source> (see page 814)	:SBUS<n>:USB:SOURce:D PLus? (see page 814)	<source> ::= {CHANNEL<n>} 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>:USB:SOURce:D IFFerential <source> (see page 815)	:SBUS<n>:USB:SOURce:D IFFerential? (see page 815)	<source> ::= {CHANNEL<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:USB:SPEed <speed> (see page 816)	:SBUS<n>:USB:SPEed? (see page 816)	<speed> ::= {LOW FULL HIGH}
:SBUS<n>:USB:TRIGger <condition> (see page 817)	:SBUS<n>:USB:TRIGger? (see page 817)	<condition> ::= {SOP EOP ENTersuspend EXITsuspend RESet TOKen DATA HANDshake SPECIAL ALLerrors PIDerror CRC5error CRC16error GLITcherror STUFFerror SElerror}
:SBUS<n>:USB:TRIGger: ADDRess <string> (see page 818)	:SBUS<n>:USB:TRIGger: ADDRess? (see page 818)	<string> ::= "nnnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: CRC <string> (see page 819)	:SBUS<n>:USB:TRIGger: CRC? (see page 819)	<string> ::= "nnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}

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

Command	Query	Options and Query Returns
:SBUS<n>:USB:TRIGger: DATA <string> (see page 820)	:SBUS<n>:USB:TRIGger: DATA? (see page 820)	<string> ::= "nnnn..." where n ::= {0 1 X} <string> ::= "0xn..." where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: DATA:LENGth <value> (see page 821)	:SBUS<n>:USB:TRIGger: DATA:LENGth? (see page 821)	<length> ::= data length between 1-20
:SBUS<n>:USB:TRIGGER: ENDPoint <string> (see page 822)	:SBUS<n>:USB:TRIGger: ENDPoint? (see page 822)	<string> ::= "nnnn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: ET <string> (see page 823)	:SBUS<n>:USB:TRIGger: ET? (see page 823)	<string> ::= "nn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 2 3 X \$}
:SBUS<n>:USB:TRIGGER: FRAMe <string> (see page 824)	:SBUS<n>:USB:TRIGger: FRAMe? (see page 824)	<string> ::= "nnnnnnnnnnnn" where n ::= {0 1 X} <string> ::= "0xnnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: HADDress <string> (see page 825)	:SBUS<n>:USB:TRIGger: HADDress? (see page 825)	<string> ::= "nnnnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: PCHeck <string> (see page 826)	:SBUS<n>:USB:TRIGger: PCHeck? (see page 826)	<string> ::= "nnnn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: PID:DATA <pid> (see page 827)	:SBUS<n>:USB:TRIGger: PID:DATA? (see page 827)	<pid> ::= {DATA0 DATA1 DATA2 MDATA}
:SBUS<n>:USB:TRIGGER: PID:HAndshake <pid> (see page 828)	:SBUS<n>:USB:TRIGger: PID:HAndshake? (see page 828)	<pid> ::= {ACK NAK STALL NYET}
:SBUS<n>:USB:TRIGGER: PID:SPECial <pid> (see page 829)	:SBUS<n>:USB:TRIGger: PID:SPECIAL? (see page 829)	<pid> ::= {PING PRE ERR SPLIt}
:SBUS<n>:USB:TRIGGER: PID:TOKen <pid> (see page 830)	:SBUS<n>:USB:TRIGger: PID:TOKen? (see page 830)	<pid> ::= {OUT IN SETup SOF}

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

Command	Query	Options and Query Returns
:SBUS<n>:USB:TRIGger: PORT <string> (see page 831)	:SBUS<n>:USB:TRIGger: PORT? (see page 831)	<string> ::= "nnnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: SC <string> (see page 832)	:SBUS<n>:USB:TRIGger: SC? (see page 832)	<string> ::= "n" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 X \$}
:SBUS<n>:USB:TRIGger: SEU <string> (see page 833)	:SBUS<n>:USB:TRIGger: SEU? (see page 833)	<string> ::= "nn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 2 3 X \$}

Table 34 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNT? (see page 837)	<count> ::= an integer count value
:SEARch:EVENT <event_number> (see page 838)	:SEARch:EVENT? (see page 838)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 839)	:SEARch:MODE? (see page 839)	<value> ::= {EDGE GLITch RUNT TRANSition SERial{1 2}}
:SEARch:STATE <value> (see page 840)	:SEARch:STATE? (see page 840)	<value> ::= {{0 OFF} {1 ON}}

Table 35 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 842)	:SEARch:EDGE:SLOPe? (see page 842)	<slope> ::= {POSitive NEGative EITHER}
:SEARch:EDGE:SOURce <source> (see page 843)	:SEARch:EDGE:SOURce? (see page 843)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 36 :SEARch:GLITch Commands Summary

Command	Query	Options and Query Returns
:SEARch:GLITch:GREater rthan <greater_than_time>[s uffix] (see page 845)	:SEARch:GLITch:GREater rthan? (see page 845)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:SEARch:GLITch:LESSth an <less_than_time>[suff ix] (see page 846)	:SEARch:GLITch:LESSth an? (see page 846)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:SEARch:GLITch:POLari ty <polarity> (see page 847)	:SEARch:GLITch:POLari ty? (see page 847)	<polarity> ::= {POSitive NEGative}
:SEARch:GLITch:QUALif ier <qualifier> (see page 848)	:SEARch:GLITch:QUALif ier? (see page 848)	<qualifier> ::= {GREaterthan LESSthan RANGe}
:SEARch:GLITch:RANGE <less_than_time>[suff ix], <greater_than_time>[s uffix] (see page 849)	:SEARch:GLITch:RANGE? (see page 849)	<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 850)	:SEARch:GLITch:SOURce ? (see page 850)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

Table 37 :SEARch:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARch:RUNT:POLarity <polarity> (see page 852)	:SEARch:RUNT:POLarity ? (see page 852)	<polarity> ::= {POSitive NEGative EITHer}
:SEARch:RUNT:QUALifie r <qualifier> (see page 853)	:SEARch:RUNT:QUALifie r? (see page 853)	<qualifier> ::= {GREaterthan LESSthan NONE}

Table 37 :SEARch:RUNT Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:RUNT:SOURce <source> (see page 854)	:SEARch:RUNT:SOURce? (see page 854)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:RUNT:TIME <time>[suffix] (see page 855)	:SEARch:RUNT:TIME? (see page 855)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

Table 38 :SEARch:TRANSition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANSition:QUALifier <qualifier> (see page 857)	:SEARch:TRANSition:QUALifier? (see page 857)	<qualifier> ::= {GREaterthan LESSthan}
:SEARch:TRANSition:SLOPe <slope> (see page 858)	:SEARch:TRANSition:SLOPe? (see page 858)	<slope> ::= {NEGative POSitive}
:SEARch:TRANSition:SOURce <source> (see page 859)	:SEARch:TRANSition:SOURce? (see page 859)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANSition:TIME <time>[suffix] (see page 860)	:SEARch:TRANSition:TIME? (see page 860)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

Table 39 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:LABEL <value> (see page 862)	:SEARch:SERial:A429:LABEL? (see page 862)	<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 863)	:SEARch:SERial:A429:MODE? (see page 863)	<condition> ::= {LABEL LBITS PERRor WERRor GERRor WGERRors ALLerrors}

4 Commands Quick Reference

Table 39 :SEARch:SERial:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:A429:PATtern:DATA <string> (see page 864)	:SEARch:SERial:A429:PATtern:DATA? (see page 864)	<string> ::= "nn...n" where n ::= {0 1}, length depends on FORMAT
:SEARch:SERial:A429:PATtern:SDI <string> (see page 865)	:SEARch:SERial:A429:PATtern:SDI? (see page 865)	<string> ::= "nn" where n ::= {0 1}, length always 2 bits
:SEARch:SERial:A429:PATtern:SSM <string> (see page 866)	:SEARch:SERial:A429:PATtern:SSM? (see page 866)	<string> ::= "nn" where n ::= {0 1}, length always 2 bits

Table 40 :SEARch:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:CAN:MODE <value> (see page 868)	:SEARch:SERial:CAN:MODE? (see page 868)	<value> ::= {DATA IDData IDEither IDRremote ALLerrors OVERload ERRor MESSage MSIGnal}
:SEARch:SERial:CAN:ATTern:DATA <string> (see page 870)	:SEARch:SERial:CAN:ATTern:DATA? (see page 870)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARch:SERial:CAN:ATTern:DATA:LENGTH <length> (see page 871)	:SEARch:SERial:CAN:ATTern:DATA:LENGTH? (see page 871)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:CAN:ATTern:ID <string> (see page 872)	:SEARch:SERial:CAN:ATTern:ID? (see page 872)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARch:SERial:CAN:ATTern:ID:MODE <value> (see page 873)	:SEARch:SERial:CAN:ATTern:ID:MODE? (see page 873)	<value> ::= {STANDARD EXTended}
:SEARch:SERial:CAN:SYMBolic:MESSAge <name> (see page 874)	:SEARch:SERial:CAN:SYMBolic:MESSAge? (see page 874)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:SIGNAl <name> (see page 875)	:SEARch:SERial:CAN:SYMBolic:SIGNAl? (see page 875)	<name> ::= quoted ASCII string
:SEARch:SERial:CAN:SYMBolic:VALue <data> (see page 876)	:SEARch:SERial:CAN:SYMBolic:VALue? (see page 876)	<data> ::= value in NR3 format

Table 41 :SEARch:SERial:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:FLEXray:CYCLE <cycle> (see page 878)	:SEARch:SERial:FLEXray:CYCLE? (see page 878)	<cycle> ::= {ALL <cycle #>} <cycle #> ::= integer from 0-63
:SEARch:SERial:FLEXray:DATA <string> (see page 879)	:SEARch:SERial:FLEXray:DATA? (see page 879)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:FLEXray:DATA:LENGTH <length> (see page 880)	:SEARch:SERial:FLEXray:DATA:LENGTH? (see page 880)	<length> ::= integer from 1 to 12 in NR1 format
:SEARch:SERial:FLEXray:FRAMe <frame id> (see page 881)	:SEARch:SERial:FLEXray:FRAMe? (see page 881)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SEARch:SERial:FLEXray:MODE <value> (see page 882)	:SEARch:SERial:FLEXray:MODE? (see page 882)	<value> ::= {FRAMe CYCLe DATA HERRor FERRor AERRor}

Table 42 :SEARch:SERial:I2S Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:I2S:AUDio <audio_ch> (see page 884)	:SEARch:SERial:I2S:AUDio? (see page 884)	<audio_ch> ::= {RIGHT LEFT EITHer}
:SEARch:SERial:I2S:MODE <value> (see page 885)	:SEARch:SERial:I2S:MODE? (see page 885)	<value> ::= {EQUAL NOTequal LESSthan GREaterthan INRange OUTRange}
:SEARch:SERial:I2S:PATTern:DATA <string> (see page 886)	:SEARch:SERial:I2S:PATTern:DATA? (see page 886)	<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

Table 42 :SEARch:SERial:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:I2S:PA TTern:FORMAT <base> (see page 887)	:SEARch:SERial:I2S:PA TTern:FORMAT? (see page 887)	<base> ::= {BINary HEX DECimal}
:SEARch:SERial:I2S:RA NGe <lower>, <upper> (see page 888)	:SEARch:SERial:I2S:RA NGe? (see page 888)	<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 43 :SEARch:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:IIC:MO DE <value> (see page 890)	:SEARch:SERial:IIC:MO DE? (see page 890)	<value> ::= {READ7 WRITE7 NACKnowledge ANACK R7Data2 W7Data2 RESTart READEprom}
:SEARch:SERial:IIC:PA TTern:ADDRess <value> (see page 892)	:SEARch:SERial:IIC:PA TTern:ADDRess? (see page 892)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA <value> (see page 893)	:SEARch:SERial:IIC:PA TTern:DATA? (see page 893)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:PA TTern:DATA2 <value> (see page 894)	:SEARch:SERial:IIC:PA TTern:DATA2? (see page 894)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARch:SERial:IIC:QU ALifier <value> (see page 895)	:SEARch:SERial:IIC:QU ALifier? (see page 895)	<value> ::= {EQUAL NOTEQUAL LESSthan GREATERthan}

Table 44 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 897)	:SEARch:SERial:LIN:ID ? (see page 897)	<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:MO DE <value> (see page 898)	:SEARch:SERial:LIN:MO DE? (see page 898)	<value> ::= { ID DATA ERRor}
:SEARch:SERial:LIN:PA TTern:DATA <string> (see page 899)	:SEARch:SERial:LIN:PA TTern:DATA? (see page 899)	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:PA TTern:DATA:LENGTH <length> (see page 900)	:SEARch:SERial:LIN:PA TTern:DATA:LENGTH? (see page 900)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN:PA TTern:FORMAT <base> (see page 901)	:SEARch:SERial:LIN:PA TTern:FORMAT? (see page 901)	<base> ::= { HEX DECimal }

Table 45 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553: MODE <value> (see page 903)	:SEARch:SERial:M1553: MODE? (see page 903)	<value> ::= { DSTArt CSTArt RTA RTA11 PERRor SERRor MERRor }

4 Commands Quick Reference

Table 45 :SEARch:SERial:M1553 Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:M1553: PATtern:DATA <string> (see page 904)	:SEARch:SERial:M1553: PATtern:DATA? (see page 904)	<string> ::= "nn...n" where n ::= {0 1}
:SEARch:SERial:M1553: RTA <value> (see page 905)	:SEARch:SERial:M1553: RTA? (see page 905)	<value> ::= 5-bit integer in decimal, <hexadecimal>, <binary>, or <string> from 0-31 < hexadecimal > ::= #Hnn where n ::= {0,...,9 A,...,F} <binary> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F}

Table 46 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MO DE <value> (see page 907)	:SEARch:SERial:SPI:MO DE? (see page 907)	<value> ::= {MOSI MISO}
:SEARch:SERial:SPI:PA TTern:DATA <string> (see page 908)	:SEARch:SERial:SPI:PA TTern:DATA? (see page 908)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:SPI:PA TTern:WIDTH <width> (see page 909)	:SEARch:SERial:SPI:PA TTern:WIDTH? (see page 909)	<width> ::= integer from 1 to 10

Table 47 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:D ATA <value> (see page 911)	:SEARch:SERial:UART:D ATA? (see page 911)	<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:M ODE <value> (see page 912)	:SEARch:SERial:UART:M ODE? (see page 912)	<value> ::= {RDATA RD1 RD0 RDX TDATA TD1 TD0 TDX PARityerror AERRor}
:SEARch:SERial:UART:Q UALifier <value> (see page 913)	:SEARch:SERial:UART:Q UALifier? (see page 913)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}

Table 48 :SEARch:SERial:USB Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:USB:MO DE <condition> (see page 916)	:SEARch:SERial:USB:MO DE? (see page 916)	<event> ::= {TOKEN DATA HANDshake SPECial ALLerrors PIDerror CRC5error CRC16error GLITCHerror STUFFerror SELerror}
:SEARch:SERial:USB:AD DRess <string> (see page 917)	:SEARch:SERial:USB:AD DRess? (see page 917)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:CR C <string> (see page 918)	:SEARch:SERial:USB:CR C? (see page 918)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:DA TA <string> (see page 919)	:SEARch:SERial:USB:DA TA? (see page 919)	<string> ::= "0xn..." where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:DA TA:LENGth <value> (see page 920)	:SEARch:SERial:USB:DA TA:LENGth? (see page 920)	<length> ::= data length between 1-20
:SEARch:SERial:USB:EN DPoint <string> (see page 921)	:SEARch:SERial:USB:EN DPoint? (see page 921)	<string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}

Table 48 :SEARch:SERial:USB Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARch:SERial:USB:ET <string> (see page 922)	:SEARch:SERial:USB:ET ? (see page 922)	<string> ::= "0xn" where n ::= {0 1 2 3 X \$}
:SEARch:SERial:USB:FR AMe <string> (see page 923)	:SEARch:SERial:USB:FR AMe? (see page 923)	<string> ::= "0xnnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:HA DDress <string> (see page 924)	:SEARch:SERial:USB:HA DDress? (see page 924)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:PI D:DATA <pid> (see page 925)	:SEARch:SERial:USB:PI D:DATA? (see page 925)	<pid> ::= {DATA0 DATA1 DATA2 MDATA}
:SEARch:SERial:USB:PI D:HAndshake <pid> (see page 926)	:SEARch:SERial:USB:PI D:HAndshake? (see page 926)	<pid> ::= {ACK NAK STALL NYET}
:SEARch:SERial:USB:PI D:SPECial <pid> (see page 927)	:SEARch:SERial:USB:PI D:SPECial? (see page 927)	<pid> ::= {PING PRE ERR SPLIt}
:SEARch:SERial:USB:PI D:TOKen <pid> (see page 928)	:SEARch:SERial:USB:PI D:TOKen? (see page 928)	<pid> ::= {OUT IN SETup SOF}
:SEARch:SERial:USB:PO RT <string> (see page 929)	:SEARch:SERial:USB:PO RT? (see page 929)	<string> ::= "0xnn" where n ::= {0 0,...,9 A,...,F X \$}
:SEARch:SERial:USB:SC <string> (see page 930)	:SEARch:SERial:USB:SC ? (see page 930)	<string> ::= "0xn" where n ::= {0 1 X \$}
:SEARch:SERial:USB:SE U <string> (see page 931)	:SEARch:SERial:USB:SE U? (see page 931)	<string> ::= "0xn" where n ::= {0 1 2 3 X \$}

Table 49 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 935)	:SYSTem:DATE? (see page 935)	<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 936)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 937)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1191).
:SYSTem:LOCK <value> (see page 938)	:SYSTem:LOCK? (see page 938)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:PRESet (see page 939)	n/a	See :SYSTem:PRESet (see page 939)
:SYSTem:PROTection:LOCK <value> (see page 942)	:SYSTem:PROTection:LOCK? (see page 942)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:SETup <setup_data> (see page 943)	:SYSTem:SETup? (see page 943)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 945)	:SYSTem:TIME? (see page 945)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TOUCH {{1 ON} {0 OFF}} (see page 946)	:SYSTem:TOUCH? (see page 946)	{1 0}

Table 50 :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see page 949)	:TIMEbase:MODE? (see page 949)	<value> ::= {MAIN WINDOW XY ROLL}
:TIMEbase:POSIon <pos> (see page 950)	:TIMEbase:POSIon? (see page 950)	<pos> ::= time from the trigger event to the display reference point in NR3 format

4 Commands Quick Reference

Table 50 :TIMEbase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMEbase:RANGE <range_value> (see page 951)	:TIMEbase:RANGE? (see page 951)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMEbase:REFClock { {0 OFF} {1 ON} } (see page 952)	:TIMEbase:REFClock? (see page 952)	{0 1}
:TIMEbase:REFERENCE { LEFT CENTER RIGHT } (see page 953)	:TIMEbase:REFERENCE? (see page 953)	<return_value> ::= { LEFT CENTER RIGHT }
:TIMEbase:SCALE <scale_value> (see page 954)	:TIMEbase:SCALE? (see page 954)	<scale_value> ::= time/div in seconds in NR3 format
:TIMEbase:VERNier {{0 OFF} {1 ON}} (see page 955)	:TIMEbase:VERNier? (see page 955)	{0 1}
:TIMEbase:WINDOW:POSITION <pos> (see page 956)	:TIMEbase:WINDOW:POSITION? (see page 956)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMEbase:WINDOW:RANGE <range_value> (see page 957)	:TIMEbase:WINDOW:RANGE? (see page 957)	<range_value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALE <scale_value> (see page 958)	:TIMEbase:WINDOW:SCALE? (see page 958)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Table 51 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 962)	n/a	n/a
:TRIGger:HFReject {{0 OFF} {1 ON}} (see page 963)	:TRIGger:HFReject? (see page 963)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 964)	:TRIGger:HOLDoff? (see page 964)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:ASETup (see page 965)	n/a	n/a

Table 51 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:LEVel:HIGH <level>, <source> (see page 966)	:TRIGger:LEVel:HIGH? <source> (see page 966)	<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 967)	:TRIGger:LEVel:LOW? <source> (see page 967)	<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 968)	:TRIGger:MODE? (see page 968)	<mode> ::= {EDGE GLITch PATtern TV DELay EBURst OR RUNT SHOLD TRANSition SBUS{1 2}} <return_value> ::= {<mode> <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREject {{0 OFF} {1 ON}} (see page 969)	:TRIGger:NREject? (see page 969)	{0 1}
:TRIGger:SWEep <sweep> (see page 970)	:TRIGger:SWEep? (see page 970)	<sweep> ::= {AUTO NORMAL}

Table 52 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SL OPe <slope> (see page 972)	:TRIGger:DELay:ARM:SL OPe? (see page 972)	<slope> ::= {NEGative POSitive}
:TRIGger:DELay:ARM:SO URce <source> (see page 973)	:TRIGger:DELay:ARM:SO URce? (see page 973)	<source> ::= {CHANnel<n> DIGItal<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 52 :TRIGger:DELay Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:DELay:TDELay :TIME <time_value> (see page 974)	:TRIGger:DELay:TDELay :TIME? (see page 974)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGge r:COUNT <count> (see page 975)	:TRIGger:DELay:TRIGge r:COUNT? (see page 975)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGge r:SLOPe <slope> (see page 976)	:TRIGger:DELay:TRIGge r:SLOPe? (see page 976)	<slope> ::= {NEGative POSitive}
:TRIGger:DELay:TRIGge r:SOURce <source> (see page 977)	:TRIGger:DELay:TRIGge r:SOURce? (see page 977)	<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:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNT <count> (see page 979)	:TRIGger:EBURst:COUNT ? (see page 979)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 980)	:TRIGger:EBURst:IDLE? (see page 980)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 981)	:TRIGger:EBURst:SLOPe ? (see page 981)	<slope> ::= {NEGative POSitive}
:TRIGger:EBURst:SOURc e <source> (see page 982)	:TRIGger:EBURst:SOURc e? (see page 982)	<source> ::= {CHANnel<n> DIGItal<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

Table 54 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC DC LFReject} (see page 985)	:TRIGger[:EDGE]:COUPling? (see page 985)	{AC DC LFReject}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 986)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 986)	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 987)	:TRIGger[:EDGE]:REJect? (see page 987)	{OFF LFReject HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 988)	:TRIGger[:EDGE]:SLOPe? (see page 988)	<polarity> ::= {POSitive NEGative EITHer ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 989)	:TRIGger[:EDGE]:SOURce? (see page 989)	<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 55 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREAt erthan <greater_than_time>[s uffix] (see page 992)	:TRIGger:GLITch:GREAt erthan? (see page 992)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LESSt han <less_than_time>[suff ix] (see page 993)	:TRIGger:GLITch:LESSt han? (see page 993)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 994)	:TRIGger:GLITch:LEVel ? (see page 994)	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:POLAr ity <polarity> (see page 995)	:TRIGger:GLITch:POLAr ity? (see page 995)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALi fier <qualifier> (see page 996)	:TRIGger:GLITch:QUALi fier? (see page 996)	<qualifier> ::= {GREaterthan LESSthan RANGE}

Table 55 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 997)	:TRIGger:GLITch:RANGE? ? (see page 997)	<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 998)	:TRIGger:GLITch:SOURce? ? (see page 998)	<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:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1000)	:TRIGger:OR? ? (see page 1000)	<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 57 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern <string>[,<edge_source>,<edge>] (see page 1002)	:TRIGger:PATTern? (see page 1003)	<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:FORM at <base> (see page 1004)	:TRIGger:PATTern:FORM at? (see page 1004)	<base> ::= {ASCII HEX}
:TRIGger:PATTern:GREaterthan <greater_than_time>[suffix] (see page 1005)	:TRIGger:PATTern:GREaterthan? (see page 1005)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:PATTern:LESS than <less_than_time>[suffix] (see page 1006)	:TRIGger:PATTern:LESS than? (see page 1006)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:PATTern:QUALifier <qualifier> (see page 1007)	:TRIGger:PATTern:QUALifier? (see page 1007)	<qualifier> ::= {ENTERed GREaterthan LESSthan INRange OUTRange TIMeout}
:TRIGger:PATTern:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 1009)	:TRIGger:PATTern:RANGE? (see page 1009)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s ms us ns ps}

Table 58 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarit y <polarity> (see page 1011)	:TRIGger:RUNT:POLarit y? (see page 1011)	<polarity> ::= {POSitive NEGative EITHer}
:TRIGger:RUNT:QUALifi er <qualifier> (see page 1012)	:TRIGger:RUNT:QUALifi er? (see page 1012)	<qualifier> ::= {GREaterthan LESSthan NONE}
:TRIGger:RUNT:SOURce <source> (see page 1013)	:TRIGger:RUNT:SOURce? (see page 1013)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1014)	:TRIGger:RUNT:TIME? (see page 1014)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

Table 59 :TRIGger:SHOLD Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLD:SLOPe <slope> (see page 1016)	:TRIGger:SHOLD:SLOPe? (see page 1016)	<slope> ::= {NEGative POSitive}
:TRIGger:SHOLD:SOURce :CLOCk <source> (see page 1017)	:TRIGger:SHOLD:SOURce :CLOCk? (see page 1017)	<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 1018)	:TRIGger:SHOLD:SOURce :DATA? (see page 1018)	<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:H OLD <time>[suffix] (see page 1019)	:TRIGger:SHOLD:TIME:H OLD? (see page 1019)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:SHOLD:TIME:S ETup <time>[suffix] (see page 1020)	:TRIGger:SHOLD:TIME:S ETup? (see page 1020)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

Table 60 :TRIGger:TRANSition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANSition:QUALifier <qualifier> (see page 1022)	:TRIGger:TRANSition:QUALifier? (see page 1022)	<qualifier> ::= {GREaterthan LESSthan}
:TRIGger:TRANSition:SLOPe <slope> (see page 1023)	:TRIGger:TRANSition:SLOPe? (see page 1023)	<slope> ::= {NEGative POSitive}
:TRIGger:TRANSition:SOURce <source> (see page 1024)	:TRIGger:TRANSition:SOURce? (see page 1024)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANSition:TIME <time>[suffix] (see page 1025)	:TRIGger:TRANSition:TIME? (see page 1025)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

Table 61 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 1027)	:TRIGger:TV:LINE? (see page 1027)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 1028)	:TRIGger:TV:MODE? (see page 1028)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LINE LFIELD1 LFIELD2 LATERNATE}
:TRIGger:TV:POLarity <polarity> (see page 1029)	:TRIGger:TV:POLarity? (see page 1029)	<polarity> ::= {POSitive NEGative}
:TRIGger:TV:SOURce <source> (see page 1030)	:TRIGger:TV:SOURce? (see page 1030)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANDARD <standard> (see page 1031)	:TRIGger:TV:STANDARD? (see page 1031)	<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 1032)	:TRIGger:TV:UDTV:ENUMber? (see page 1032)	<count> ::= edge number in NR1 format

Table 61 :TRIGger:TV Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:TV:UDTV:HSYN c {{0 OFF} {1 ON}} (see page 1033)	:TRIGger:TV:UDTV:HSYN c? (see page 1033)	{0 1}
:TRIGger:TV:UDTV:HTIM e <time> (see page 1034)	:TRIGger:TV:UDTV:HTIM e? (see page 1034)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTH an <min_time> (see page 1035)	:TRIGger:TV:UDTV:PGTH an? (see page 1035)	<min_time> ::= seconds in NR3 format

Table 62 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE:SOURce <source> (see page 1037)	:TRIGger:ZONE:SOURce? (see page 1037)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:ZONE:STATe { {0 OFF} {1 ON}} (see page 1038)	:TRIGger:ZONE:STATe? (see page 1038)	{0 1}
:TRIGger:ZONE<n>:MODE <mode> (see page 1039)	:TRIGger:ZONE<n>:MODE ? (see page 1039)	<mode> ::= {INTersect NOTintersect} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:PLACement <width>, <height>, <x_center>, <y_center> (see page 1040)	:TRIGger:ZONE<n>:PLACement? (see page 1040)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <n> ::= 1-2 in NR1 format
n/a	:TRIGger:ZONE<n>:VALidity? (see page 1041)	<value> ::= {VALid INValid OSCreen} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:STATe { {0 OFF} {1 ON}} (see page 1042)	:TRIGger:ZONE<n>:STATe? (see page 1042)	{0 1} <n> ::= 1-2 in NR1 format

Table 63 :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see page 1051)	:WAVEform:BYTeorder? (see page 1051)	<value> ::= {LSBFFirst MSBFFirst}
n/a	:WAVEform:COUNT? (see page 1052)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see page 1053)	<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 1055)	:WAVEform:FORMAT? (see page 1055)	<value> ::= {WORD BYTE ASCII}
:WAVEform:POINTs <# points> (see page 1056)	:WAVEform:POINTs? (see page 1056)	<# 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 1058)	:WAVEform:POINTs:MODE ? (see page 1058)	<points_mode> ::= {NORMAl MAXimum RAW}

Table 63 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVEform:PREamble? (see page 1060)	<p><preamble_block> ::= <format NR1>, <type NR1>, <points NR1>, <count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>, <yincrement NR3>, <yorigin NR3>, <yreference NR1></p> <p><format> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for BYTE format • 1 for WORD format • 2 for ASCII format <p><type> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for NORMAL type • 1 for PEAK detect type • 3 for AVERAGE type • 4 for HRESolution type <p><count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</p>
n/a	:WAVEform:SEGmented:COUNT? (see page 1063)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see page 1064)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see page 1065)	:WAVEform:SOURce? (see page 1065)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> SBUS} for DSO models</p> <p><source> ::= {CHANnel<n> POD{1 2} BUS{1 2} FUNCtion<m> MATH<m> SBUS} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p>
:WAVEform:SOURce:SUBSource <subsource> (see page 1069)	:WAVEform:SOURce:SUBSource? (see page 1069)	<subsource> ::= {{SUB0 RX MOSI} {SUB1 TX MISO}}
n/a	:WAVEform:TYPE? (see page 1070)	<return_mode> ::= {NORM PEAK AVER HRES}
:WAVEform:UNSIGNED { {0 OFF} {1 ON} } (see page 1071)	:WAVEform:UNSIGNED? (see page 1071)	{0 1}

4 Commands Quick Reference

Table 63 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:VIEW <view> (see page 1072)	:WAVEform:VIEW? (see page 1072)	<view> ::= {MAIN}
n/a	:WAVEform:XINCrement? (see page 1073)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVEform:XORigin? (see page 1074)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVEform:XREFerence? (see page 1075)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVEform:YINCrement? (see page 1076)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVEform:YORigin? (see page 1077)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVEform:YREFerence? (see page 1078)	<return_value> ::= y-reference value in the current preamble in NR1 format

Table 64 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BY Teorder <order> (see page 1083)	:WGEN<w>:ARBitrary:BY Teorder? (see page 1083)	<order> ::= {MSBFFirst LSBFirst} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:DA TA {<binary> <value>, <value> ...} (see page 1084)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 or 2 in NR1 format
n/a	:WGEN<w>:ARBitrary:DA TA:ATTRibute:POINTs? (see page 1085)	<points> ::= number of points in NR1 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:DA TA:CLEar (see page 1086)	n/a	<w> ::= 1 or 2 in NR1 format

Table 64 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:DA TA:DAC {<binary> <value>, <value> ...} (see page 1087)	n/a	<p><binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format</p> <p><value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:ARBitrary:IN Terpolate {{0 OFF} {1 ON}} (see page 1088)	:WGEN<w>:ARBitrary:IN Terpolate? (see page 1088)	<p>{0 1}</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:ARBitrary:ST ORe <source> (see page 1089)	n/a	<p><source> ::= {CHANnel<n> WMEMory<r> FUNCTion<m> MATH<m>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FREQuency <frequency> (see page 1090)	:WGEN<w>:FREQuency? (see page 1090)	<p><frequency> ::= frequency in Hz in NR3 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion <signal> (see page 1091)	:WGEN<w>:FUNCtion? (see page 1093)	<p><signal> ::= {SINusoid SQUARE RAMP PULSe NOISE DC SINC EXPRise EXPFall CARDiac GAUSSian ARBitrary}</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:PUL Se:WIDTH <width> (see page 1094)	:WGEN<w>:FUNCtion:PUL Se:WIDTH? (see page 1094)	<p><width> ::= pulse width in seconds in NR3 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:RAM P:SYMMetry <percent> (see page 1095)	:WGEN<w>:FUNCtion:RAM P:SYMMetry? (see page 1095)	<p><percent> ::= symmetry percentage from 0% to 100% in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:SQU are:DCYCle <percent> (see page 1096)	:WGEN<w>:FUNCtion:SQU are:DCYCle? (see page 1096)	<p><percent> ::= duty cycle percentage from 20% to 80% in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>

Table 64 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:MODulation:A M:DEPTH <percent> (see page 1097)	:WGEN<w>:MODulation:A M:DEPTH? (see page 1097)	<percent> ::= AM depth percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:A M:FREQuency <frequency> (see page 1098)	:WGEN<w>:MODulation:A M:FREQuency? (see page 1098)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F M:DEViation <frequency> (see page 1099)	:WGEN<w>:MODulation:F M:DEViation? (see page 1099)	<frequency> ::= frequency deviation in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F M:FREQuency <frequency> (see page 1100)	:WGEN<w>:MODulation:F M:FREQuency? (see page 1100)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F SKey:FREQuency <percent> (see page 1101)	:WGEN<w>:MODulation:F SKey:FREQuency? (see page 1101)	<frequency> ::= hop frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F SKey:RATE <rate> (see page 1102)	:WGEN<w>:MODulation:F SKey:RATE? (see page 1102)	<rate> ::= FSK modulation rate in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F UNCTion <shape> (see page 1103)	:WGEN<w>:MODulation:F UNCTion? (see page 1103)	<shape> ::= {SINusoid SQUare RAMP} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F UNCTion:RAMP:SYMMetry <percent> (see page 1104)	:WGEN<w>:MODulation:F UNCTion:RAMP:SYMMetry ? (see page 1104)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:N OISe <percent> (see page 1105)	:WGEN<w>:MODulation:N OISe? (see page 1105)	<percent> ::= 0 to 100 <w> ::= 1 or 2 in NR1 format
:WGEN<w>:MODulation:S TATE {{0 OFF} {1 ON}} (see page 1106)	:WGEN<w>:MODulation:S TATE? (see page 1106)	{0 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:T YPE <type> (see page 1107)	:WGEN<w>:MODulation:T YPE? (see page 1107)	<type> ::= {AM FM FSK} <w> ::= 1 in NR1 format

Table 64 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:OUTPut {{0 OFF} {1 ON}} (see page 1109)	:WGEN<w>:OUTPut? (see page 1109)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1110)	:WGEN<w>:OUTPut:LOAD? (see page 1110)	<impedance> ::= {ONEMeg FIFTy} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:OUTPut:POLarity <polarity> (see page 1111)	:WGEN<w>:OUTPut:POLarity? (see page 1111)	<polarity> ::= {NORMAL INVerted} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:PERiod <period> (see page 1112)	:WGEN<w>:PERiod? (see page 1112)	<period> ::= period in seconds in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:RST (see page 1113)	n/a	<w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK {{0 OFF} {1 ON}} (see page 1114)	:WGEN<w>:TRACK? (see page 1114)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:AMPLitude {{0 OFF} {1 ON}} (see page 1115)	:WGEN<w>:TRACK:AMPLitude? (see page 1115)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:CSIGnality (see page 1116)	n/a	<w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:FREQuency {{0 OFF} {1 ON}} (see page 1117)	:WGEN<w>:TRACK:FREQuency? (see page 1117)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage <amplitude> (see page 1118)	:WGEN<w>:VOLTage? (see page 1118)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1119)	:WGEN<w>:VOLTage:HIGH? (see page 1119)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:LOW <low> (see page 1120)	:WGEN<w>:VOLTage:LOW? (see page 1120)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1121)	:WGEN<w>:VOLTage:OFFSet? (see page 1121)	<offset> ::= offset in volts in NR3 format <w> ::= 1 or 2 in NR1 format

Table 65 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1125)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay { {0 OFF} {1 ON}} (see page 1126)	:WMEMory<r>:DISPlay? (see page 1126)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 1}
:WMEMory<r>:LABEL <string> (see page 1127)	:WMEMory<r>:LABEL? (see page 1127)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 1128)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> FUNCTion<m> MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1129)	:WMEMory<r>:SKEW? (see page 1129)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFset <offset>[suffix] (see page 1130)	:WMEMory<r>:YOFFset? (see page 1130)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YRANGE <range>[suffix] (see page 1131)	:WMEMory<r>:YRANGE? (see page 1131)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YSCALE <scale>[suffix] (see page 1132)	:WMEMory<r>:YSCALE? (see page 1132)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V mV}

Syntax Elements

- "[Number Format](#)" on page 161
- "[<NL> \(Line Terminator\)](#)" on page 161
- "[\[\] \(Optional Syntax Terms\)](#)" on page 161
- "[{ } \(Braces\)](#)" on page 161
- "[::= \(Defined As\)](#)" on page 161
- "[<> \(Angle Brackets\)](#)" on page 162
- "[... \(Ellipsis\)](#)" on page 162
- "[n,,,p \(Value Ranges\)](#)" on page 162
- "[d \(Digits\)](#)" on page 162
- "[Quoted ASCII String](#)" on page 162
- "[Definite-Length Block Response Data](#)" on page 162

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> ::= indicates that <A> can be replaced by 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 168.

Table 66 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 169)	n/a	n/a
*ESE <mask> (see page 170)	*ESE? (see page 170)	<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 172)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 172)	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 175)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 176)	*OPC? (see page 176)	ASCII "1" is placed in the output queue when all pending device operations have completed.



5 Common (*) Commands

Table 66 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 177)	<pre> <return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <MSO>, <Xilinx FPGA Probe>, <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <FlexRay Serial>, <Power Measurements>, <RS-232/UART Serial>, <reserved>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>, <reserved>, <reserved>, <I2S Serial>, <reserved>, <Educator's Kit>, <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>, <reserved>, <reserved>, <reserved>, <reserved>, <Digital Voltmeter>, <Spectrum Visualizer>, <reserved>, <reserved>, <USB 2.0 Low/Full Speed>, <USB 2.0 High Speed> <All field> ::= {0 All} <reserved> ::= 0 <MSO> ::= {0 MSO} <Xilinx FPGA Probe> ::= {0 FPGAX} <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} <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} <Digital Voltmeter> ::= {0 DVM} <Spectrum Visualizer> ::= {0 SPECTRUM} <USB 2.0 Low/Full Speed> ::= {0 USF} <USB 2.0 High Speed> ::= {0 HUF} </pre>

Table 66 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns																																													
*RCL <value> (see page 179)	n/a	<value> ::= { 0 1 4 5 6 7 8 9 }																																													
*RST (see page 180)	n/a	See *RST (Reset) (see page 180)																																													
*SAV <value> (see page 183)	n/a	<value> ::= { 0 1 4 5 6 7 8 9 }																																													
*SRE <mask> (see page 184)	*SRE? (see page 185)	<p><mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>Enables</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>Operation Status Reg</td> </tr> <tr> <td>6</td> <td>64</td> <td>---</td> <td>(Not used.)</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td>Event Status Bit</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td>Message Available</td> </tr> <tr> <td>3</td> <td>8</td> <td>---</td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td>Message</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td>User</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td>Trigger</td> </tr> </tbody> </table>	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									
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 186)	<p><value> ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>"1"</th> <th>Indicates</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>OPER</td> <td>---</td> <td>Operation status condition occurred.</td> </tr> <tr> <td>6</td> <td>64</td> <td>RQS/</td> <td>MSS</td> <td>Instrument is requesting service.</td> </tr> <tr> <td>5</td> <td>32</td> <td>ESB</td> <td>---</td> <td>Enabled event status condition occurred.</td> </tr> <tr> <td>4</td> <td>16</td> <td>MAV</td> <td>---</td> <td>Message available.</td> </tr> <tr> <td>3</td> <td>8</td> <td>---</td> <td>---</td> <td>(Not used.)</td> </tr> <tr> <td>2</td> <td>4</td> <td>MSG</td> <td>---</td> <td>Message displayed.</td> </tr> <tr> <td>1</td> <td>2</td> <td>USR</td> <td>---</td> <td>User event condition occurred.</td> </tr> <tr> <td>0</td> <td>1</td> <td>TRG</td> <td>---</td> <td>A trigger occurred.</td> </tr> </tbody> </table>	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.
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 188)	n/a	n/a																																													
n/a	*TST? (see page 189)	<result> ::= 0 or non-zero value; an integer in NR1 format																																													
*WAI (see page 190)	n/a	n/a																																													

5 Common (*) Commands

Introduction to Common (*) Commands

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

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

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

NOTE

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

*CLS (Clear Status)



(see page 1232)

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 168
- "*STB (Read Status Byte)" on page 186
- "*ESE (Standard Event Status Enable)" on page 170
- "*ESR (Standard Event Status Register)" on page 172
- "*SRE (Service Request Enable)" on page 184
- ":SYSTem:ERRor" on page 937

***ESE (Standard Event Status Enable)**

C (see page 1232)

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.

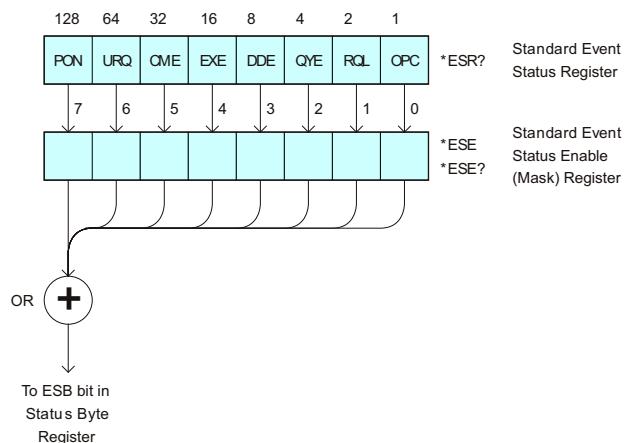


Table 67 Standard Event Status Enable (ESE)

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 168
- "["*ESR \(Standard Event Status Register\)](#)" on page 172
- "["*OPC \(Operation Complete\)](#)" on page 176
- "["*CLS \(Clear Status\)](#)" on page 169

***ESR (Standard Event Status Register)**

C (see page 1232)

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.

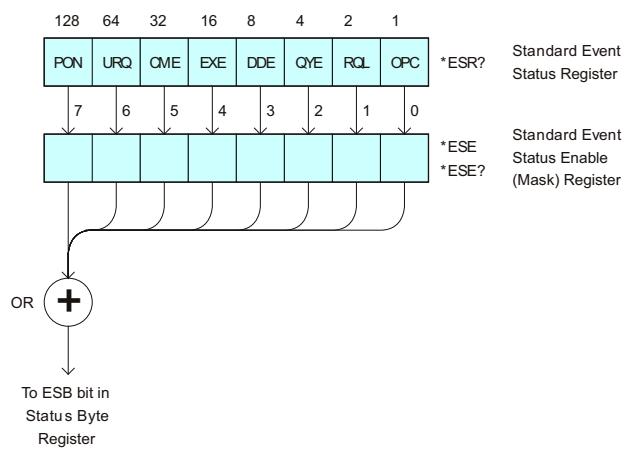


Table 68 Standard Event Status Register (ESR)

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 168
- "[*ESE \(Standard Event Status Enable\)](#)" on page 170
- "[*OPC \(Operation Complete\)](#)" on page 176
- "[*CLS \(Clear Status\)](#)" on page 169
- "[:SYSTem:ERRor](#)" on page 937

***IDN (Identification Number)**



(see [page 1232](#))

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 168
• "*OPT (Option Identification)" on page 177

*LRN (Learn Device Setup)



(see [page 1232](#))

Query Syntax

`*LRN?`

The `*LRN?` query result contains the current state of the instrument. This query is similar to the `:SYSTem:SETup?` (see [page 943](#)) 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 168
- "["*RCL \(Recall\)"](#) on page 179
- "["*SAV \(Save\)"](#) on page 183
- "[":SYSTem:SETup"](#) on page 943

***OPC (Operation Complete)**

 (see page 1232)

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 168
 - "[*ESE \(Standard Event Status Enable\)](#)" on page 170
 - "[*ESR \(Standard Event Status Register\)](#)" on page 172
 - "[*CLS \(Clear Status\)](#)" on page 169

***OPT (Option Identification)**

(see page 1232)

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>, <Xilinx FPGA Probe>,
    <Memory>, <Low Speed Serial>, <Automotive Serial>, <reserved>,
    <FlexRay Serial>, <Power Measurements>, <RS-232/UART Serial>,
    <reserved>, <Mask Test>, <reserved>, <Bandwidth>, <reserved>,
    <reserved>, <reserved>, <I2S Serial>, <reserved>, <Educator's Kit>,
    <Waveform Generator>, <MIL-1553/ARINC 429 Serial>, <Extended Video>,
    <reserved>, <reserved>, <reserved>, <Digital Voltmeter>,
    <Spectrum Visualizer>, <reserved>, <reserved>,
    <USB 2.0 Low/Full Speed>, <USB 2.0 High Speed>

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

<reserved> ::= 0

<MSO> ::= {0 | MSO}

<Xilinx FPGA Probe> ::= {0 | FPGAX}

<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}

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

<Bandwidth> ::= {0 | 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}

<Digital Voltmeter> ::= {0 | DVM}

<Spectrum Visualizer> ::= {0 | ASV}

<USB 2.0 Low/Full Speed> ::= {0 | USF}

```

5 Common (*) Commands

<USB 2.0 High Speed> ::= { 0 | U2H }

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

The *OPT? query returns the following:

- See Also**

 - ["Introduction to Common \(*\) Commands" on page 168](#)
 - ["IDN \(Identification Number\)" on page 174](#)

***RCL (Recall)** (see page 1232)**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 168
- "*SAV (Save)" on page 183

***RST (Reset)**

(see page 1232)

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 (1.4 V)

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive

5 Common (*) Commands

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also**
- "Introduction to Common (*) Commands" on page 168
 - ":SYSTem:PRESet" on page 939

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 41, “Programming Examples,”](#) starting on page 1241

***SAV (Save)**

(see page 1232)

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 168
- "*RCL (Recall)" on page 179

*SRE (Service Request Enable)

C (see page 1232)

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.

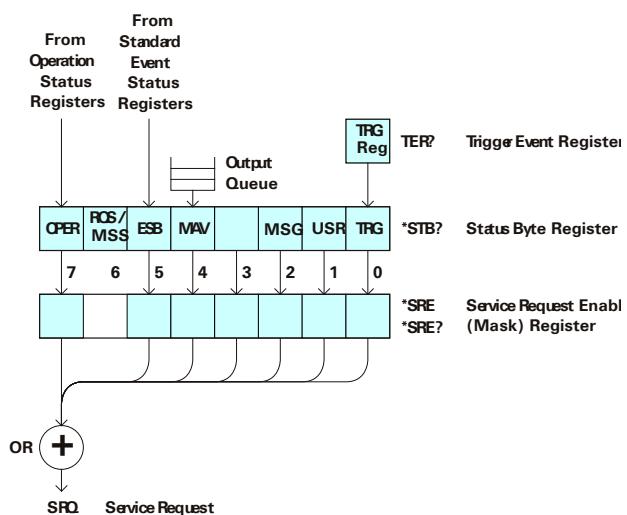


Table 69 Service Request Enable Register (SRE)

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 168
 - "["*STB \(Read Status Byte\)"](#) on page 186
 - "["*CLS \(Clear Status\)"](#) on page 169

*STB (Read Status Byte)

C (see page 1232)

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

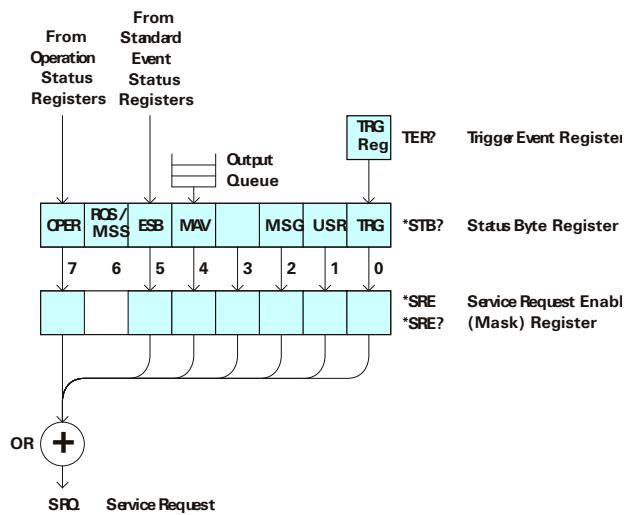


Table 70 Status Byte Register (STB)

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 168
- "["*SRE \(Service Request Enable\)](#)" on page 184

***TRG (Trigger)**



(see [page 1232](#))

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 168
 - "[":DIGITIZE](#)" on page 203
 - "[":RUN](#)" on page 224
 - "[":STOP](#)" on page 228

***TST (Self Test)**

(see page 1232)

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 168

***WAI (Wait To Continue)**



(see [page 1232](#))

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 168

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 194.

Table 71 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 195)	:ACTivity? (see page 195)	<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 196)	{0 1}; an integer in NR1 format
:AUToscale [<source>[,...<source>]] (see page 197)	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 199)	:AUToscale:AMODE? (see page 199)	<value> ::= {NORMal CURRent}
:AUToscale:CHANnels <value> (see page 200)	:AUToscale:CHANnels? (see page 200)	<value> ::= {ALL DISPlayed}
:AUToscale:FDEBug {{0 OFF} {1 ON}} (see page 201)	:AUToscale:FDEBug? (see page 201)	{0 1}



6 Root (:) Commands

Table 71 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:BLANK [<source>] (see page 202)	n/a	<p><source> ::= {CHANnel<n>} FUNCTION<m> MATH<m> SBUS{1 2} WMMemory<r>} for DSO models <source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCTION<m> MATH<m> SBUS{1 2} WMMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format</p>
:DIGItize [<source>[,...,<source>]] (see page 203)	n/a	<p><source> ::= {CHANnel<n>} FUNCTION<m> MATH<m> SBUS{1 2}} for DSO models <source> ::= {CHANnel<n>} DIGItal<d> POD{1 2} BUS{1 2} FUNCTION<m> MATH<m> SBUS{1 2}} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format</p>
:HWEenable <n> (see page 205)	:HWEenable? (see page 205)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDiti on? (see page 207)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister[:EVENT]? (see page 208)	<n> ::= 16-bit integer in NR1 format
:MTEenable <n> (see page 209)	:MTEenable? (see page 209)	<n> ::= 16-bit integer in NR1 format
n/a	:MTERregister[:EVENT]? (see page 211)	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see page 213)	:OPEE? (see page 214)	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDiti on? (see page 215)	<n> ::= 15-bit integer in NR1 format

Table 71 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:OPERegister[:EVENT] ? (see page 217)	<n> ::= 15-bit integer in NR1 format
:OVLenable <mask> (see page 219)	:OVLenable? (see page 220)	<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 221)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINT [<options>] (see page 223)	n/a	<options> ::= [<print option>] [,...,<print option>] <print option> ::= {COLOR GRAYscale PRINTER0 BMP8bit BMP PNG NOFactors FACTors} <print option> can be repeated up to 5 times.
:RUN (see page 224)	n/a	n/a
n/a	:SERial (see page 225)	<return value> ::= unquoted string containing serial number
:SINGle (see page 226)	n/a	n/a
n/a	:STATus? <display> (see page 227)	{0 1} <display> ::= {CHANnel<n> DIGital<d> POD{1 2} BUS{1 2} FUNCTION<m> MATH<m> SBUS{1 2} WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format
:STOP (see page 228)	n/a	n/a

6 Root (:) Commands

Table 71 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 229)	{0 1}
:VIEW <source> (see page 230)	n/a	<source> ::= {CHANnel<n> FUNCTION<m> MATH<m> SBUS{1 2} WMEMORY<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> POD{1 2} BUS{1 2} FUNCTION<m> MATH<m> SBUS{1 2} WMEMORY<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format

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

:ACTivity

N (see [page 1232](#))

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 194
- "[":POD<n>:THReshold](#)" on page 555
- "[":DIGital<d>:THReshold](#)" on page 313

:AER (Arm Event Register)

(see page 1232)

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 194
- "[":OPEE \(Operation Status Enable Register\)"](#) on page 213
- "[":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 215
- "[":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 217
- "[":*STB \(Read Status Byte\)"](#) on page 186
- "[":*SRE \(Service Request Enable\)"](#) on page 184

:AUToscale

C (see page 1232)

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 200) 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.

6 Root (:) Commands

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

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 194
 - "[":AUToscale:CHANnels](#)" on page 200
 - "[":AUToscale:AMODE](#)" on page 199

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 41, “Programming Examples,”](#) starting on page 1241

:AUToscale:AMODE

N (see page 1232)

Command Syntax :AUToscale:AMODE <value>

<value> ::= {NORMAL | CURR}

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 CURR 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 194
 - "[":AUToscale"](#) on page 197
 - "[":AUToscale:CHANnels"](#) on page 200
 - "[":ACQuire:TYPE"](#) on page 244
 - "[":ACQuire:MODE"](#) on page 235

:AUToscale:CHANnels

N (see [page 1232](#))

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 194
 - "[":AUToscale"](#) on page 197
 - "[":AUToscale:AMODE"](#) on page 199
 - "[":VIEW"](#) on page 230
 - "[":BLANK"](#) on page 202

:AUToscale:FDEBug

N (see page 1232)

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 194

• ":AUToscale" on page 197

:BLANK

N (see page 1232)

Command Syntax

```
:BLANK [<source>]

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

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

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

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

<m> ::= 1-4 in NR1 format

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

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

NOTE

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

NOTE

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

See Also

- "Introduction to Root (:) Commands" on page 194
- ":DISPlay:CLEar" on page 321
- ":CHANnel<n>:DISPlay" on page 272
- ":DIGItal<d>:DISPlay" on page 309
- ":FUNCtion<m>:DISPlay" on page 353
- ":POD<n>:DISPlay" on page 553
- ":SBUS<n>:DISPlay" on page 660
- ":WMEMory<r>:DISPlay" on page 1126
- ":STATus" on page 227
- ":VIEW" on page 230

Example Code

- "Example Code" on page 230

:DIGItize

(see page 1232)

Command Syntax

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

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

           for the DSO models  

<source> ::= {CHANnel<n> | DIGItal<d> | POD{1 | 2}  

           | BUS{1 | 2} | FUNCtion<m> | MATH<m> | SBUS{1 | 2}}  

           for the MSO models  

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

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

<m> ::= 1-4 in NR1 format  

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

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

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

NOTE

The :DIGItize command is only executed when the :TIMEbase:MODE is MAIN or WINDOW.

NOTE

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

NOTE

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

See Also

- "Introduction to Root (: Commands" on page 194
- ":RUN" on page 224
- ":SINGle" on page 226
- ":STOP" on page 228
- ":TIMEbase:MODE" on page 949
- Chapter 7, "ACQuire Commands," starting on page 231
- Chapter 33, "WAVeform Commands," starting on page 1043

6 Root (:) Commands

Example Code

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

See complete example programs at: [Chapter 41](#), “Programming Examples,” starting on page 1241

:HWEenable (Hardware Event Enable Register)

N (see page 1232)

Command Syntax :HWEenable <mask>

<mask> ::= 16-bit integer

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

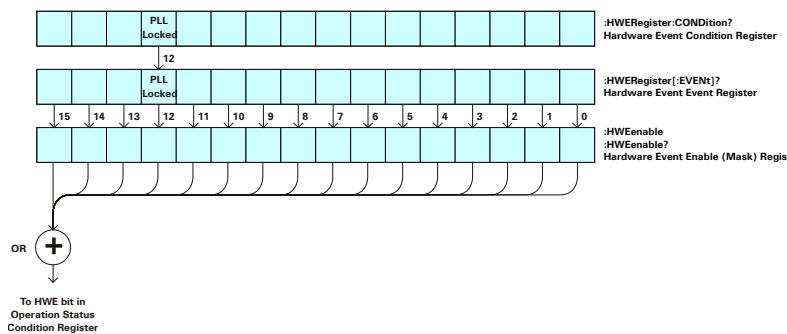


Table 72 Hardware Event Enable Register (HWEenable)

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

Query Syntax :HWEenable?

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(: \) Commands](#)" on page 194
- "[:AER \(Arm Event Register\)"](#) on page 196
- "[:CHANnel<n>:PROTection"](#) on page 282
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 217
- "[:OVLenable \(Overload Event Enable Register\)"](#) on page 219

6 Root (:) Commands

- "[:OVLRegister \(Overload Event Register\)](#)" on page 221
- "[*:STB \(Read Status Byte\)](#)" on page 186
- "[*:SRE \(Service Request Enable\)](#)" on page 184

:HWERegister:CONDition (Hardware Event Condition Register)

N (see page 1232)

Query Syntax :HWERegister:CONDition?

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

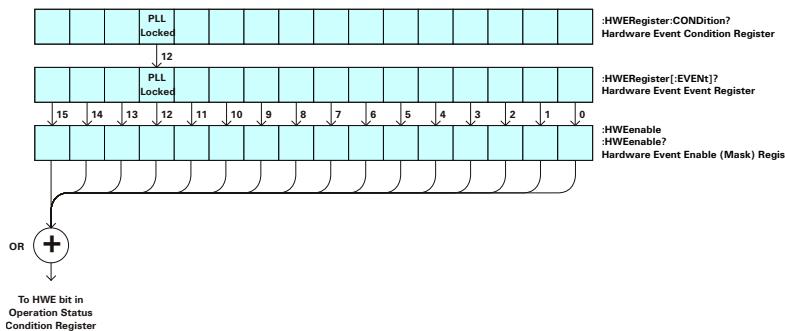


Table 73 Hardware Event Condition Register

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(: \) Commands](#)" on page 194
- "[:CHANnel<n>:PROTection](#)" on page 282
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 213
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 217
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 219
- "[:OVLRegister \(Overload Event Register\)](#)" on page 221
- "[*:STB \(Read Status Byte\)](#)" on page 186
- "[*:SRE \(Service Request Enable\)](#)" on page 184

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

N (see page 1232)

Query Syntax :HWERegister [:EVENT]?

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

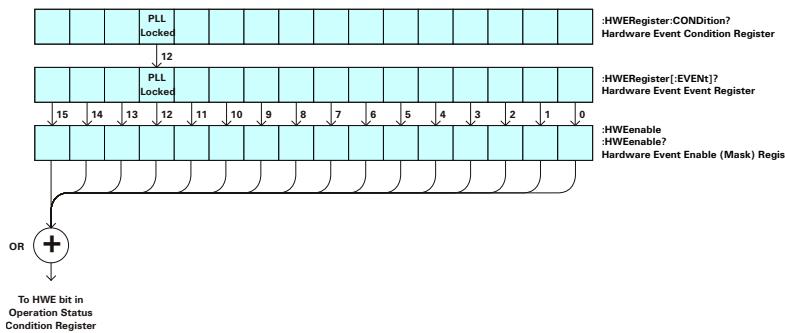


Table 74 Hardware Event Event Register

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 194
 - "[:CHANnel<n>:PROTection](#)" on page 282
 - "[:OPEE \(Operation Status Enable Register\)](#)" on page 213
 - "[:OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 215
 - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 219
 - "[:OVLRegister \(Overload Event Register\)](#)" on page 221
 - "[*:STB \(Read Status Byte\)](#)" on page 186
 - "[*:SRE \(Service Request Enable\)](#)" on page 184

:MTEenable (Mask Test Event Enable Register)

N (see page 1232)

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.

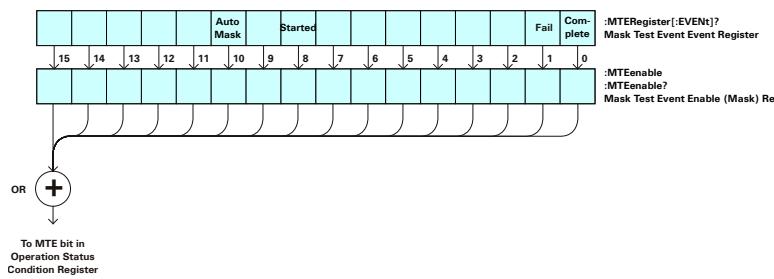


Table 75 Mask Test Event Enable Register (MTEenable)

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	Complete	Mask Test Complete	Mask test is complete.

Query Syntax :MTEenable?

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

Return Format <value><NL>
<value> ::= integer in NR1 format.

See Also • "[Introduction to Root \(: Commands](#)" on page 194
• "[:AER \(Arm Event Register\)](#)" on page 196

- "[:CHANnel<n>:PROTection](#)" on page 282
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 217
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 219
- "[:OVLRegister \(Overload Event Register\)](#)" on page 221
- "[:*STB \(Read Status Byte\)](#)" on page 186
- "[:*SRE \(Service Request Enable\)](#)" on page 184

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

N (see page 1232)

Query Syntax :MTERegister [:EVENT] ?

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

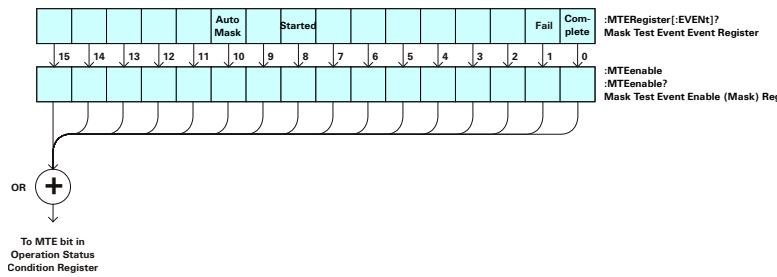


Table 76 Mask Test Event Event Register

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	Complete	Mask Test Complete	The mask test is complete.

Return Format <value><NL>

<value> ::= integer in NR1 format.

- See Also**
- "[Introduction to Root \(: \) Commands](#)" on page 194
 - "[:CHANnel<n>:PROTection](#)" on page 282
 - "[:OPEE \(Operation Status Enable Register\)](#)" on page 213
 - "[:OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 215
 - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 219
 - "[:OVLRegister \(Overload Event Register\)](#)" on page 221

6 Root (:) Commands

- ["*STB \(Read Status Byte\)" on page 186](#)
- ["*SRE \(Service Request Enable\)" on page 184](#)

:OPEE (Operation Status Enable Register)

C (see page 1232)

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.

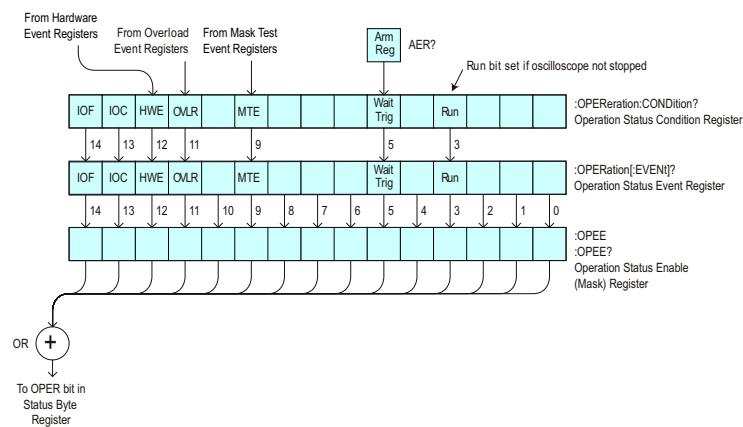


Table 77 Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
14	IOF	IO Operation Failed	Event when the IO operation fails.
13	IOC	IO Operation Complete	Event when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	Event when 50Ω input overload occurs.
10	---	---	(Not used.)
9	MTE	Mask Test Event	Event when mask test event occurs.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)
3	Run	Running	Event when the oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Query Syntax :OPEE?

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

Return Format <value><NL>

<value> ::= integer in NR1 format.

See Also

- "[Introduction to Root \(:\) Commands](#)" on page 194
- "[":AER \(Arm Event Register\)](#)" on page 196
- "[":CHANnel<n>:PROTection](#)" on page 282
- "[":OPERegister\[:EVENT\]](#) (Operation Status Event Register)" on page 217
- "[":OVLenable \(Overload Event Enable Register\)](#)" on page 219
- "[":OVLRegister \(Overload Event Register\)](#)" on page 221
- "[":STB \(Read Status Byte\)](#)" on page 186
- "[":SRE \(Service Request Enable\)](#)" on page 184

:OPERegister:CONDition (Operation Status Condition Register)



(see page 1232)

Query Syntax

:OPERegister:CONDition?

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

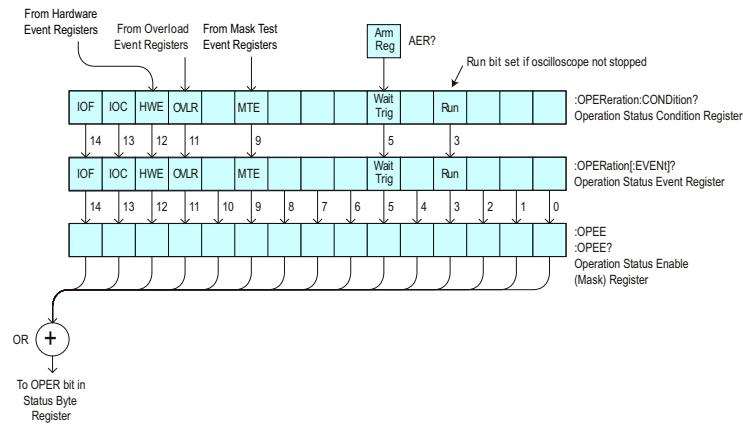


Table 78 Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14	IOF	IO Operation Failed	Event when the IO operation fails.
13	IOC	IO Operation Complete	Event when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)
9	MTE	Mask Test Event	A mask test event has occurred.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(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 194
 - "[":CHANnel<n>:PROTection](#)" on page 282
 - "[":OPEE \(Operation Status Enable Register\)"](#) on page 213
 - "[":OPERegister\[:EVENT\]](#) (Operation Status Event Register)" on page 217
 - "[":OVLenable \(Overload Event Enable Register\)"](#) on page 219
 - "[":OVLRegister \(Overload Event Register\)"](#) on page 221
 - "[":*STB \(Read Status Byte\)"](#) on page 186
 - "[":*SRE \(Service Request Enable\)"](#) on page 184
 - "[":MTERegister\[:EVENT\]](#) (Mask Test Event Event Register)" on page 211
 - "[":MTEenable \(Mask Test Event Enable Register\)"](#) on page 209

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

C (see page 1232)

Query Syntax :OPERegister [:EVENT] ?

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

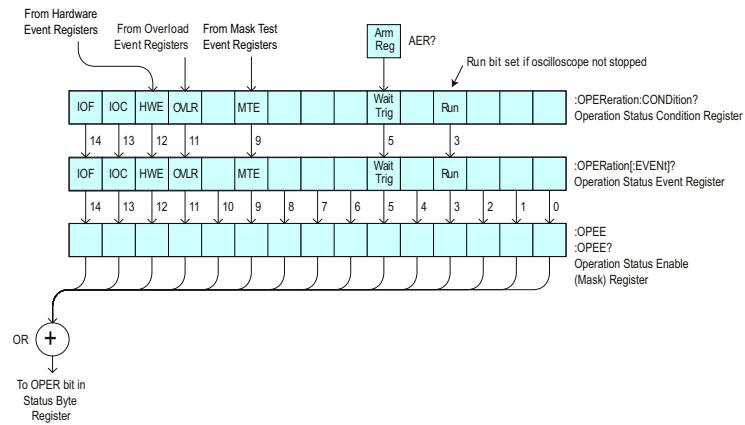


Table 79 Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
14	IOF	IO Operation Failed	Event when the IO operation fails.
13	IOC	IO Operation Complete	Event when the IO operation completes.
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	A 50Ω input overload has occurred.
10	---	---	(Not used.)
9	MTE	Mask Test Event	A mask test event has occurred.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(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 194
 - "[":CHANnel<n>:PROTection](#)" on page 282
 - "[":OPEE \(Operation Status Enable Register\)](#)" on page 213
 - "[":OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 215
 - "[":OVLenable \(Overload Event Enable Register\)](#)" on page 219
 - "[":OVLRegister \(Overload Event Register\)](#)" on page 221
 - "[":*STB \(Read Status Byte\)](#)" on page 186
 - "[":*SRE \(Service Request Enable\)](#)" on page 184
 - "[":MTERegister\[:EVENT\] \(Mask Test Event Event Register\)](#)" on page 211
 - "[":MTEenable \(Mask Test Event Enable Register\)](#)" on page 209

:OVLenable (Overload Event Enable Register)

C (see page 1232)

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.

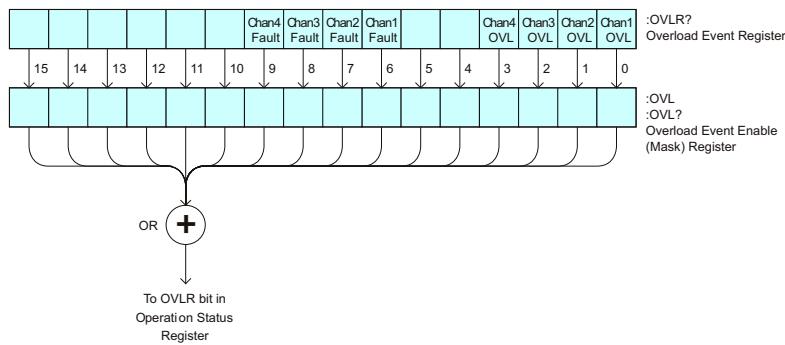


Table 80 Overload Event Enable Register (OVL)

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 80 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 194
 - "[:CHANnel<n>:PROtection](#)" on page 282
 - "[:OPEE \(Operation Status Enable Register\)](#)" on page 213
 - "[:OPERegister:CONDITION \(Operation Status Condition Register\)](#)" on page 215
 - "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 217
 - "[:OVLRegister \(Overload Event Register\)](#)" on page 221
 - "[*:STB \(Read Status Byte\)](#)" on page 186
 - "[*:SRE \(Service Request Enable\)](#)" on page 184

:OVLRegister (Overload Event Register)

C (see page 1232)

Query Syntax :OVLRegister?

The :OVLRegister query returns the overload protection value stored in the Overload Event Register (OCLR). 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.

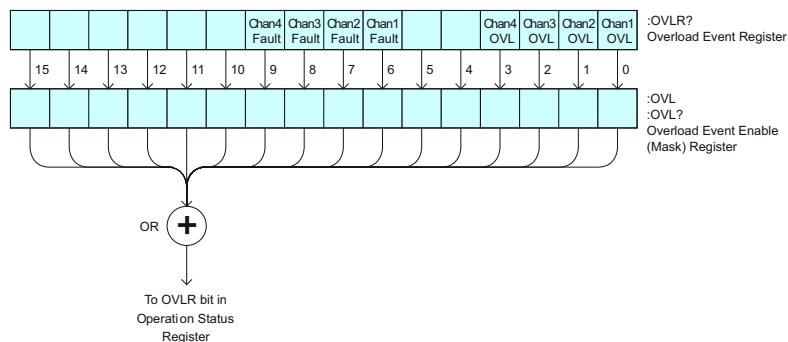


Table 81 Overload Event Register (OCLR)

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 194
 - "[:CHANnel<n>:PROTection](#)" on page 282
 - "[:OPEE \(Operation Status Enable Register\)](#)" on page 213
 - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 219
 - "[:*STB \(Read Status Byte\)](#)" on page 186
 - "[:*SRE \(Service Request Enable\)](#)" on page 184

:PRINt**C** (see page 1232)**Command Syntax**

```
:PRINt [<options>]  
<options> ::= [<print option>] [,...<print option>]  
<print option> ::= {COLOR | GRAYscale | PRINTER0 | PRINTER1 | 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 194
- "[Introduction to :HARDcopy Commands](#)" on page 376
- "[:HARDcopy:FACTors](#)" on page 379
- "[:HARDcopy:GRAYscale](#)" on page 1157
- "[:DISPlay:DATA](#)" on page 322

:RUN



(see [page 1232](#))

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 194
- "[":SINGle"](#) on page 226
- "[":STOP"](#) on page 228

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 41, “Programming Examples,”](#) starting on [page 1241](#)

:SERial

N (see page 1232)

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 194

:SINGle



(see page 1232)

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 194
- "[":RUN"](#) on page 224
- "[":STOP"](#) on page 228

:STATus

N (see page 1232)

Query Syntax

```
:STATus? <source>

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

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

<n> ::= 1 to (# analog channels) in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

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

NOTE

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

Return Format

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

See Also

- "Introduction to Root (: Commands" on page 194
- ":BLANK" on page 202
- ":CHANnel<n>:DISPlay" on page 272
- ":DIGItal<d>:DISPlay" on page 309
- ":FUNCtion<m>:DISPlay" on page 353
- ":POD<n>:DISPlay" on page 553
- ":SBUS<n>:DISPlay" on page 660
- ":WMEMory<r>:DISPlay" on page 1126
- ":VIEW" on page 230

:STOP



(see [page 1232](#))

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 194
- "[":RUN"](#) on page 224
- "[":SINGLe"](#) on page 226

Example Code

- "[Example Code](#)" on page 224

:TER (Trigger Event Register)



(see page 1232)

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 194
- "*SRE (Service Request Enable)" on page 184
- "*STB (Read Status Byte)" on page 186

:VIEW

N (see page 1232)

Command Syntax

```
:VIEW <source>

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

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

<n> ::= 1 to (# analog channels) in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

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

NOTE

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

See Also

- "Introduction to Root (:) Commands" on page 194
- ":BLANK" on page 202
- ":CHANnel<n>:DISPlay" on page 272
- ":DIGItal<d>:DISPlay" on page 309
- ":FUNCtion<m>:DISPlay" on page 353
- ":POD<n>:DISPlay" on page 553
- ":SBUS<n>:DISPlay" on page 660
- ":WMEMory<r>:DISPlay" on page 1126
- ":STATus" on page 227

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 41, “Programming Examples,”](#) starting on page 1241

7 :ACQuire Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 231.

Table 82 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:COMPLETE <complete> (see page 233)	:ACQuire:COMPLETE? (see page 233)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see page 234)	:ACQuire:COUNT? (see page 234)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:MODE <mode> (see page 235)	:ACQuire:MODE? (see page 235)	<mode> ::= {RTIMe ETIMe SEGmented}
n/a	:ACQuire:POINTS? (see page 236)	<# points> ::= an integer in NR1 format
:ACQuire:RSIGNAL <ref_signal_mode> (see page 237)	:ACQuire:RSIGNAL? (see page 237)	<ref_signal_mode> ::= {OFF OUT IN}
:ACQuire:SEGMENTED:ANALyze (see page 238)	n/a	n/a (with Option SGM)
:ACQuire:SEGMENTED:COUNT <count> (see page 239)	:ACQuire:SEGMENTED:COUNT? (see page 239)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
:ACQuire:SEGMENTED:INDEX <index> (see page 240)	:ACQuire:SEGMENTED:INDEX? (see page 240)	<index> ::= an integer from 1 to 1000 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATE? (see page 243)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 244)	:ACQuire:TYPE? (see page 244)	<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 1232)

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 :COMComplete 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 231
 - "[":ACQuire:TYPE](#)" on page 244
 - "[":DIGItize](#)" on page 203
 - "[":WAVeform:POINTs](#)" on page 1056

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 41](#), "Programming Examples," starting on page 1241

:ACQuire:COUNt

(see page 1232)

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 231
- ":ACQuire:TYPE" on page 244
- ":DIGitize" on page 203
- ":WAVEform:COUNt" on page 1052

:ACQuire:MODE

C (see page 1232)

Command Syntax :ACQuire:MODE <mode>

<mode> ::= {RTIMe | ETIMe | 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 ETIMe command sets the oscilloscope in equivalent time mode. Available with 4000 X-Series 1 GHz and 1.5 GHz bandwidth models only.
- 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 | ETIM | SEGM}

See Also

- "Introduction to :ACQuire Commands" on page 231
- ":ACQuire:TYPE" on page 244

:ACQuire:POINTS

(see page 1232)

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 231

• ":DIGITIZE" on page 203

• ":WAVeform:POINTS" on page 1056

:ACQuire:RSIGnal

N (see [page 1232](#))

Command Syntax :ACQuire:RSIGnal <ref_signal_mode>
 <ref_signal_mode> ::= {OFF | OUT | IN}

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

- The OFF mode disables the oscilloscope's 10 MHz REF BNC connector.
- The OUT mode is used to synchronize the timebase of two or more instruments.
- The IN mode is used to supply a sample clock to the oscilloscope. A 10 MHz square or sine wave signal is input to the BNC connector labeled 10 MHz REF. The amplitude must be from - 5 dBm to 17 dBm (356 mVpp to 4.48 Vpp).

CAUTION

Do not apply more than 20 dBm Max (6.32 Vpp Max) at the 10 MHz REF BNC connector on the rear panel or damage to the instrument may occur.

Query Syntax :ACQuire:RSIGnal?

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

Return Format <ref_signal_mode><NL>
 <ref_signal_mode> ::= {OFF | OUT | IN}

See Also • "[:TIMEbase:REFClock](#)" on [page 952](#)
 • The *Agilent InfiniiVision 4000 X Series Oscilloscopes User's Guide* for information on using the 10 MHz reference clock.

:ACQuire:SEGmented:ANALyze**N** (see page 1232)**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 235
- "[:ACQuire:SEGmented:COUNt](#)" on page 239
- "[Introduction to :ACQuire Commands](#)" on page 231

:ACQuire:SEGmented:COUNt

N (see page 1232)

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 235
- "[:DIGITIZE](#)" on page 203
- "[:SINGLE](#)" on page 226
- "[:RUN](#)" on page 224
- "[:WAVEFORM:SEGMENTED:COUNt](#)" on page 1063
- "[:ACQuire:SEGMENTED:ANALYZE](#)" on page 238
- "[Introduction to :ACQuire Commands](#)" on page 231

Example Code • "[Example Code](#)" on page 240

:ACQuire:SEGmented:INDEX

N (see page 1232)

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 235
- "[:ACQuire:SEGmented:COUNt](#)" on page 239
- "[:DIGITIZE](#)" on page 203
- "[:SINGLE](#)" on page 226
- "[:RUN](#)" on page 224
- "[:WAveform:SEGmented:COUNt](#)" on page 1063
- "[:WAveform:SEGmented:TTAG](#)" on page 1064
- "[:ACQuire:SEGmented:ANALyze](#)" on page 238
- "[Introduction to :ACQuire Commands](#)" on page 231

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

```

7 :ACQuire Commands

```
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 1232)

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 231
• ":ACQuire:POINts" on page 236

:ACQuire:TYPE

 (see page 1232)

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}
```

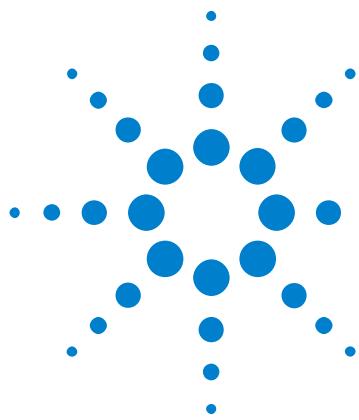
See Also • "[Introduction to :ACQuire Commands](#)" on page 231

- "[:ACQuire:COUNt](#)" on page 234
- "[:ACQuire:MODE](#)" on page 235
- "[:DIGitize](#)" on page 203
- "[:WAveform:FORMat](#)" on page 1055
- "[:WAveform:TYPE](#)" on page 1070
- "[:WAveform:PREamble](#)" on page 1060

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 41, “Programming Examples,”](#) starting on page 1241



8

:BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "[Introduction to :BUS<n> Commands](#)" on page 248.

Table 83 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 OFF} {1 ON}} (see page 249)	:BUS<n>:BIT<m>? (see page 249)	{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 250)	:BUS<n>:BITS? (see page 250)	<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 252)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 253)	:BUS<n>:DISPlay? (see page 253)	{0 1} <n> ::= 1 or 2; an integer in NR1 format



Table 83 :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:LABEL <string> (see page 254)	:BUS<n>:LABEL? (see page 254)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 255)	:BUS<n>:MASK? (see page 255)	<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 1232)

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 248
- ":BUS<n>:BITS" on page 250
- ":BUS<n>:CLEar" on page 252
- ":BUS<n>:DISPlay" on page 253
- ":BUS<n>:LABEL" on page 254
- ":BUS<n>:MASK" on page 255

Example Code

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

:BUS<n>:BITS

N (see page 1232)

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 248
- "[":BUS<n>:BIT<m>"](#) on page 249
- "[":BUS<n>:CLEar"](#) on page 252
- "[":BUS<n>:DISPlay"](#) on page 253
- "[":BUS<n>:LABEL"](#) on page 254
- "[":BUS<n>:MASK"](#) on page 255

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 1232)

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 248
- "[":BUS<n>:BIT<m>"](#) on page 249
- "[":BUS<n>:BITS"](#) on page 250
- "[":BUS<n>:DISPlay"](#) on page 253
- "[":BUS<n>:LABEL"](#) on page 254
- "[":BUS<n>:MASK"](#) on page 255

:BUS<n>:DISPlay

N (see page 1232)

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 248
 - "[:BUS<n>:BIT<m>](#)" on page 249
 - "[:BUS<n>:BITS](#)" on page 250
 - "[:BUS<n>:CLEar](#)" on page 252
 - "[:BUS<n>:LABEL](#)" on page 254
 - "[:BUS<n>:MASK](#)" on page 255

:BUS<n>:LABEL

N (see page 1232)

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 248
- "[:BUS<n>:BIT<m>](#)" on page 249
- "[:BUS<n>:BITS](#)" on page 250
- "[:BUS<n>:CLEAR](#)" on page 252
- "[:BUS<n>:DISPLAY](#)" on page 253
- "[:BUS<n>:MASK](#)" on page 255
- "[:CHANNEL<n>:LABEL](#)" on page 275
- "[:DISPLAY:LABELLIST](#)" on page 324
- "[:DIGITAL<d>:LABEL](#)" on page 310

Example Code

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

:BUS<n>:MASK

N (see page 1232)

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 248
 - "[:BUS<n>:BIT<m>](#)" on page 249
 - "[:BUS<n>:BITS](#)" on page 250
 - "[:BUS<n>:CLEar](#)" on page 252
 - "[:BUS<n>:DISPlay](#)" on page 253
 - "[:BUS<n>:LABel](#)" on page 254

9

:CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 258.

Table 84 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 259)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABEL <string> (see page 260)	:CALibrate:LABEL?	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 261)	:CALibrate:OUTPut?	<signal> ::= {TRIGgers MASK WAVEgen WGEN1 WGEN2} Note: WAVE and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.
n/a	:CALibrate:PROTected?	{PROTected UNPROtected}
:CALibrate:START (see page 263)	n/a	n/a
n/a	:CALibrate:STATus?	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPeratur e?	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME?	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format



- Introduction to :CALibrate Commands**
- The CALibrate subsystem provides utility commands for:
- Determining the state of the calibration factor protection switch (CAL PROTECT).
 - Saving and querying the calibration label string.
 - Reporting the calibration time and date.
 - Reporting changes in the temperature since the last calibration.
 - Starting the user calibration procedure.

:CALibrate:DATE

N (see page 1232)

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 258

:CALibrate:LABEL

N (see [page 1232](#))

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 258

:CALibrate:OUTPut

N (see page 1232)

Command Syntax :CALibrate:OUTPut <signal>

<signal> ::= {TRIGgers | MASK | WAVEgen | WGEN1 | WGEN2}

Note: WAVE and WGEN1 are equivalent.

Note: WGEN2 only available on models with 2 WaveGen outputs.

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, WGEN1, WGEN2 – waveform generator sync output signal.
This signal depends on the :WGEN<w>:FUNCtion setting:

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

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 | WGEN2}

See Also

- "Introduction to :CALibrate Commands" on page 258
- ":WGEN<w>:FUNCtion" on page 1091

:CALibrate:PROTected

N (see [page 1232](#))

Query Syntax :CALibrate:PROTected?

The :CALibrate:PROTected? query returns the rear-panel calibration protect (CAL PROTECT) button state. The value PROT 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 258

:CALibrate:STARt

N (see page 1232)

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 258
- ":CALibrate:PROTected" on page 262

:CALibrate:STATus

N (see page 1232)

Query Syntax :CALibrate:STATus?

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

Return Format

```
<return value><NL>
<return value> ::= <status_code>,<status_string>
<status_code> ::= an integer status code
<status_string> ::= an ASCII status string
```

The status codes and strings can be:

Status Code	Status String
+0	Calibrated
-1	Not Calibrated

See Also • "Introduction to :CALibrate Commands" on page 258

:CALibrate:TEMPerature

N (see page 1232)

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 258

:CALibrate:TIME

N (see [page 1232](#))

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 258

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 268.

Table 85 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit { {0 OFF} {1 ON} } (see page 270)	:CHANnel<n>:BWLimit? (see page 270)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 271)	:CHANnel<n>:COUPling? (see page 271)	<coupling> ::= {AC DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay { {0 OFF} {1 ON} } (see page 272)	:CHANnel<n>:DISPlay? (see page 272)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 273)	:CHANnel<n>:IMPedance? (see page 273)	<impedance> ::= {ONEMeg FIFTy} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert { {0 OFF} {1 ON} } (see page 274)	:CHANnel<n>:INVert? (see page 274)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABEL <string> (see page 275)	:CHANnel<n>:LABEL? (see page 275)	<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 276)	:CHANnel<n>:OFFSet? (see page 276)	<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 277)	:CHANnel<n>:PROBe? (see page 277)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format



10 :CHANnel<n> Commands

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

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:HEA D[:TYPE] <head_param> (see page 278)	:CHANnel<n>:PROBe:HEA D[:TYPE]? (see page 278)	<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 279)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKE W <skew_value> (see page 280)	:CHANnel<n>:PROBe:SKE W? (see page 280)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STY Pe <signal type> (see page 281)	:CHANnel<n>:PROBe:STY Pe? (see page 281)	<signal type> ::= {DIFFerential SINGle} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTectio n (see page 282)	:CHANnel<n>:PROTectio n? (see page 282)	{NORM TRIP} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 283)	:CHANnel<n>:RANGE? (see page 283)	<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 284)	:CHANnel<n>:SCALe? (see page 284)	<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 285)	:CHANnel<n>:UNITS? (see page 285)	<units> ::= {VOLT AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier { {0 OFF} {1 ON} } (see page 286)	:CHANnel<n>:VERNier? (see page 286)	{0 1} <n> ::= 1 to (# analog channels) in NR1 format

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

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

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

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

NOTE

The obsolete CHANnel subsystem is supported.

Reporting the Setup

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

Return Format

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

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

:CHANnel<n>:BWLimit



(see [page 1232](#))

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 268

:CHANnel<n>:COUPLing

C (see page 1232)

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 268

:CHANnel<n>:DISPlay

(see page 1232)

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 268
- "[:VIEW](#)" on page 230
- "[:BLANK](#)" on page 202
- "[:STATus](#)" on page 227
- "[:POD<n>:DISPlay](#)" on page 553
- "[:DIGital<d>:DISPlay](#)" on page 309

:CHANnel<n>:IMPedance

(see page 1232)

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 268

:CHANnel<n>:INVert

N (see [page 1232](#))

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 268

:CHANnel<n>:LABel

N (see page 1232)

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 268
- "[:DISPlay:LABel](#)" on page 323
- "[:DIGItal<d>:LABel](#)" on page 310
- "[:DISPlay:LABList](#)" on page 324
- "[:BUS<n>:LABel](#)" on page 254

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 41, “Programming Examples,”](#) starting on page 1241

:CHANnel<n>:OFFSet

(see page 1232)

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 268
- "[":CHANnel<n>:RANGE](#)" on page 283
- "[":CHANnel<n>:SCALE](#)" on page 284
- "[":CHANnel<n>:PROBe](#)" on page 277

:CHANnel<n>:PROBe

(see page 1232)

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 268
 - "[":CHANnel<n>:RANGE](#)" on page 283
 - "[":CHANnel<n>:SCALE](#)" on page 284
 - "[":CHANnel<n>:OFFSet](#)" on page 276

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 41, “Programming Examples,”](#) starting on page 1241

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

(see page 1232)

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 268
- ":CHANnel<n>:PROBe" on page 277
- ":CHANnel<n>:PROBe:ID" on page 279
- ":CHANnel<n>:PROBe:SKEW" on page 280
- ":CHANnel<n>:PROBe:STYPe" on page 281

:CHANnel<n>:PROBe:ID

(see page 1232)

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 268

:CHANnel<n>:PROBe:SKEW



(see page 1232)

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 268

:CHANnel<n>:PROBe:STYPe

(see page 1232)

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 268
- ":CHANnel<n>:OFFSet" on page 276

:CHANnel<n>:PROTection

N (see page 1232)

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 $1\text{ M}\Omega$.

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 273) 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 268
 - "[:CHANnel<n>:COUpling](#)" on page 271
 - "[:CHANnel<n>:IMPedance](#)" on page 273
 - "[:CHANnel<n>:PROBe](#)" on page 277

:CHANnel<n>:RANGE

(see page 1232)

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 268
 - "[":CHANnel<n>:SCALE](#)" on page 284
 - "[":CHANnel<n>:PROBe](#)" on page 277

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 41, “Programming Examples,”](#) starting on page 1241

:CHANnel<n>:SCALe

N (see page 1232)

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 268
 - "[":CHANnel<n>:RANGE](#)" on page 283
 - "[":CHANnel<n>:PROBe](#)" on page 277

:CHANnel<n>:UNITS

N (see page 1232)

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 268
 - "[":CHANnel<n>:RANGE](#)" on page 283
 - "[":CHANnel<n>:PROBe](#)" on page 277
 - "[":EXTernal:UNITS](#)" on page 341

:CHANnel<n>:VERNier

N (see [page 1232](#))

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 268

11 :COMPliance Commands

Control the optional DSOX4USBSQ USB 2.0 signal quality analysis feature.

Table 86 :COMPliance Commands Summary

Command	Query	Options and Query Returns
:COMPliance:USB:AUTosetup (see page 288)	n/a	n/a
:COMPliance:USB:HUBS<number> (see page 289)	:COMPliance:USB:HUBS? (see page 289)	<number> ::= 0-5 in NR1 format
:COMPliance:USB:RUN (see page 290)	n/a	n/a
:COMPliance:USB:SOURce:ADJacent <source> (see page 291)	:COMPliance:USB:SOURce:ADJacent? (see page 291)	<source> ::= {CHANnel<n>}
:COMPliance:USB:SOURce:DIFFerential <source> (see page 292)	:COMPliance:USB:SOURce:DIFFerential? (see page 292)	<source> ::= {CHANnel<n>}
:COMPliance:USB:SOURce:DMINus <source> (see page 293)	:COMPliance:USB:SOURce:DMINus? (see page 293)	<source> ::= {CHANnel<n>}
:COMPliance:USB:SOURce:DPLus <source> (see page 294)	:COMPliance:USB:SOURce:DPLus? (see page 294)	<source> ::= {CHANnel<n>}
:COMPliance:USB:TEST <test> (see page 295)	:COMPliance:USB:TEST? (see page 295)	<test> ::= {DHSS HHSS DLSS HLSS DFSS HFSS}
:COMPliance:USB:TEST:CONNECTION <connection> (see page 296)	:COMPliance:USB:TEST:CONNECTION? (see page 296)	<connection> ::= {SINGleended DIFFerential}
:COMPliance:USB:TEST:TYPE <type> (see page 297)	:COMPliance:USB:TEST:TYPE? (see page 297)	<type> ::= {NEARend FARend}



:COMPliance:USB:AUTosetup

N (see page 1232)

Command Syntax

:COMPliance:USB:AUTosetup

The :COMPliance:USB:AUTosetup command automatically sets up the oscilloscope for the selected signal quality test and USB compliance test packets. Automatically set are:

- Horizontal scale and delay.
- Analog input channel(s) scale and vertical offset.
- Trigger mode and trigger level.

The :COMPliance:USB:AUTosetup command causes the oscilloscope to run (that is, capture acquisitions that meet the trigger condition).

If you are testing the signal quality of real-world packets (instead of USB compliance test packets), you may need to adjust the auto setup.

Note that gating of measurements can only be done via the oscilloscope's horizontal controls. This can be done after the :COMPliance:USB:AUTosetup command and before the :COMPliance:USB:RUN command. The horizontal controls should be adjusted so that only the data to be tested is on screen when the :COMPliance:USB:RUN command is executed. This is normally one packet of data.

See Also

- "[:COMPliance:USB:TEST](#)" on page 295
- "[:COMPliance:USB:TEST:TYPE](#)" on page 297
- "[:COMPliance:USB:TEST:CONNection](#)" on page 296
- "[:COMPliance:USB:RUN](#)" on page 290

:COMPliance:USB:HUBS

N (see page 1232)

Command Syntax :COMPliance:USB:HUBS <number>
 <number> ::= 0-5 in NR1 format

The :COMPliance:USB:HUBS command specifies the number of internal hubs between the host and the test point.

When the Near-end test type is selected, you can specify 0 to 5 hubs.

When the Far-end test type is selected, the number of hubs is set to 0.

The specified number of hubs affects the test limits for the Sync results and the EOP Width results.

Query Syntax :COMPliance:USB:HUBS?

The :COMPliance:USB:HUBS? query returns the specified the number of internal hubs.

Return Format <number><NL>
 <number> ::= 0-5 in NR1 format

- See Also**
- "[:COMPliance:USB:TEST](#)" on page 295
 - "[:COMPliance:USB:TEST:TYPE](#)" on page 297
 - "[:COMPliance:USB:SOURce:DPLus](#)" on page 294
 - "[:COMPliance:USB:SOURce:DMINus](#)" on page 293
 - "[:COMPliance:USB:SOURce:ADJacent](#)" on page 291
 - "[:COMPliance:USB:SOURce:DIFFerential](#)" on page 292

:COMPliance:USB:RUN

N (see [page 1232](#))

Command Syntax :COMPliance:USB:RUN

The :COMPliance:USB:RUN command runs the selected signal quality test.

Please be patient as tests can take several minutes to complete.

When tests are run, the oscilloscope stops acquisitions if they are running, analyzes the data on screen, and then displays the results. The analyzed acquisition remains on screen and can be viewed by moving or dismissing the results dialog.

You can run tests when the oscilloscope is already stopped. This is useful in the case of embedded hosts (and other cases) where the device under test (DUT) may not easily be placed in a test mode, and you want to analyze already acquired data.

- See Also**
- "[:COMPliance:USB:TEST](#)" on page 295
 - "[:COMPliance:USB:AUTosetup](#)" on page 288
 - "[*OPC \(Operation Complete\)](#)" on page 176
 - "[:SAVE:COMPliance:USB\[:STARt\]](#)" on page 637

:COMPliance:USB:SOURce:ADJacent

N (see page 1232)

Command Syntax :COMPliance:USB:SOURce:ADJacent <source>

```
<source> ::= {CHANnel<n>}

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

The :COMPliance:USB:SOURce:ADJacent command specifies the analog input channel that is probing the adjacent D+ or D- signal.

When the Device Full Speed Signal Quality test is selected, the specified channel probes the adjacent D+ signal.

When the Device Low Speed Signal Quality test is selected, this specified channel probes the adjacent D- signal.

Query Syntax :COMPliance:USB:SOURce:ADJacent?

The :COMPliance:USB:SOURce:ADJacent? query returns the specified analog input channel.

Return Format <source><NL>

```
<source> ::= {CHAN<n>}
```

See Also • "[:COMPliance:USB:SOURce:DPLus](#)" on page 294

• "[:COMPliance:USB:SOURce:DMINus](#)" on page 293

:COMPliance:USB:SOURce:DIFFerential

N (see page 1232)

Command Syntax :COMPliance:USB:SOURce:DIFFerential <source>

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

The :COMPliance:USB:SOURce:DIFFerential command specifies the analog input channel whose differential probe is connected to the Hi-Speed signal to be tested.

Query Syntax :COMPliance:USB:SOURce:DIFFerential?

The :COMPliance:USB:SOURce:DIFFerential? query returns the specified analog input channel.

Return Format <source><NL>

```
<source> ::= {CHAN<n>}
```

See Also

- "[:COMPliance:USB:TEST](#)" on page 295
- "[:COMPliance:USB:TEST:CONNection](#)" on page 296

:COMPliance:USB:SOURce:DMINus

N (see page 1232)

Command Syntax :COMPliance:USB:SOURce:DMINus <source>

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

The :COMPliance:USB:SOURce:DMINus command specifies the analog input channel that is probing the D- signal.

On 4-channel oscilloscopes, you are forced to use different channel pairs for the D+ and D- signals. This provides the maximum sample rate. (Channels 1 and 2 are one pair and channels 3 and 4 are the other pair.)

Query Syntax :COMPliance:USB:SOURce:DMINus?

The :COMPliance:USB:SOURce:DMINus? query returns the specified analog input channel.

Return Format <source><NL>

```
<source> ::= {CHAN<n>}
```

See Also

- "[:COMPliance:USB:SOURce:DPLus](#)" on page 294
- "[:COMPliance:USB:SOURce:ADJacent](#)" on page 291

:COMPliance:USB:SOURce:DPLus

N (see page 1232)

Command Syntax :COMPliance:USB:SOURce:DPLus <source>

<source> ::= {CHANnel<n>}

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

The :COMPliance:USB:SOURce:DPLus command specifies the analog input channel that is probing the D+ signal.

On 4-channel oscilloscopes, you are forced to use different channel pairs for the D+ and D- signals. This provides the maximum sample rate. (Channels 1 and 2 are one pair and channels 3 and 4 are the other pair.)

Query Syntax :COMPliance:USB:SOURce:DPLus?

The :COMPliance:USB:SOURce:DPLus? query returns the specified analog input channel.

Return Format <source><NL>

<source> ::= {CHAN<n>}

See Also

- "[:COMPliance:USB:SOURce:DMINus](#)" on page 293
- "[:COMPliance:USB:SOURce:ADJacent](#)" on page 291

:COMPliance:USB:TEST

N (see page 1232)

Command Syntax

```
:COMPliance:USB:TEST <test>
<test> ::= {DHSS | HHSS | DLSS | HLSS | DFSS | HFSS}
```

The :COMPliance:USB:TEST command selects the type of signal quality test to perform:

- DHSS – Device Hi-Speed Signal Quality.
- HHSS – Host Hi-Speed Signal Quality.
- DLSS – Device Low Speed Signal Quality.
- HLSS – Host Low Speed Signal Quality.
- DFSS – Device Full Speed Signal Quality.
- HFSS – Host Full Speed Signal Quality.

For Hub upstream testing, select the Device tests.

For Hub downstream testing, select the Host tests.

Low and Full Speed tests require two or three single-ended probes.

Hi-Speed tests are available only on 1.5 GHz bandwidth oscilloscopes, and a differential probe or SMA cable connections are required.

Note that the N2750A Series InfiniiMode differential probes cannot be used with the Hi-Speed Signal Quality test fixtures because they have damping resistors and the USB test fixture also has damping resistors specifically designed for the E2678A socketed probe heads.

Query Syntax

```
:COMPliance:USB:TEST?
```

The :COMPliance:USB:TEST? query returns the selected signal quality test type.

Return Format

```
<test><NL>
<test> ::= {DHSS | HHSS | DLSS | HLSS | DFSS | HFSS}
```

See Also

- "[:COMPliance:USB:TEST:TYPE](#)" on page 297
- "[:COMPliance:USB:SOURce:DPLus](#)" on page 294
- "[:COMPliance:USB:SOURce:DMINus](#)" on page 293
- "[:COMPliance:USB:SOURce:ADJacent](#)" on page 291
- "[:COMPliance:USB:SOURce:DIFFerential](#)" on page 292

:COMPliance:USB:TEST:CONNnection

N (see page 1232)

Command Syntax `:COMPliance:USB:TEST:CONNnection <connection>`
`<connection> ::= {SINGleended | DIFFerential}`

When a Hi-Speed test has been selected, the :COMPliance:USB:TEST:CONNnection command specifies the test fixture connection type:

- DIFFerential – specifies that a differential probe is used to probe the signal under test.

In this case, use the :COMPliance:USB:SOURce:DIFFerential command to specify the analog input channel connected to the differential probe.

- SINGleended – specifies that two single-ended SMA cables are used to probe the signal under test.

In this case, use the :COMPliance:USB:SOURce:DPLus command to specify the analog input channel that is probing the D+ signal, and use the :COMPliance:USB:SOURce:DMINus command to specify the analog input channel that is probing the D- signal.

Query Syntax `:COMPliance:USB:TEST:CONNnection?`

The :COMPliance:USB:TEST:CONNnection? query returns the specified the test fixture connection type.

Return Format `<connection><NL>`
`<connection> ::= {SING | DIFF}`

See Also • "[:COMPliance:USB:SOURce:DIFFerential](#)" on page 292
 • "[:COMPliance:USB:SOURce:DPLus](#)" on page 294
 • "[:COMPliance:USB:SOURce:DMINus](#)" on page 293

:COMPliance:USB:TEST:TYPE

N (see [page 1232](#))

Command Syntax :COMPliance:USB:TEST:TYPE <type>
<type> ::= {NEARend | FARend}

When a Hi-Speed test has been selected, the :COMPliance:USB:TEST:TYPE command selects whether the test type is near-end or far-end.

Query Syntax :COMPliance:USB:TEST:TYPE?

The :COMPliance:USB:TEST:TYPE? query returns the selected test type.

Return Format <type><NL>
<type> ::= {NEAR | FAR}

See Also • "[:COMPliance:USB:TEST](#)" on page 295

12 :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 299.

Table 87 :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCTION <signal> (see page 300)	:DEMO:FUNCTION? (see page 303)	<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 USB NMONotonic}
:DEMO:FUNCTION:PHASE:PHASE <angle> (see page 304)	:DEMO:FUNCTION:PHASE:PHASE? (see page 304)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0 OFF} {1 ON}} (see page 305)	:DEMO:OUTPut? (see page 305)	{0 1}

Introduction to :DEMO Commands The :DEMO subsystem provides commands to output demonstration signals on the oscilloscope'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 1232)

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 | USB | NMONotonic}
```

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 304 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 μ s 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

12 :DEMO Commands

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
SPI	<p>Off</p> <p>Signals are internally routed to digital channels D6 through D9:</p> <ul style="list-style-type: none"> • 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	<p>Off</p> <p>Signals are internally routed to digital channels D7 through D9:</p> <ul style="list-style-type: none"> • 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
USB	USB Low Speed D+ signal, ~2.8 Vpp, 1.4 V offset	USB Low Speed D- signal, ~2.8 Vpp, 1.4 V offset
NMOnotonic	Digital pulse train with infrequent non-monotonic rising edges @ ~ 2.85 Vpp, 1.42 V offset	Off

Query Syntax :DEMO:FUNCTION?

The :DEMO:FUNCTION? query returns the currently selected demo signal type.

Return Format <signal><NL>

```
<signal> ::= {SIN | NOIS | PHAS | RING | SING1 | AM | CLK | GLIT  
| BURS | MSO | RUNT | TRAN | RFB | SHOL | LFS | FMB  
| ETE | CAN | LIN | UART | I2C | SPI | I2S | CANL  
| ARIN | FLEX | MIL | MIL2 | USB | NMON}
```

See Also • "Introduction to :DEMO Commands" on page 299

:DEMO:FUNCTION:PHASE:PHASE

N (see [page 1232](#))

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 299

• "[:DEMO:FUNCTION](#)" on page 300

:DEMO:OUTPut

N (see page 1232)

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 299
• ":DEMO:FUNCTION" on page 300

13 :DIGItal<d> Commands

Control all oscilloscope functions associated with individual digital channels. See "[Introduction to :DIGItal<d> Commands](#)" on page 307.

Table 88 :DIGItal<d> Commands Summary

Command	Query	Options and Query Returns
:DIGItal<d>:DISPlay { {0 OFF} {1 ON}} (see page 309)	:DIGItal<d>:DISPlay? (see page 309)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0 1}
:DIGItal<d>:LABel <string> (see page 310)	:DIGItal<d>:LABel? (see page 310)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGItal<d>:POSIon <position> (see page 311)	:DIGItal<d>:POSIon? (see page 311)	<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 312)	:DIGItal<d>:SIZE? (see page 312)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALL MEDIUM LARGe}
:DIGItal<d>:THReShold <value>[suffix] (see page 313)	:DIGItal<d>:THReShold? (see page 313)	<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



Agilent Technologies

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 1232)

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 307
 - ":POD<n>:DISPlay" on page 553
 - ":CHANnel<n>:DISPlay" on page 272
 - ":VIEW" on page 230
 - ":BLANK" on page 202
 - ":STATus" on page 227

:DIGItal<d>:LABel

N (see page 1232)

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 307
- ":CHANnel<n>:LABEL" on page 275
- ":DISPlay:LABList" on page 324
- ":BUS<n>:LABEL" on page 254

:DIGItal<d>:POSIon

N (see page 1232)

Command Syntax :DIGItal<d>:POSIon <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>:POSIon 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>:POSIon?

The :DIGItal<d>:POSIon? 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 307

:DIGItal<d>:SIZE

N (see [page 1232](#))

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 307

- "[":POD<n>:SIZE](#)" on page 554

- "[":DIGItal<d>:POStion](#)" on page 311

:DIGItal<d>:THreshold

N (see page 1232)

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 307
- ":POD<n>:THreshold" on page 555
- ":TRIGger[:EDGE]:LEVel" on page 986

14 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 316.

Table 89 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:ANNotation { {0 OFF} {1 ON}} (see page 317)	:DISPlay:ANNotation? (see page 317)	{0 1}
:DISPlay:ANNotation:B ACKground <mode> (see page 318)	:DISPlay:ANNotation:B ACKground? (see page 318)	<mode> ::= {OPAQue INVerted TRANsparent}
:DISPlay:ANNotation:C OLor <color> (see page 319)	:DISPlay:ANNotation:C OLor? (see page 319)	<color> ::= {CH1 CH2 CH3 CH4 DIG MATH REF MARKer WHITe RED}
:DISPlay:ANNotation:T EXT <string> (see page 320)	:DISPlay:ANNotation:T EXT? (see page 320)	<string> ::= quoted ASCII string (up to 254 characters)
:DISPlay:CLEar (see page 321)	n/a	n/a
n/a	:DISPlay:DATA? [<format>] [,] [<palett e>] (see page 322)	<format> ::= {BMP BMP8bit PNG} <palette> ::= {COLOR GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel { {0 OFF} {1 ON}} (see page 323)	:DISPlay:LABel? (see page 323)	{0 1}
:DISPlay:LABList <binary block> (see page 324)	:DISPlay:LABList? (see page 324)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:MENU <menu> (see page 325)	n/a	<menu> ::= {MASK MEASure SEGmented LISTer POWER}



Table 89 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:SIDebar <sidebar> (see page 326)	n/a	<sidebar> ::= {SUMmary CURSors MEASurements DVM NAVigate CONTrols}
:DISPlay:PERSistence <value> (see page 327)	:DISPlay:PERSistence? (see page 327)	<value> ::= {MINimum INFinite <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {{0 OFF} {1 ON}} (see page 328)	:DISPlay:VECTors? (see page 328)	{0 1} OFF or 0 can only be selected with the 1 GHz or 1.5 GHz bandwidth models

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 1232)

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 320
 - "[:DISPlay:ANNotation:COLOR](#)" on page 319
 - "[:DISPlay:ANNotation:BACKground](#)" on page 318
 - "[Introduction to :DISPlay Commands](#)" on page 316

:DISPLAY:ANNotation:BACKground

N (see page 1232)

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 317
 - "[:DISPLAY:ANNotation:TEXT](#)" on page 320
 - "[:DISPLAY:ANNotation:COLor](#)" on page 319
 - "[Introduction to :DISPLAY Commands](#)" on page 316

:DISPlay:ANNotation:COLor

N (see page 1232)

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 317
 - "[:DISPlay:ANNotation:TEXT](#)" on page 320
 - "[:DISPlay:ANNotation:BACKground](#)" on page 318
 - "[Introduction to :DISPlay Commands](#)" on page 316

:DISPlay:ANNotation:TEXT

N (see [page 1232](#))

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 317

• "[:DISPlay:ANNotation:COLor](#)" on page 319

• "[:DISPlay:ANNotation:BACKground](#)" on page 318

• "[Introduction to :DISPlay Commands](#)" on page 316

:DISPlay:CLEar

N (see page 1232)

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 316

:DISPLAY:DATA

N (see page 1232)

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 316
 - ":HARDcopy:INKSaver" on page 381
 - ":PRINT" on page 223
 - "*RCL (Recall)" on page 179
 - "*SAV (Save)" on page 183
 - ":VIEW" on page 230

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 for output.
Put #1, , byteData      ' Write data.
Close #1      ' Close file.
myScope.IO.Timeout = 5000
```

See complete example programs at: [Chapter 41, “Programming Examples,”](#) starting on page 1241

:DISPlay:LABel

N (see page 1232)

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 316
 - ":CHANnel<n>:LABEL" on page 275

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 41](#), "Programming Examples," starting on page 1241

:DISPLAY:LABList

N (see page 1232)

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 316
 - "[":DISPLAY:LABEL](#)" on page 323
 - "[":CHANNEL<n>:LABEL](#)" on page 275
 - "[":DIGITAL<d>:LABEL](#)" on page 310
 - "[":BUS<n>:LABEL](#)" on page 254

:DISPlay:MENU

N (see page 1232)

Command Syntax :DISPlay:MENU <menu>

<menu> ::= {MASK | MEASure | SEGmented | LISTer | POWer}

The :DISPlay:MENU command changes the front panel softkey menu.

:DISPLAY:SIDebar

N (see page 1232)

Command Syntax `:DISPLAY:SIDebar <sidebar>`

```
<sidebar> ::= {SUMMary | CURSors | MEASurements | DVM | NAVigate  
| CONTrols}
```

The `:DISPLAY:SIDebar` command specifies the sidebar dialog to display on the screen.

:DISPLAY:PERStance

N (see page 1232)

Command Syntax :DISPLAY:PERStance <value>

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

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

The :DISPLAY:PERStance 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:PERStance?

The :DISPLAY:PERStance? query returns the specified persistence value.

Return Format <value><NL>

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

See Also

- "Introduction to :DISPLAY Commands" on page 316
- ":DISPLAY:CLEar" on page 321

:DISPlay:VECTors

N (see [page 1232](#))

Command Syntax `:DISPlay:VECTors <vectors>`

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

The `:DISPlay:VECTors` command turns vector display on or off.

When vectors are turned on, the oscilloscope displays lines connecting sampled data points.

On the 1 GHz and 1.5 GHz bandwidth models, you can turn off vectors to view just waveform data points.

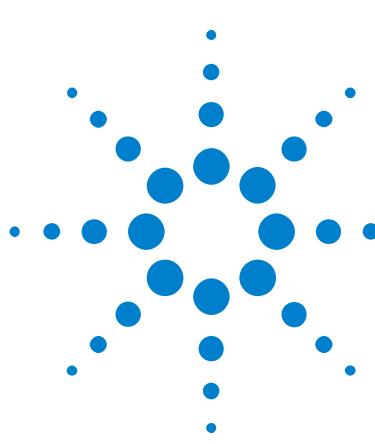
Query Syntax `:DISPlay:VECTors?`

The `:DISPlay:VECTors?` query returns the vectors setting.

Return Format `<vectors><NL>`

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

See Also • "Introduction to `:DISPlay Commands`" on [page 316](#)



15 :DVM Commands

When the optional DSOXDVM digital voltmeter analysis feature is licensed, these commands control the digital voltmeter (DVM) feature.

Table 90 :DVM Commands Summary

Command	Query	Options and Query Returns
:DVM:ARAnge {{0 OFF} {1 ON}} (see page 330)	:DVM:ARAnge? (see page 330)	{0 1}
n/a	:DVM:CURREnt? (see page 331)	<dvm_value> ::= floating-point number in NR3 format
:DVM:ENABLE {{0 OFF} {1 ON}} (see page 332)	:DVM:ENABLE? (see page 332)	{0 1}
n/a	:DVM:FREQuency? (see page 331)	<freq_value> ::= floating-point number in NR3 format
:DVM:MODE <mode> (see page 334)	:DVM:MODE? (see page 334)	<dvm_mode> ::= {ACRMs DC DCRMs FREQuency}
:DVM:SOURce <source> (see page 335)	:DVM:SOURce? (see page 335)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 in NR1 format



:DVM:ARAnge

N (see [page 1232](#))

Command Syntax :DVM:ARAnge <setting>

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

If the selected digital voltmeter (DVM) source channel is not used in oscilloscope triggering, the :DVM:ARAnge command turns the digital voltmeter's Auto Range capability on or off.

- When on, the DVM channel's vertical scale, vertical (ground level) position, and trigger (threshold voltage) level (used for the counter frequency measurement) are automatically adjusted.

The Auto Range capability overrides attempted adjustments of the channel's vertical scale and position.

- When off, you can adjust the channel's vertical scale and position normally.

Query Syntax :DVM:ARAnge?

The :DVM:ARAnge? query returns a flag indicating whether the digital voltmeter's Auto Range capability is on or off.

Return Format <setting><NL>

<setting> ::= {0 | 1}

- See Also**
- "[:DVM:SOURce](#)" on page 335
 - "[:DVM:ENABLE](#)" on page 332
 - "[:DVM:MODE](#)" on page 334

:DVM:CURRent

N (see page 1232)

Query Syntax :DVM:CURRent?

The :DVM:CURRent? query returns the displayed 3-digit DVM value based on the current mode.

Return Format <dvm_value><NL>

<dvm_value> ::= floating-point number in NR3 format

- See Also**
- "[:DVM:SOURce](#)" on page 335
 - "[:DVM:ENABLE](#)" on page 332
 - "[:DVM:MODE](#)" on page 334
 - "[:DVM:FREQuency](#)" on page 333

:DVM:ENABLE

N (see page 1232)

Command Syntax `:DVM:ENABLE <setting>`

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

The :DVM:ENABLE command turns the digital voltmeter (DVM) analysis feature on or off.

Query Syntax `:DVM:ENABLE?`

The :DVM:ENABLE? query returns a flag indicating whether the digital voltmeter (DVM) analysis feature is on or off.

Return Format `<setting><NL>`

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

- See Also**
- "[:DVM:SOURce](#)" on page 335
 - "[:DVM:MODE](#)" on page 334
 - "[:DVM:ARAnge](#)" on page 330

:DVM:FREQuency

N (see page 1232)

Query Syntax

:DVM:FREQuency?

The :DVM:FREQuency? query returns the displayed 5-digit frequency value that is displayed below the main DVM value.

Return Format

<freq_value><NL>
<freq_value> ::= floating-point number in NR3 format

See Also

- "[:DVM:SOURce](#)" on page 335
- "[:DVM:ENABLE](#)" on page 332
- "[:DVM:MODE](#)" on page 334
- "[:DVM:CURRent](#)" on page 331

:DVM:MODE

N (see page 1232)

Command Syntax :DVM:MODE <dvm_mode>

<dvm_mode> ::= {ACRMs | DC | DCRMs | FREQuency}

The :DVM:MODE command sets the digital voltmeter (DVM) mode:

- ACRMs – displays the root-mean-square value of the acquired data, with the DC component removed.
- DC – displays the DC value of the acquired data.
- DCRMs – displays the root-mean-square value of the acquired data.
- FREQuency – displays the frequency counter measurement.

Query Syntax :DVM:MODE?

The :DVM:MODE? query returns the selected DVM mode.

Return Format <dvm_mode><NL>

<dvm_mode> ::= {ACRM | DC | DCRM | FREQ}

- See Also**
- "[:DVM:ENABLE](#)" on page 332
 - "[:DVM:SOURce](#)" on page 335
 - "[:DVM:ARAnge](#)" on page 330
 - "[:DVM:CURRent](#)" on page 331
 - "[:DVM:FREQuency](#)" on page 333

:DVM:SOURce

N (see page 1232)

Command Syntax

```
:DVM:SOURce <source>
<source> ::= {CHANnel<n>}
<n> ::= 1-2 or 1-4 in NR1 format
```

The :DVM:SOURce command sets the select the analog channel on which digital voltmeter (DVM) measurements are made.

The selected channel does not have to be on (displaying a waveform) in order for DVM measurements to be made.

Query Syntax

```
:DVM:SOURce?
```

The :DVM:SOURce? query returns the selected DVM input source.

Return Format

```
<source><NL>
<source> ::= {CHAN<n>}
<n> ::= 1-2 or 1-4 in NR1 format
```

- See Also**
- "[:DVM:ENABLE](#)" on page 332
 - "[:DVM:MODE](#)" on page 334
 - "[:DVM:ARAnge](#)" on page 330
 - "[:DVM:CURRent](#)" on page 331
 - "[:DVM:FREQuency](#)" on page 333

16

:EXternal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXternal Trigger Commands" on page 337.

Table 91 :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLIMIT <bwlimit> (see page 338)	:EXternal:BWLIMIT? (see page 338)	<bwlimit> ::= {0 OFF}
:EXternal:PROBE <attenuation> (see page 339)	:EXternal:PROBE? (see page 339)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXternal:RANGE <range>[<suffix>] (see page 340)	:EXternal:RANGE? (see page 340)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXternal:UNITS <units> (see page 341)	:EXternal:UNITS? (see page 341)	<units> ::= {VOLT AMPere}

**Introduction to
:EXternal Trigger
Commands**

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

Reporting the Setup

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

Return Format

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

```
:EXT:BWL 0;RANG +8.0E+00;UNIT VOLT;PROB +1.000E+00
```



:EXternal:BWLimits

C (see page 1232)

Command Syntax `:EXTernal:BWLimits <bwlimits>`
`<bwlimits> ::= {0 | OFF}`

The :EXTernal:BWLimits 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:BWLimits?`

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

Return Format `<bwlimits><NL>`
`<bwlimits> ::= 0`

See Also • "Introduction to :EXternal Trigger Commands" on page 337
• "Introduction to :TRIGger Commands" on page 959
• ":TRIGger:HFReject" on page 963

:EXternal:PROBe

 (see page 1232)

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 337
- ":EXternal:RANGE" on page 340
- "Introduction to :TRIGger Commands" on page 959
- ":CHANnel<n>:PROBe" on page 277

:EXTernal:RANGE

 (see page 1232)

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 is either 1.6 V or 8 V.

The range is automatically recalculated when the external trigger probe attenuation factor is changed.

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 337
 - ":EXTernal:PROBe" on page 339
 - "Introduction to :TRIGger Commands" on page 959

:EXternal:UNITS

N (see page 1232)

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 337
- "Introduction to :TRIGger Commands" on page 959
- ":EXternal:RANGE" on page 340
- ":EXternal:PROBe" on page 339
- ":CHANnel<n>:UNITS" on page 285

:FUNCTION<m> Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCTION<m> Commands" on page 346.

Table 92 :FUNCTION<m> Commands Summary

Command	Query	Options and Query Returns
:FUNCTION<m>:AVERage:COUNT <count> (see page 347)	:FUNCTION<m>:AVERage:COUNT? (see page 347)	<count> ::= an integer from 2 to 65536 in NR1 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:BUS:CLOCK <source> (see page 348)	:FUNCTION<m>:BUS:CLOCK? (see page 348)	<source> ::= {CHANNEL<n> DIGITAL<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:BUS:SLOPE <slope> (see page 349)	:FUNCTION<m>:BUS:SLOPE? (see page 349)	<slope> ::= {NEGATIVE POSITIVE EITHER} <m> ::= 1-4 in NR1 format
:FUNCTION<m>:BUS:YINC rement <value> (see page 350)	:FUNCTION<m>:BUS:YINC rement? (see page 350)	<value> ::= value per bus code, in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:BUS:YORigin <value> (see page 351)	:FUNCTION<m>:BUS:YORigin? (see page 351)	<value> ::= value at bus code = 0, in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:BUS:YUNITS <units> (see page 352)	:FUNCTION<m>:BUS:YUNITS? (see page 352)	<units> ::= {VOLT AMPERE NONE} <m> ::= 1-4 in NR1 format
:FUNCTION<m>:DISPLAY {{0 OFF} {1 ON}} (see page 353)	:FUNCTION<m>:DISPLAY? (see page 353)	{0 1} <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:CENTER <frequency> (see page 354)	:FUNCTION<m>[:FFT]:CENTER? (see page 354)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz. <m> ::= 1-4 in NR1 format



17 :FUNCTION<m> Commands

Table 92 :FUNCTION<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION<m>[:FFT]:SPAN (see page 355)	:FUNCTION<m>[:FFT]:SPAN? (see page 355)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz. <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:VTPe <units> (see page 356)	:FUNCTION<m>[:FFT]:VTPe? (see page 356)	<units> ::= {DECibel VRMS} <m> ::= 1-4 in NR1 format
:FUNCTION<m>[:FFT]:WINDow <window> (see page 357)	:FUNCTION<m>[:FFT]:WINDow? (see page 357)	<window> ::= {RECTangular HANNing FLATtop BHARris} <m> ::= 1-4 in NR1 format
:FUNCTION<m>:FREQuency:HIGHpass <3dB_freq> (see page 358)	:FUNCTION<m>:FREQuency:HIGHpass? (see page 358)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:FREQuency:LOWPass <3dB_freq> (see page 359)	:FUNCTION<m>:FREQuency:LOWPass? (see page 359)	<3dB_freq> ::= 3dB cutoff frequency value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:INTegrat e:IOFFset <input_offset> (see page 360)	:FUNCTION<m>:INTegrat e:IOFFset? (see page 360)	<input_offset> ::= DC offset correction in NR3 format. <m> ::= 1-4 in NR1 format
:FUNCTION<m>:LINear:GAIN <value> (see page 361)	:FUNCTION<m>:LINear:GAIN? (see page 361)	<value> ::= 'A' in Ax + B, value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:LINear:OFFSet <value> (see page 362)	:FUNCTION<m>:LINear:OFFSet? (see page 362)	<value> ::= 'B' in Ax + B, value in NR3 format <m> ::= 1-4 in NR1 format
:FUNCTION<m>:OFFSET <offset> (see page 363)	:FUNCTION<m>:OFFSET? (see page 363)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function. <m> ::= 1-4 in NR1 format
:FUNCTION<m>:OPERatio n <operation> (see page 364)	:FUNCTION<m>:OPERatio n? (see page 365)	<operation> ::= {ADD SUBTract MULTIply DIVide INTegrate DIFF FFT SQRT MAGNify ABSolute SQUare LN LOG EXP TEN LOWPass HIGHpass AVERage LINEar TREnd BTIMing BSTate} <m> ::= 1-4 in NR1 format

Table 92 :FUNCTION<m> Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION<m>:RANGE <range> (see page 367)	:FUNCTION<m>:RANGE? (see page 367)	<p><range> ::= the full-scale vertical axis value in NR3 format.</p> <p>The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3.</p> <p>The range for the DIFF function is 80E-3 to 8.0E12 (depends on current sweep speed).</p> <p>The range for the FFT function is 8 to 800 dBV.</p> <p><m> ::= 1-4 in NR1 format</p>
:FUNCTION<m>:REFERenc e <level> (see page 368)	:FUNCTION<m>:REFERenc e? (see page 368)	<p><level> ::= the value at center screen in NR3 format.</p> <p>The range of legal values is +/-10 times the current sensitivity of the selected function.</p> <p><m> ::= 1-4 in NR1 format</p>
:FUNCTION<m>:SCALE <scale value>[<suffix>] (see page 369)	:FUNCTION<m>:SCALE? (see page 369)	<p><scale value> ::= integer in NR1 format</p> <p><suffix> ::= {V dB}</p> <p><m> ::= 1-4 in NR1 format</p>
:FUNCTION<m>:SOURcel <source> (see page 370)	:FUNCTION<m>:SOURcel? (see page 370)	<p><source> ::= {CHANnel<n> FUNCTION<c> MATH<c> BUS}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><c> ::= {1 2 3}, must be lower than <m></p> <p> ::= {1 2}</p> <p><m> ::= 1-4 in NR1 format</p>
:FUNCTION<m>:SOURce2 <source> (see page 372)	:FUNCTION<m>:SOURce2? (see page 372)	<p><source> ::= {CHANnel<n> NONE}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p>
:FUNCTION<m>:TRENd:ME ASurement <type> (see page 373)	:FUNCTION<m>:TRENd:ME ASurement? (see page 373)	<p><type> ::= {VAverage ACRMs VRATIO PERiod FREQuency PWIDth NWIDTH DUTYcycle RISeTime FALLtime}</p> <p><m> ::= 1-4 in NR1 format</p>

Introduction to :FUNCTION<m> Commands The FUNCTION subsystem controls the math functions in the oscilloscope. Four math functions are available – the <m> in :FUNCTION<m> can be from 1 to 4.

The math function operator, transform, filter, or visualization is selected using the :FUNCTION<m>:OPERation command. Depending on the selected operation, there may be other commands for specifying options for that operation. See "[:FUNCTION<m>:OPERation](#)" on page 364.

The SOURCE1, DISPLAY, RANGE, and OFFSET (or REFERENCE) commands apply to any function.

Reporting the Setup

Use :FUNCTION<m>? to query setup information for the FUNCTION subsystem.

Return Format

The following is a sample response from the :FUNCTION1? query. In this case, the query was issued following a *RST command.

```
:FUNC1:OPER ADD;DISP 0;SOUR1 CHAN1;SOUR2 CHAN2;RANG +8.00E+00;OFFS  
+0.0E+00
```

:FUNCTION<m>:AVERage:COUNt

N (see page 1232)

Command Syntax :FUNCTION<m>:AVERage:COUNt <count>

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

<m> ::= 1-4 in NR1 format

The :FUNCTION<m>:AVERage:COUNt command sets the number of waveforms to be averaged together.

The number of averages can be set from 2 to 65536 in increments of powers of 2.

Increasing the number of averages will increase resolution and reduce noise.

Query Syntax :FUNCTION<m>:AVERage:COUNt?

The :FUNCTION<m>:AVERage:COUNt? query returns the number of waveforms to be averaged together.

Return Format <count><NL>

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

See Also • "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:BUS:CLOCK

N (see page 1232)

Command Syntax `:FUNCTION<m>:BUS:CLOCK <source>`

`<m> ::= 1-4 in NR1 format`

`<source> ::= {DIGITAL<d>}`

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

The `:FUNCTION<m>:BUS:CLOCK` command selects the clock signal source for the Chart Logic Bus State operation.

Query Syntax `:FUNCTION<m>:BUS:CLOCK?`

The `:FUNCTION<m>:BUS:CLOCK` query returns the source selected for the clock signal.

Return Format `<source><NL>`

`<source> ::= {DIGITAL<d>}`

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

See Also • "`:FUNCTION<m>:OPERATION`" on page 364

:FUNCTION<m>:BUS:SLOPe

N (see page 1232)

Command Syntax :FUNCTION<m>:BUS:SLOPe <slope>

<m> ::= 1-4 in NRI format

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

The :FUNCTION<m>:BUS:SLOPe command specifies the clock signal edge for the Chart Logic Bus State operation.

Query Syntax :FUNCTION<m>:BUS:SLOPe?

The :FUNCTION<m>:BUS:SLOPe query returns the clock edge setting.

Return Format <slope><NL>

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

See Also • "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:BUS:YINCrement

N (see [page 1232](#))

Command Syntax :FUNCTION<m>:BUS:YINCrement <value>

<m> ::= 1-4 in NR1 format

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

The :FUNCTION<m>:BUS:YINCrement command specifies the value associated with each increment in Chart Logic Bus data.

Query Syntax :FUNCTION<m>:BUS:YINCrement?

The :FUNCTION<m>:BUS:YINCrement query returns the value associated with each increment in Chart Logic Bus data.

Return Format <value><NL>

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

See Also • [":FUNCTION<m>:OPERation" on page 364](#)

:FUNCTION<m>:BUS:YORigin

N (see page 1232)

Command Syntax :FUNCTION<m>:BUS:YORigin <value>

<m> ::= 1-4 in NR1 format

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

The :FUNCTION<m>:BUS:YORigin command specifies the value associated with Chart Logic Bus data equal to zero.

Query Syntax :FUNCTION<m>:BUS:YORigin?

The :FUNCTION<m>:BUS:YORigin query returns the value for associated with data equal to zero.

Return Format <value><NL>

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

See Also • "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:BUS:YUNits

N (see [page 1232](#))

Command Syntax `:FUNCTION<m>:BUS:YUNits <units>`

`<m> ::= 1-4 in NRI format`

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

The `:FUNCTION<m>:BUS:YUNits` command specifies the vertical units for the Chart Logic Bus operations.

Query Syntax `:FUNCTION<m>:BUS:YUNits?`

The `:FUNCTION<m>:BUS:YUNits` query returns the Chart Logic Bus vertical units.

Return Format `<units><NL>`

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

See Also • [":FUNCTION<m>:OPERation"](#) on page 364

:FUNCTION<m>:DISPLAY

N (see page 1232)

Command Syntax :FUNCTION<m>:DISPLAY <display>

<m> ::= 1-4 in NRI format

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

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

Query Syntax :FUNCTION<m>:DISPLAY?

The :FUNCTION<m>:DISPLAY? query returns whether the function display is on or off.

Return Format <display><NL>

<display> ::= {1 | 0}

- See Also**
- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
 - "[:VIEW](#)" on page 230
 - "[:BLANK](#)" on page 202
 - "[:STATUS](#)" on page 227

:FUNCTION<m>[:FFT]:CENTer

N (see page 1232)

Command Syntax `:FUNCTION<m>[:FFT]:CENTer <frequency>`

`<m>` ::= 1-4 in NR1 format

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

The `:FUNCTION<m>[:FFT]:CENTer` command sets the center frequency when FFT (Fast Fourier Transform) is selected.

Query Syntax `:FUNCTION<m>[:FFT]:CENTer?`

The `:FUNCTION<m>[:FFT]:CENTer?` query returns the current center frequency in Hertz.

Return Format `<frequency><NL>`

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

NOTE

After a *RST (Reset) or :AUToscale command, the values returned by the `:FUNCTION<m>[:FFT]:CENTer?` and `:FUNCTION<m>:SPAN?` queries depend on the current :TIMEbase:RANGE value. Once you change either the `:FUNCTION<m>[:FFT]:CENTer` or `:FUNCTION<m>:SPAN` value, they no longer track the :TIMEbase:RANGE value.

See Also

- "Introduction to :FUNCTION<m> Commands" on page 346
- ":FUNCTION<m>[:FFT]:SPAN" on page 355
- ":TIMEbase:RANGE" on page 951
- ":TIMEbase:SCALE" on page 954

:FUNCTION<m>[:FFT]:SPAN

N (see page 1232)

Command Syntax :FUNCTION<m>[:FFT]:SPAN

<m> ::= 1-4 in NR1 format

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

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

The :FUNCTION<m>[:FFT]:SPAN command sets the frequency span of the display (left graticule to right graticule) when FFT (Fast Fourier Transform) is selected.

Query Syntax :FUNCTION<m>[:FFT]:SPAN?

The :FUNCTION<m>[:FFT]:SPAN? query returns the current frequency span in Hertz.

NOTE

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

Return Format <NL>

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

See Also • "[Introduction to :FUNCTION<m> Commands](#)" on page 346

• "[:FUNCTION<m>\[:FFT\]:CENTer](#)" on page 354

• "[:TIMEbase:RANGE](#)" on page 951

• "[:TIMEbase:SCALe](#)" on page 954

:FUNCTION<m>[:FFT]:VTPe

N (see [page 1232](#))

Command Syntax `:FUNCTION<m>[:FFT]:VTPe <units>`

`<m> ::= 1-4 in NRI format`

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

The `:FUNCTION<m>[:FFT]:VTPe` command specifies FFT vertical units as DECibel or VRMS.

Query Syntax `:FUNCTION<m>[:FFT]:VTPe?`

The `:FUNCTION<m>[:FFT]:VTPe?` query returns the current FFT vertical units.

Return Format `<units><NL>`

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

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>[:FFT]:WINDOW

N (see page 1232)

Command Syntax :FUNCTION<m> [:FFT] :WINDOW <window>

<m> ::= 1-4 in NRI format

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

The :FUNCTION<m>[: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<m> [:FFT] :WINDOW?

The :FUNCTION<m>[:FFT]:WINDOW? query returns the value of the window selected for the FFT function.

Return Format <window><NL>

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

See Also • "Introduction to :FUNCTION<m> Commands" on page 346

:FUNCTION<m>:FREQuency:HIGHpass

N (see [page 1232](#))

Command Syntax `:FUNCTION<m>:FREQuency:HIGHpass <3dB_freq>`

`<m> ::= 1-4 in NR1 format`

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

The `:FUNCTION<m>:FREQuency:HIGHpass` command sets the high-pass filter's -3 dB cutoff frequency.

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

Query Syntax `:FUNCTION<m>:FREQuency:HIGHpass?`

The `:FUNCTION<m>:FREQuency:HIGHpass` query returns the high-pass filter's cutoff frequency.

Return Format `<3dB_freq><NL>`

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

See Also • [":FUNCTION<m>:OPERation"](#) on page 364

:FUNCTION<m>:FREQuency:LOWPass

N (see page 1232)

Command Syntax :FUNCTION<m>:FREQuency:LOWPass <3dB_freq>

<m> ::= 1-4 in NR1 format

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

The :FUNCTION<m>:FREQuency:LOWPass command sets the low-pass filter's -3 dB cutoff frequency.

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

Query Syntax :FUNCTION<m>:FREQuency:LOWPass?

The :FUNCTION<m>:FREQuency:LOWPass query returns the low-pass filter's cutoff frequency.

Return Format <3dB_freq><NL>

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

See Also • "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:INTegrate:IOFFset

N (see [page 1232](#))

Command Syntax `:FUNCTION<m>:INTegrate:IOFFset <input_offset>`

`<m> ::= 1-4 in NR1 format`

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

The `:FUNCTION<m>:INTegrate:IOFFset` command lets you enter a DC offset correction factor for the integrate math waveform input signal. This DC offset correction lets you level a "ramp"ed waveform.

Query Syntax `:FUNCTION<m>:INTegrate:IOFFset?`

The `:FUNCTION<m>:INTegrate:IOFFset?` query returns the current input offset value.

Return Format `<input_offset><NL>`

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

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:LINear:GAIN

N (see page 1232)

Command Syntax :FUNCTION<m>:LINear:GAIN <value>

<m> ::= 1-4 in NR1 format

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

The :FUNCTION<m>:LINear:GAIN command specifies the 'A' value in the Ax + B operation.

Query Syntax :FUNCTION<m>:LINear:GAIN?

The :FUNCTION<m>:LINear:GAIN query returns the gain value.

Return Format <value><NL>

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

See Also • "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION<m>:LINear:OFFSet

N (see [page 1232](#))

Command Syntax `:FUNCTION<m>:LINear:OFFSet <value>`

`<m> ::= 1-4 in NR1 format`

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

The `:FUNCTION<m>:LINear:OFFSet` command specifies the 'B' value in the $Ax + B$ operation.

Query Syntax `:FUNCTION<m>:LINear:OFFSet?`

The `:FUNCTION<m>:LINear:OFFSet` query returns the offset value.

Return Format `<value><NL>`

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

See Also • [":FUNCTION<m>:OPERation"](#) on page 364

:FUNCTION<m>:OFFSet

N (see page 1232)

Command Syntax :FUNCTION<m>:OFFSet <offset>

<m> ::= 1-4 in NR1 format

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

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

NOTE

The :FUNCTION<m>:OFFset command is equivalent to the :FUNCTION<m>:REFERENCE command.

Query Syntax

:FUNCTION<m>:OFFSet?

The :FUNCTION<m>:OFFSet? query outputs the current offset value for the selected function.

Return Format

<offset><NL>

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

See Also

- "Introduction to :FUNCTION<m> Commands" on page 346
- ":FUNCTION<m>:RANGE" on page 367
- ":FUNCTION<m>:REFERENCE" on page 368
- ":FUNCTION<m>:SCALE" on page 369

:FUNCTION<m>:OPERation

N (see page 1232)

Command Syntax :FUNCTION<m>:OPERation <operation>

<m> ::= 1-4 in NRI format

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

The :FUNCTION<m>:OPERation command sets the desired waveform math operator, transform, filter or visualization:

- Operators:

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTiply – Source1 * source2.
- DIVide – Source1 / source2.

Operators perform their function on two analog channel sources.

- Transforms:

- DIFF – Differentiate
- INTegrate – The INTegrate:IOFFset command lets you specify a DC offset correction factor.
- FFT – 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).
- LINear – Ax + B – The LINear commands set the gain (A) and offset (B) values for this function.
- SQUare
- SQRT – Square root
- ABSolute – Absolute Value
- LOG – Common Logarithm
- LN – Natural Logarithm
- EXP – Exponential (e^x)
- TEN – Base 10 exponential (10^x)

Transforms operate on a single analog channel source or on lower math functions.

- Filters:

- LOWPass – Low pass filter – The FREQuency:LOWPass command sets the -3 dB cutoff frequency.

- HIGHpass – High pass filter – The FREQuency:HIGHpass command sets the -3 dB cutoff frequency.
- AVERage – Averaged value – The AVERage:COUNt command specifies the number of averages.

Unlike acquisition averaging, the math averaging operator can be used to average the data on a single analog input channel or math function.

If acquisition averaging is also used, the analog input channel data is averaged and the math function averages it again. You can use both types of averaging to get a certain number of averages on all waveforms and an increased number of averages on a particular waveform.

Averages are calculated using a "decaying average" approximation, where:

```
next_average = current_average + (new_data - current_average)/N
```

Where N starts at 1 for the first acquisition and increments for each following acquisition until it reaches the selected number of averages, where it holds.

Filters operate on a single analog channel source or on a lower math function.

- Visualizations:
 - MAGNify – Operates on a single analog channel source or on a lower math function.
 - 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:YUNits 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:CLOCKS and BUS:SLOPe commands specify the clock source and edge.

Query Syntax :FUNCTION<m>:OPERation?

The :FUNCTION<m>:OPERation? query returns the current operation for the selected function.

Return Format <operation><NL>

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

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[":FUNCTION<m>:SOURCE1](#)" on page 370
- "[":FUNCTION<m>:SOURCE2](#)" on page 372
- "[":FUNCTION<m>:INTEGRATE:IOFFSET](#)" on page 360
- "[":FUNCTION<m>\[:FFT\]:SPAN](#)" on page 355
- "[":FUNCTION<m>\[:FFT\]:CENTER](#)" on page 354
- "[":FUNCTION<m>\[:FFT\]:VTYPe](#)" on page 356
- "[":FUNCTION<m>\[:FFT\]:WINDOW](#)" on page 357
- "[":FUNCTION<m>:LINEAR:GAIN](#)" on page 361
- "[":FUNCTION<m>:LINEAR:OFFSET](#)" on page 362
- "[":FUNCTION<m>:FREQUENCY:LOWPASS](#)" on page 359
- "[":FUNCTION<m>:FREQUENCY:HIGHPASS](#)" on page 358
- "[":FUNCTION<m>:AVERAGE:COUNT](#)" on page 347
- "[":FUNCTION<m>:TREND:MEASUREMENT](#)" on page 373
- "[":FUNCTION<m>:BUS:YINCrement](#)" on page 350
- "[":FUNCTION<m>:BUS:YORigin](#)" on page 351
- "[":FUNCTION<m>:BUS:YUNITS](#)" on page 352
- "[":FUNCTION<m>:BUS:CLOCK](#)" on page 348
- "[":FUNCTION<m>:BUS:SLOPE](#)" on page 349

:FUNCTION<m>:RANGE

N (see page 1232)

Command Syntax :FUNCTION<m>:RANGE <range>

<m> ::= 1-4 in NR1 format

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

The :FUNCTION<m>:RANGE command defines the full-scale vertical axis for the selected function.

Query Syntax :FUNCTION<m>:RANGE?

The :FUNCTION<m>:RANGE? query returns the current full-scale range value for the selected function.

Return Format <range><NL>

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

See Also • "Introduction to :FUNCTION<m> Commands" on page 346
• ":FUNCTION<m>:SCALE" on page 369

:FUNCTION<m>:REFERENCE

N (see [page 1232](#))

Command Syntax :FUNCTION<m>:REFERENCE <level>

<m> ::= 1-4 in NR1 format

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

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

NOTE

The FUNCTION:REFERENCE command is equivalent to the :FUNCTION<m>:OFFSET command.

Query Syntax

:FUNCTION<m>:REFERENCE?

The :FUNCTION<m>:REFERENCE? query outputs the current reference level value for the selected function.

Return Format

<level><NL>

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

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[":FUNCTION<m>:OFFSET](#)" on page 363
- "[":FUNCTION<m>:RANGE](#)" on page 367
- "[":FUNCTION<m>:SCALE](#)" on page 369

:FUNCTION<m>:SCALE

N (see page 1232)

Command Syntax :FUNCTION<m>:SCALE <scale value>[<suffix>]

<m> ::= 1-4 in NR1 format

<scale value> ::= integer in NR1 format

<suffix> ::= {V | dB}

The :FUNCTION<m>:SCALE command sets the vertical scale, or units per division, of the selected function. Legal values for the scale depend on the selected function.

Query Syntax :FUNCTION<m>:SCALE?

The :FUNCTION<m>:SCALE? query returns the current scale value for the selected function.

Return Format <scale value><NL>

<scale value> ::= integer in NR1 format

See Also

- "Introduction to :FUNCTION<m> Commands" on page 346
- ":FUNCTION<m>:RANGE" on page 367

:FUNCTION<m>:SOURce1

N (see page 1232)

Command Syntax

```
:FUNCTION<m>:SOURce1 <value>
<m> ::= 1-4 in NR1 format
<value> ::= {CHANnel<n> | FUNCTION<c> | MATH<c> | BUS<b>}
<n> ::= 1 to (# analog channels) in NR1 format
<c> ::= {1 | 2 | 3}, must be lower than <m>
<b> ::= {1 | 2}
```

The :FUNCTION<m>:SOURce1 command is used for any :FUNCTION<m>:OPERation selection. This command selects the first source for the operator math functions or the single source for the transform functions, filter functions, or visualization functions.

The FUNCTION<c> or MATH<c> parameters are available for the transform functions, filter functions, and the magnify visualization function (see "Introduction to :FUNCTION<m> Commands" on page 346) when <c> is lower than <m>.

In other words, higher math functions can operate on lower math functions when using operators other than the simple arithmetic operations (+, -, *, /). For example, if :FUNCTION1:OPERation is a SUBTract of CHANnel1 and CHANnel2, the :FUNCTION2:OPERation could be set up as a FFT operation on the FUNCTION1 source. These are called cascaded math functions.

To cascade math functions, select the lower math function using the :FUNCTION<m>:SOURce1 command.

When cascading math functions, to get the most accurate results, be sure to vertically scale lower math functions so that their waveforms take up the full screen without being clipped.

The BUS<m> parameter is available for the bus charting visualization functions.

NOTE

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

Query Syntax

```
:FUNCTION<m>:SOURce1?
```

The :FUNCTION<m>:SOURce1? query returns the current source1 for function operations.

Return Format

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

```
<value> ::= {CHAN<n> | FUNCtion<c> | MATH<c> | BUS<b>}  
<n> ::= 1 to (# analog channels) in NR1 format  
<c> ::= {1 | 2 | 3}, must be lower than <m>  
<b> ::= {1 | 2}
```

- See Also**
- "Introduction to :FUNCTION<m> Commands" on page 346
 - ":FUNCTION<m>:OPERation" on page 364

:FUNCTION<m>:SOURce2

N (see page 1232)

Command Syntax `:FUNCTION<m>:SOURce2 <value>`

```
<m> ::= 1-4 in NR1 format
<value> ::= {CHANnel<n> | NONE}
<n> ::= 1 to (# analog channels) in NR1 format
```

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

The :FUNCTION<m>:SOURce2 setting is not used for the transform functions, filter functions, or visualization functions (except when the measurement trend visualization's measurement requires two sources).

Query Syntax `:FUNCTION<m>:SOURce2?`

The :FUNCTION<m>:SOURce2? query returns the currently specified second source for math operations.

Return Format `<value><NL>`

```
<value> ::= {CHAN<n> | NONE}
<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "Introduction to :FUNCTION<m> Commands" on page 346
- ":FUNCTION<m>:OPERation" on page 364
- ":FUNCTION<m>:SOURce1" on page 370

:FUNCTION<m>:TRENd:MEASurement

N (see [page 1232](#))

Command Syntax :FUNCTION<m>:TRENd:MEASurement <type>
 <m> ::= 1-4 in NRI format
 <type> ::= {VAverage | ACRMs | VRATio | PERiod | FREQuency | PWIDth
 | NWIDth | DUTYcycle | RISetime | FALLtime}

The :FUNCTION<m>:TRENd:MEASurement command selects the measurement whose trend is shown in the math waveform.

Query Syntax :FUNCTION<m>:TRENd:MEASurement?

The :FUNCTION<m>: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<m>:OPERation](#)" on page 364

18 :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "Introduction to :HARDcopy Commands" on page 376.

Table 93 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 377)	:HARDcopy:AREA? (see page 377)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 378)	:HARDcopy:APRinter? (see page 378)	<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 379)	:HARDcopy:FACTors? (see page 379)	{0 1}
:HARDcopy:FFEd {{0 OFF} {1 ON}} (see page 380)	:HARDcopy:FFEd? (see page 380)	{0 1}
:HARDcopy:INKSaver { {0 OFF} {1 ON}} (see page 381)	:HARDcopy:INKSaver? (see page 381)	{0 1}
:HARDcopy:LAYout <layout> (see page 382)	:HARDcopy:LAYout? (see page 382)	<layout> ::= {LANDscape PORTRait}
:HARDcopy:NETWork:ADD Ress <address> (see page 383)	:HARDcopy:NETWork:ADD Ress? (see page 383)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see page 384)	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see page 385)	:HARDcopy:NETWork:DOM ain? (see page 385)	<domain> ::= quoted ASCII string



Table 93 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:NETWork:PAS Sword <password> (see page 386)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLO T <slot> (see page 387)	:HARDcopy:NETWork:SLO T? (see page 387)	<slot> ::= {NET0 NET1}
:HARDcopy:NETWork:USE Rname <username> (see page 388)	:HARDcopy:NETWork:USE Rname? (see page 388)	<username> ::= quoted ASCII string
:HARDcopy:PAlette <palette> (see page 389)	:HARDcopy:PAlette? (see page 389)	<palette> ::= {COLOR GRAYscale NONE}
n/a	:HARDcopy:PRINTER:LIS T? (see page 390)	<list> ::= [<printer_spec>] ... <printer_spec> ::= " <index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y N} <name> ::= name of printer
:HARDcopy:START (see page 391)	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 1232)

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 376
 - ":HARDcopy:STARt" on page 391
 - ":HARDcopy:APRinter" on page 378
 - ":HARDcopy:PRINter:LIST" on page 390
 - ":HARDcopy:FACTors" on page 379
 - ":HARDcopy:FFEed" on page 380
 - ":HARDcopy:INKSaver" on page 381
 - ":HARDcopy:LAYout" on page 382
 - ":HARDcopy:PALETTE" on page 389

:HARDcopy:APRinter

N (see [page 1232](#))

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 376

• "[":HARDcopy:PRINter:LIST](#)" on page 390

• "[":HARDcopy:STARt](#)" on page 391

:HARDcopy:FACTors

N (see page 1232)

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 376
- ":HARDcopy:STARt" on page 391
- ":HARDcopy:FFEed" on page 380
- ":HARDcopy:INKSaver" on page 381
- ":HARDcopy:LAYOUT" on page 382
- ":HARDcopy:PALETTE" on page 389

:HARDcopy:FFEed

N (see page 1232)

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 376
 - ":HARDcopy:STARt" on page 391
 - ":HARDcopy:FACTors" on page 379
 - ":HARDcopy:INKSaver" on page 381
 - ":HARDcopy:LAYOUT" on page 382
 - ":HARDcopy:PALETTE" on page 389

:HARDcopy:INKSaver

N (see page 1232)

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 376
 - ":HARDcopy:STARt" on page 391
 - ":HARDcopy:FACTors" on page 379
 - ":HARDcopy:FFEed" on page 380
 - ":HARDcopy:LAYOUT" on page 382
 - ":HARDcopy:PALETTE" on page 389

:HARDcopy:LAYout

N (see page 1232)

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 376

- "[":HARDcopy:STARt](#)" on page 391
- "[":HARDcopy:FACTors](#)" on page 379
- "[":HARDcopy:PALETTE](#)" on page 389
- "[":HARDcopy:FFEEd](#)" on page 380
- "[":HARDcopy:INKSaver](#)" on page 381

:HARDcopy:NETWork:ADDResS

N (see page 1232)

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 376
 - ":HARDcopy:NETWork:SLOT" on page 387
 - ":HARDcopy:NETWork:APPLy" on page 384
 - ":HARDcopy:NETWork:DOMain" on page 385
 - ":HARDcopy:NETWork:USERname" on page 388
 - ":HARDcopy:NETWork:PASSword" on page 386

:HARDcopy:NETWork:APPLy

N (see page 1232)

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 376
- "[":HARDcopy:NETWork:SLOT](#)" on page 387
- "[":HARDcopy:NETWork:ADDResS](#)" on page 383
- "[":HARDcopy:NETWork:DOMain](#)" on page 385
- "[":HARDcopy:NETWork:USERname](#)" on page 388
- "[":HARDcopy:NETWork:PASSword](#)" on page 386

:HARDcopy:NETWork:DOMain

N (see page 1232)

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 376
- ":HARDcopy:NETWork:SLOT" on page 387
- ":HARDcopy:NETWork:APPLy" on page 384
- ":HARDcopy:NETWork:ADDress" on page 383
- ":HARDcopy:NETWork:USERname" on page 388
- ":HARDcopy:NETWork:PASSword" on page 386

:HARDcopy:NETWork:PASSword

N (see page 1232)

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 376
 - "[":HARDcopy:NETWork:USERname](#)" on page 388
 - "[":HARDcopy:NETWork:DOMain](#)" on page 385
 - "[":HARDcopy:NETWork:SLOT](#)" on page 387
 - "[":HARDcopy:NETWork:APPLy](#)" on page 384
 - "[":HARDcopy:NETWork:ADDress](#)" on page 383

:HARDcopy:NETWork:SLOT

N (see page 1232)

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 376
- ":HARDcopy:NETWork:APPLy" on page 384
- ":HARDcopy:NETWork:ADDress" on page 383
- ":HARDcopy:NETWork:DOMain" on page 385
- ":HARDcopy:NETWork:USERname" on page 388
- ":HARDcopy:NETWork:PASSword" on page 386

:HARDcopy:NETWork:USERname

N (see page 1232)

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 376
 - "[":HARDcopy:NETWork:DOMain](#)" on page 385
 - "[":HARDcopy:NETWork:PASSword](#)" on page 386
 - "[":HARDcopy:NETWork:SLOT](#)" on page 387
 - "[":HARDcopy:NETWork:APPLy](#)" on page 384
 - "[":HARDcopy:NETWork:ADDRess](#)" on page 383

:HARDcopy:PALETTE

N (see page 1232)

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 376
 - ":HARDcopy:STARt" on page 391
 - ":HARDcopy:FACTors" on page 379
 - ":HARDcopy:LAYOUT" on page 382
 - ":HARDcopy:FFEd" on page 380
 - ":HARDcopy:INKSaver" on page 381

:HARDcopy:PRINTER:LIST

N (see page 1232)

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 376
- "[":HARDcopy:APRINTER](#)" on page 378
- "[":HARDcopy:STARt](#)" on page 391

:HARDcopy:STARt

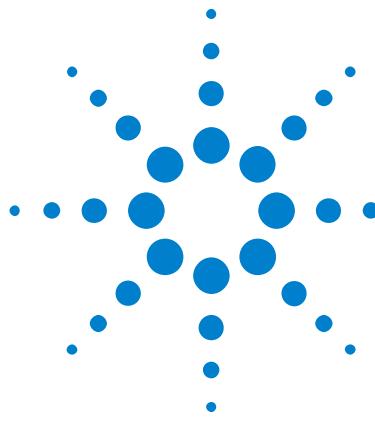
N (see page 1232)

Command Syntax :HARDcopy:STARt

The :HARDcopy:STARt command starts a print job.

See Also

- "Introduction to :HARDcopy Commands" on page 376
- ":HARDcopy:APRinter" on page 378
- ":HARDcopy:PRINTER:LIST" on page 390
- ":HARDcopy:FACTors" on page 379
- ":HARDcopy:FFEed" on page 380
- ":HARDcopy:INKSaver" on page 381
- ":HARDcopy:LAYOUT" on page 382
- ":HARDcopy:PALETTE" on page 389



19
:LISTER Commands

Table 94 :LISTER Commands Summary

Command	Query	Options and Query Returns
n/a	:LISTER:DATA? (see page 394)	<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 395)	:LISTER:DISPLAY? (see page 395)	{OFF SBUS1 SBUS2 ALL}
:LISTER:REFERENCE <time_ref> (see page 396)	:LISTER:REFERENCE? (see page 396)	<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 1232)

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 393
 - ":LISTer:DISPLAY" on page 395
 - "Definite-Length Block Response Data" on page 162

:LISTer:DISPlay

N (see page 1232)

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 393
 - ":SBUS<n>:DISPLAY" on page 660
 - ":LISTer:DATA" on page 394

:LISTER:REFERENCE

N (see page 1232)

Command Syntax :LISTER:REFERENCE <time_ref>
 <time_ref> ::= {TRIGGER | PREVIOUS}

The :LISTER:REFERENCE command selects whether the time value for a Lister row is relative to the trigger or the previous Lister row.

Query Syntax :LISTER:REFERENCE?

The :LISTER:REFERENCE? query returns the Lister time reference setting.

Return Format <time_ref><NL>
 <time_ref> ::= {TRIGGER | PREVIOUS}

See Also • "Introduction to :LISTER Commands" on page 393
 • ":SBUS<n>:DISPLAY" on page 660
 • ":LISTER:DATA" on page 394
 • ":LISTER:DISPLAY" on page 395

20 :MARKer Commands

Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). See "[Introduction to :MARKer Commands](#)" on page 398.

Table 95 :MARKer Commands Summary

Command	Query	Options and Query Returns
n/a	:MARKer:DYDX? (see page 400)	<return_value> ::= ΔY/ΔX value in NR3 format
:MARKer:MODE <mode> (see page 401)	:MARKer:MODE? (see page 401)	<mode> ::= {OFF MEASurement MANual WAVEform BINARY HEX}
:MARKer:X1Position <position>[suffix] (see page 402)	:MARKer:X1Position? (see page 402)	<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 403)	:MARKer:X1Y1source? (see page 403)	<source> ::= {CHANNEL<n> FUNCTION<m> MATH<m> WMEMORY<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 404)	:MARKer:X2Position? (see page 404)	<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



Table 95 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X2Y2source <source> (see page 405)	:MARKer:X2Y2source? (see page 405)	<pre> <source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= <source> </pre>
n/a	:MARKer:XDELta? (see page 406)	<pre> <return_value> ::= X cursors delta value in NR3 format </pre>
:MARKer:XUNits <mode> (see page 407)	:MARKer:XUNits? (see page 407)	<pre> <units> ::= {SEConds HERTz DEGRees PERCent} </pre>
:MARKer:XUNits:USE (see page 408)	n/a	n/a
:MARKer:Y1Position <position>[suffix] (see page 409)	:MARKer:Y1Position? (see page 409)	<pre> <position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y1 cursor position value in NR3 format </pre>
:MARKer:Y2Position <position>[suffix] (see page 410)	:MARKer:Y2Position? (see page 410)	<pre> <position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y2 cursor position value in NR3 format </pre>
n/a	:MARKer:YDELta? (see page 411)	<pre> <return_value> ::= Y cursors delta value in NR3 format </pre>
:MARKer:YUNits <mode> (see page 412)	:MARKer:YUNits? (see page 412)	<pre> <units> ::= {BASE PERCent} </pre>
:MARKer:YUNits:USE (see page 413)	n/a	n/a

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

Reporting the Setup

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

Return Format

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

```
:MARK:X1Y1 CHAN1;X2Y2 CHAN1;MODE MAN
```

:MARKer:DYDX**N** (see page 1232)**Query Syntax** :MARKer:DYDX?

The MARKer:DYDX? query returns the cursor $\Delta Y/\Delta X$ value.

X cursor units are set by the :MARKer:XUNits command.

NOTE

If the front-panel cursors are off, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVEform to put the cursors in the front-panel Normal mode.

Return Format <value><NL>

<value> ::= $\Delta Y/\Delta X$ value in NR3 format.

See Also

- "Introduction to :MARKer Commands" on page 398
- ":MARKer:MODE" on page 401
- ":MARKer:X1Position" on page 402
- ":MARKer:X2Position" on page 404
- ":MARKer:X1Y1source" on page 403
- ":MARKer:X2Y2source" on page 405
- ":MARKer:XUNits" on page 407

:MARKer:MODE

N (see page 1232)

Command Syntax

```
:MARKer:MODE <mode>
<mode> ::= {OFF | MEASurement | MANual | WAVEform | BINary | HEX}
```

The :MARKer:MODE command sets the cursors mode:

- OFF – removes the cursor information from the display.
- MANual – enables manual placement of the X and Y cursors.

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

- MEASurement – cursors track the most recent measurement.

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

- WAVEform – the Y1 cursor tracks the voltage value at the X1 cursor of the waveform specified by the X1Y1source, and the Y2 cursor does the same for the X2 cursor and its X2Y2source.
- BINary – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in binary.
- HEX – logic levels of displayed waveforms at the current X1 and X2 cursor positions are displayed in the Cursor sidebar dialog in hexadecimal.

Query Syntax

```
:MARKer:MODE?
```

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

Return Format

```
<mode><NL>
<mode> ::= {OFF | MEAS | MAN | WAV | BIN | HEX}
```

See Also

- "Introduction to :MARKer Commands" on page 398
- ":MARKer:X1Y1source" on page 403
- ":MARKer:X2Y2source" on page 405
- ":MEASure:SOURce" on page 468
- ":MARKer:X1Position" on page 402
- ":MARKer:X2Position" on page 404
- ":MARKer:Y1Position" on page 409
- ":MARKer:Y2Position" on page 410

:MARKer:X1Position

N (see page 1232)

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 401).
- 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 398

- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X2Position](#)" on page 404
- "[:MARKer:X1Y1source](#)" on page 403
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MARKer:XUNits](#)" on page 407
- "[:MEASure:TSTArt](#)" on page 1166

:MARKer:X1Y1source

N (see page 1232)

Command Syntax

```
:MARKer:X1Y1source <source>
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see "[:MARKer:MODE](#)" on page 401):

- 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 398
- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MEASure:SOURce](#)" on page 468

:MARKer:X2Position

N (see page 1232)

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 401).
- 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 398
- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X1Position](#)" on page 402
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MARKer:XUNits](#)" on page 407
- "[:MEASure:TSTOp](#)" on page 1167

:MARKer:X2Y2source

N (see page 1232)

Command Syntax

```
:MARKer:X2Y2source <source>
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see "[:MARKer:MODE](#)" on page 401):

- 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 398
- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X1Y1source](#)" on page 403
- "[:MEASure:SOURce](#)" on page 468

:MARKer:XDELta

N (see [page 1232](#))

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 398
- "[":MARKer:MODE](#)" on page 401
- "[":MARKer:X1Position](#)" on page 402
- "[":MARKer:X2Position](#)" on page 404
- "[":MARKer:X1Y1source](#)" on page 403
- "[":MARKer:X2Y2source](#)" on page 405
- "[":MARKer:XUNits](#)" on page 407

:MARKer:XUNits

N (see page 1232)

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 398
- "[":MARKer:XUNits:USE](#)" on page 408
- "[":MARKer:X1Y1source](#)" on page 403
- "[":MARKer:X2Y2source](#)" on page 405
- "[":MEASure:SOURce](#)" on page 468
- "[":MARKer:X1Position](#)" on page 402
- "[":MARKer:X2Position](#)" on page 404

:MARKer:XUNits:USE

N (see page 1232)

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 398
- ":MARKer:XUNits" on page 407
- ":MARKer:X1Y1source" on page 403
- ":MARKer:X2Y2source" on page 405
- ":MEASure:SOURce" on page 468
- ":MARKer:X1Position" on page 402
- ":MARKer:X2Position" on page 404
- ":MARKer:XDELta" on page 406

:MARKer:Y1Position

N (see page 1232)

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 401), 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 398

- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X1Y1source](#)" on page 403
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MARKer:Y2Position](#)" on page 410
- "[:MARKer:YUNits](#)" on page 412
- "[:MEASure:VSTArt](#)" on page 1172

:MARKer:Y2Position

N (see page 1232)

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 401), 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 398

- "[:MARKer:MODE](#)" on page 401
- "[:MARKer:X1Y1source](#)" on page 403
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MARKer:Y1Position](#)" on page 409
- "[:MARKer:YUNits](#)" on page 412
- "[:MEASure:VSTOP](#)" on page 1173

:MARKer:YDELta

N (see page 1232)

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 398
- "[":MARKer:MODE](#)" on page 401
- "[":MARKer:X1Y1source](#)" on page 403
- "[":MARKer:X2Y2source](#)" on page 405
- "[":MARKer:Y1Position](#)" on page 409
- "[":MARKer:Y2Position](#)" on page 410
- "[":MARKer:YUNits](#)" on page 412

:MARKer:YUNits

N (see page 1232)

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 398
 - ":MARKer:YUNits:USE" on page 413
 - ":MARKer:X1Y1source" on page 403
 - ":MARKer:X2Y2source" on page 405
 - ":MEASure:SOURce" on page 468
 - ":MARKer:Y1Position" on page 409
 - ":MARKer:Y2Position" on page 410
 - ":MARKer:YDELta" on page 411

:MARKer:YUNits:USE

N (see page 1232)

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 398
- ":MARKer:YUNits" on page 412
- ":MARKer:X1Y1source" on page 403
- ":MARKer:X2Y2source" on page 405
- ":MEASure:SOURce" on page 468
- ":MARKer:Y1Position" on page 409
- ":MARKer:Y2Position" on page 410
- ":MARKer:YDELta" on page 411

21 :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 428.

Table 96 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 430)	n/a	n/a
:MEASure:AREA [<i><interval></i>] [,<source>] (see page 431)	:MEASure:AREA? [<interval>] [,<source>] (see page 431)	<p><interval> ::= {CYCLE DISPLAY}</p> <p><source> ::= {CHANNEL<n> FUNCTION<m> MATH<m> WMEMORY<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= area in volt-seconds, NR3 format</p>
:MEASure:BWIDth [<source>] (see page 432)	:MEASure:BWIDth? [<source>] (see page 432)	<p><source> ::= {CHANNEL<n> FUNCTION<m> MATH<m> WMEMORY<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= burst width in seconds, NR3 format</p>
:MEASure:CLEar (see page 433)	n/a	n/a



Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:COUNTER [<source>] (see page 434)	:MEASure:COUNTER? [<source>] (see page 434)	<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 436)	:MEASure:DEFine? DELay (see page 437)	<delay spec> ::= <edge_spec1>, <edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ -} <occurrence> ::= integer
:MEASure:DEFine THresholds, <threshold spec> (see page 436)	:MEASure:DEFine? THresholds (see page 437)	<threshold spec> ::= {STANDARD} {<threshold mode>, <upper>, <middle>, <lower>} <threshold mode> ::= {PERCent ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 439)	:MEASure:DELay? [<source1>] [,<source2>] (see page 439)	<source1,2> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUAL:CHARGE [<interval>] [,<source1>] [,<source2>] (see page 441)	:MEASure:DUAL:CHARge? [<interval>] [,<source1>] [,<source2>] (see page 441)	<interval> ::= {CYCLE DISPLAY} <source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= area in Amp-hours, NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUAL:VAMplitude [<source1>] [,<source2>] (see page 442)</source1>	:MEASure:DUAL:VAMplitude? [<source1>] [,<source2>] (see page 442)</source1>	<source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:DUAL:VAverage [<interval>] [,<source1>] [,<source2>] (see page 443)	:MEASure:DUAL:VAverage? [<interval>] [,<source1>] [,<source2>] (see page 443)	<interval> ::= {CYCLE DISPLAY} <source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:DUAL:VBASE [<source1>] [,<source2>] (see page 444)	:MEASure:DUAL:VBASE? [<source1>] [,<source2>] (see page 444)	<source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:DUAL:VPP [<source1>] [,<source2>] (see page 445)	:MEASure:DUAL:VPP? [<source1>] [,<source2>] (see page 445)	<source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:DUAL:VRMS [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 446)	:MEASure:DUAL:VRMS? [<interval>] [,<type>] [,<source1>] [,<source2>] (see page 446)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source1>, <source2> ::= CHANnel<n> with N2820A probe connected <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= calculated RMS voltage in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 447)	:MEASure:DUTYcycle? [<source>] (see page 447)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= ratio of positive pulse width to period in NR3 format</p>
:MEASure:FALLtime [<source>] (see page 448)	:MEASure:FALLtime? [<source>] (see page 448)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= time in seconds between the lower and upper thresholds in NR3 format</p>

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FREQuency [<source>] (see page 449)	:MEASure:FREQuency? [<source>] (see page 449)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION<m> MATH<m> WMMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= frequency in Hertz in NR3 format</p>
:MEASure:NDUTy [<source>] (see page 450)	:MEASure:NDUTy? [<source>] (see page 450)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMMemory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTION<m> MATH<m> WMMemory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= ratio of negative pulse width to period in NR3 format</p>
:MEASure:NEDGes [<source>] (see page 451)	:MEASure:NEDGes? [<source>] (see page 451)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMMemory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the falling edge count in NR3 format</p>

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NPULses [<source>] (see page 452)	:MEASure:NPULses? [<source>] (see page 452)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the falling pulse count in NR3 format</p>
:MEASure:NWIDth [<source>] (see page 453)	:MEASure:NWIDth? [<source>] (see page 453)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= negative pulse width in seconds-NR3 format</p>
:MEASure:OVERshoot [<source>] (see page 454)	:MEASure:OVERshoot? [<source>] (see page 454)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the percent of the overshoot of the selected waveform in NR3 format</p>

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PEDGes [<source>] (see page 456)	:MEASure:PEDGes? [<source>] (see page 456)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the rising edge count in NR3 format</p>
:MEASure:PERiod [<source>] (see page 457)	:MEASure:PERiod? [<source>] (see page 457)	<p><source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models</p> <p><source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><d> ::= 0 to (# digital channels - 1) in NR1 format</p> <p><return_value> ::= waveform period in seconds in NR3 format</p>
:MEASure:PHASE [<source1>] [,<source2>] (see page 458)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 458)	<p><source1,2> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the phase angle value in degrees in NR3 format</p>

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PPULses [<source>] (see page 459)	:MEASure:PPULses? [<source>] (see page 459)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the rising pulse count in NR3 format
:MEASure:PRESHoot [<source>] (see page 460)	:MEASure:PRESHoot? [<source>] (see page 460)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 461)	:MEASure:PWIDth? [<source>] (see page 461)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 462)	<result_list> ::= comma-separated list of measurement results

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:RISetime [<source>] (see page 465)	:MEASure:RISetime? [<source>] (see page 465)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 466)	:MEASure:SDEViation? [<source>] (see page 466)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {{0 OFF} {1 ON}} (see page 467)	:MEASure:SHOW? (see page 467)	{0 1}
:MEASure:SOURce <source1> [,<source2>] (see page 468)	:MEASure:SOURce? (see page 468)	<source1,2> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMMemory<r> EXTERNAL} for DSO models <source1,2> ::= {CHANnel<n> DIGItal<d> FUNCtion<m> MATH<m> WMMemory<r> EXTERNAL} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source> NONE}
:MEASure:STATistics <type> (see page 470)	:MEASure:STATistics? (see page 470)	<type> ::= {{ON 1} CURRENT MEAN MINimum MAXimum STDDev COUNT} ON ::= all statistics returned

21 :MEASure Commands

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:STATistics:D ISPlay {{0 OFF} {1 ON}} (see page 471)	:MEASure:STATistics:D ISPlay? (see page 471)	{0 1}
:MEASure:STATistics:I NCEment (see page 472)	n/a	n/a
:MEASure:STATistics:M CCount <setting> (see page 473)	:MEASure:STATistics:M CCount? (see page 473)	<setting> ::= {INFinite <count>} <count> ::= 2 to 2000 in NR1 format
:MEASure:STATistics:R ESet (see page 474)	n/a	n/a
:MEASure:STATistics:R SDeviation {{0 OFF} {1 ON}} (see page 475)	:MEASure:STATistics:R SDeviation? (see page 475)	{0 1}
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 476)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMORY<r>} for DSO models <source> ::= {CHANnel<n> DIGItal<d> FUNCTION<m> MATH<m> WMEMORY<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TValue? <value>, [<>slope>]<occurrence> [,,<source>] (see page 479)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMplitude [<>source>] (see page 481)	:MEASure:VAMplitude? [<>source>] (see page 481)	<source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<>interval>] [,,<source>] (see page 482)	:MEASure:VAverage? [<>interval>] [,,<source>] (see page 482)	<interval> ::= {CYCLE DISPLAY} <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated average voltage in NR3 format

21 :MEASure Commands

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VBASe [<source>] (see page 483)	:MEASure:VBASe? [<source>] (see page 483)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 484)	:MEASure:VMAX? [<source>] (see page 484)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 485)	:MEASure:VMIN? [<source>] (see page 485)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 486)	:MEASure:VPP? [<source>] (see page 486)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VRATio [<interval>] [,<source 1>] [,<source2>] (see page 487)	:MEASure:VRATio? [<interval>] [,<source 1>] [,<source2>] (see page 487)	<interval> ::= {CYCLE DISPLAY} <source1,2> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<interval>] [,<type>] [,<source>] (see page 488)	:MEASure:VRMS? [<interval>] [,<type>] [,<source>] (see page 488)	<interval> ::= {CYCLE DISPLAY} <type> ::= {AC DC} <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtim>[,<source>] (see page 489)	<vtim> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>} for DSO models <source> ::= {CHANnel<n> DIGital<d> FUNCtion<m> MATH<m> WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format

Table 96 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 490)	:MEASure:VTOP? [<source>] (see page 490)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDOW <type> (see page 491)	:MEASure:WINDOW? (see page 491)	<type> ::= {MAIN ZOOM AUTO GATE}
:MEASure:XMAX [<source>] (see page 492)	:MEASure:XMAX? [<source>] (see page 492)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 493)	:MEASure:XMIN? [<source>] (see page 493)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format

Introduction to :MEASure Commands The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the zoomed (delayed) time base mode (:TIMEbase:MODE WINDOW), the oscilloscope will attempt to make the measurement inside the zoomed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on 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 1232](#))

Command Syntax :MEASure:ALL

This command installs a Snapshot All measurement on the screen.

See Also • "Introduction to :MEASure Commands" on page 428

:MEASure:AREA

N (see page 1232)

Command Syntax :MEASure:AREA [<interval>] [,<source>]

```
<interval> ::= {CYCLE | DISPLAY}
<source> ::= {CHANnel<n> | FUNCTion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:AREA command installs an area measurement on screen. Area measurements show the area between the waveform and the ground level.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:AREA? [<interval>] [,<source>]

The :MEASure:AREA? query measures and returns the area value.

Return Format

<value><NL>

<value> ::= the area value in volt-seconds in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:BWIDth

N (see page 1232)

Command Syntax

```
:MEASure:BWIDth [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:BWIDth command installs a burst width measurement on screen. If the optional source parameter is not specified, the current measurement source is used.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:BWIDth? [<source>]
```

The :MEASure:BWIDth? query measures and returns the width of the burst on the screen.

The burst width is calculated as follows:

$$\text{burst width} = (\text{last edge on screen} - \text{first edge on screen})$$

Return Format

```
<value><NL>
<value> ::= burst width in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:CLEar

N (see page 1232)

Command Syntax :MEASure:CLEar

This command clears all selected measurements and markers from the screen.

See Also • "Introduction to :MEASure Commands" on page 428

:MEASure:COUNter

N (see page 1232)

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 or Reference Waveforms may be selected for the source.

The counter measurement counts trigger level crossings within a certain amount of time (gate time) and displays the results in Hz.

The gate time is the horizontal range of the oscilloscope but is limited to ≥ 0.1 s and ≤ 10 s. Unlike other measurements, the Zoom horizontal timebase window does not gate the Counter measurement.

The Counter measurement can measure frequencies up to the bandwidth of the oscilloscope. The minimum frequency supported is $2.0 / \text{gateTime}$.

The Y cursor shows 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 428
- ":MEASure:SOURce" on page 468

- "[:MEASure:FREQuency](#)" on page 449
- "[:MEASure:CLEar](#)" on page 433

:MEASure:DEFine

N (see page 1232)

Command Syntax

```
:MEASure:DEFine <meas_spec>
<meas_spec> ::= {DEDelay | 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 DEDelay 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	DEDelay	THresholds
DUTYcycle		x
DEDelay	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**DEDelay 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:

$$\text{delay} = t(<\text{edge_spec2}>) - t(<\text{edge_spec1}>)$$

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 284: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 428
- "[":MEASure:DELay](#)" on page 439
- "[":MEASure:SOURce](#)" on page 468
- "[":CHANnel<n>:RANGE](#)" on page 283
- "[":CHANnel<n>:SCALE](#)" on page 284
- "[":CHANnel<n>:PROBe](#)" on page 277
- "[":CHANnel<n>:UNITs](#)" on page 285

:MEASure:DElay

N (see page 1232)

Command Syntax

```
:MEASure:DElay [<source1> [,<source2>]
<source1>, <source2> ::= {CHANnel<n> | FUNCTion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:DElay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

$$\text{delay} = t(\text{edge spec 2}) - t(\text{edge spec 1})$$

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 428](#)

- [":MEASure:DEFIne" on page 436](#)
- [":MEASure:PHASe" on page 458](#)

:MEASure:DUAL:CHARge

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax :MEASure:DUAL:CHARge [<interval>] [,<source1>] [,<source2>]

```
<interval> ::= {CYCLE | DISPLAY}
<source1>,<source2> ::= CHANnel<n> with N2820A probe connected
<n> ::= 1 to (# analog channels) in NR1 format
```

The :MEASure:DUAL:CHARge command installs a charge measurement on screen. Charge measurements show the area between the waveform and the ground level.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax :MEASure:DUAL:CHARge? [<interval>] [,<source1>] [,<source2>]

The :MEASure:DUAL:CHARge? query measures and returns the charge measurement value.

Return Format <value><NL>

```
<value> ::= the charge value in Amp-hours in NR3 format
```

- See Also**
- "[:MEASure:DUAL:VAMplitude](#)" on page 442
 - "[:MEASure:DUAL:VAverage](#)" on page 443
 - "[:MEASure:DUAL:VBASe](#)" on page 444
 - "[:MEASure:DUAL:VPP](#)" on page 445
 - "[:MEASure:DUAL:VRMS](#)" on page 446
 - "[Introduction to :MEASure Commands](#)" on page 428
 - "[:MEASure:SOURce](#)" on page 468

:MEASure:DUAL:VAMPlitude

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax :MEASure:DUAL:VAMPlitude [<source1>[,<source2>]

<source1>,<source2> ::= CHANnel<n> with N2820A probe connected
<n> ::= 1 to (# analog channels) in NR1 format

The :MEASure:DUAL:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax :MEASure:DUAL:VAMPlitude? [<source1>[,<source2>]

The :MEASure:DUAL:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

$$\text{vertical amplitude} = \text{Vtop} - \text{Vbase}$$

Return Format <value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format

See Also

- "[:MEASure:DUAL:CHARge](#)" on page 441
- "[:MEASure:DUAL:VAverage](#)" on page 443
- "[:MEASure:DUAL:VBASe](#)" on page 444
- "[:MEASure:DUAL:VPP](#)" on page 445
- "[:MEASure:DUAL:VRMS](#)" on page 446
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:VTOP](#)" on page 490

:MEASure:DUAL:VAverage

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax :MEASure:DUAL:VAverage [<interval>] [,<source1>] [,<source2>]

```
<interval> ::= {CYCLE | DISPLAY}
<source1>,<source2> ::= CHANnel<n> with N2820A probe connected
<n> ::= 1 to (# analog channels) in NR1 format
```

The :MEASure:DUAL:VAverage command installs a screen measurement and starts an average value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax :MEASure:DUAL:VAverage? [<interval>] [,<source1>] [,<source2>]

The :MEASure:DUAL:VAverage? query returns the average value measurement.

Return Format

```
<value><NL>
<value> ::= calculated average value in NR3 format
```

See Also

- "[:MEASure:DUAL:CHARge](#)" on page 441
- "[:MEASure:DUAL:VAMplitude](#)" on page 442
- "[:MEASure:DUAL:VBASe](#)" on page 444
- "[:MEASure:DUAL:VPP](#)" on page 445
- "[:MEASure:DUAL:VRMS](#)" on page 446
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468

:MEASure:DUAL:VBASe

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax `:MEASure:DUAL:VBASe [<source1>[,<source2>]`

`<source1>,<source2>` ::= CHANnel<n> with N2820A probe connected
`<n>` ::= 1 to (# analog channels) in NR1 format

The :MEASure:DUAL:VBASe command installs a screen measurement and starts a waveform base value measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax `:MEASure:DUAL:VBASe? [<source1>[,<source2>]`

The :MEASure:DUAL:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format `<base_voltage><NL>`

`<base_voltage>` ::= value at the base of the selected waveform in NR3 format

- See Also**
- "[:MEASure:DUAL:CHARge](#)" on page 441
 - "[:MEASure:DUAL:VAMplitude](#)" on page 442
 - "[:MEASure:DUAL:VAverage](#)" on page 443
 - "[:MEASure:DUAL:VPP](#)" on page 445
 - "[:MEASure:DUAL:VRMS](#)" on page 446
 - "[Introduction to :MEASure Commands](#)" on page 428
 - "[:MEASure:SOURce](#)" on page 468
 - "[:MEASure:VTOP](#)" on page 490
 - "[:MEASure:VMIN](#)" on page 485

:MEASure:DUAL:VPP

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax :MEASure:DUAL:VPP [<source1>] [,<source2>]

<source1>, <source2> ::= CHANnel<n> with N2820A probe connected
<n> ::= 1 to (# analog channels) in NR1 format

The :MEASure:DUAL:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax :MEASure:DUAL:VPP? [<source1>] [,<source2>]

The :MEASure:DUAL:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

$$V_{pp} = V_{max} - V_{min}$$

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format <value><NL>

<value> ::= vertical peak to peak value in NR3 format

- See Also**
- "[:MEASure:DUAL:CHARge](#)" on page 441
 - "[:MEASure:DUAL:VAMPLitude](#)" on page 442
 - "[:MEASure:DUAL:VAverage](#)" on page 443
 - "[:MEASure:DUAL:VBASe](#)" on page 444
 - "[:MEASure:DUAL:VRMS](#)" on page 446
 - "[Introduction to :MEASure Commands](#)" on page 428
 - "[:MEASure:SOURce](#)" on page 468
 - "[:MEASure:VMAX](#)" on page 484
 - "[:MEASure:VMIN](#)" on page 485

:MEASure:DUAL:VRMS

N (see page 1232)

Overview This measurement is available with the N2820A high sensitivity current probe when both the Primary and Secondary probe cables are used. This measurement joins the Zoom In waveform data below the probe's clamp level with Zoom Out waveform data above the probe's clamp level to create the waveform on which the measurement is made.

Command Syntax

```
:MEASure:DUAL:VRMS [<interval>[,<type>][,<source1>][,<source2>]
<interval> ::= {CYCLE | DISPLAY}
<type> ::= {AC | DC}
<source1>,<source2> ::= CHANnel<n> with N2820A probe connected
<n> ::= 1 to (# analog channels) in NR1 format
```

The :MEASure:DUAL:VRMS command installs a screen measurement and starts an RMS value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

If the optional source parameter(s) are specified, the currently specified source(s) are modified.

Query Syntax

```
:MEASure:DUAL:VRMS? [<interval>[,<type>][,<source1>][,<source2>]
```

The :MEASure:DUAL:VRMS? query measures and outputs the RMS value measurement.

Return Format

```
<value><NL>
<value> ::= calculated dc RMS value in NR3 format
```

See Also

- "[:MEASure:DUAL:CHARge](#)" on page 441
- "[:MEASure:DUAL:VAMplitude](#)" on page 442
- "[:MEASure:DUAL:VAverage](#)" on page 443
- "[:MEASure:DUAL:VBASe](#)" on page 444
- "[:MEASure:DUAL:VPP](#)" on page 445
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468

:MEASure:DUTYcycle

C (see page 1232)

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models
<d> ::= 0 to (# digital channels - 1) in NR1 format
<n> ::= 1 to (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:DUTYcycle command installs a screen measurement and starts a positive duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE

The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the positive duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

$$\text{+duty cycle} = (\text{+pulse width}/\text{period}) * 100$$

Return Format

```
<value><NL>

<value> ::= ratio of positive pulse width to period in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:PERiod](#)" on page 457
- "[:MEASure:PWIDth](#)" on page 461
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:NDUTy](#)" on page 450

Example Code

- "[Example Code](#)" on page 469

:MEASure:FALLtime

 (see page 1232)

Command Syntax

```
:MEASure:FALLtime [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FALLtime? [<source>]
```

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

$$\text{fall time} = \text{time at lower threshold} - \text{time at upper threshold}$$

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 428
- "[":MEASure:RISetime](#)" on page 465
- "[":MEASure:SOURce](#)" on page 468

:MEASure:FREQuency

C (see page 1232)

Command Syntax

```
:MEASure:FREQuency [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCTion<m> | MATH<m>
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:FREQuency command installs a screen measurement and starts a frequency measurement. If the optional source parameter is specified, the current source is modified.

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format

```
<source><NL>

<source> ::= frequency in Hertz in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:PERiod](#)" on page 457

Example Code

- "[Example Code](#)" on page 469

:MEASure:NDUTy

N (see page 1232)

Command Syntax

```
:MEASure:NDUTy [<source>]
<source> ::= {<digital channels> | CHANnel<n> | FUNCTion<m> | MATH<m>
              | WMEMemory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<d> ::= 0 to (# digital channels - 1) in NR1 format
<n> ::= 1 to (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:NDUTy command installs a screen measurement and starts a negative duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE

The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NDUTy? [<source>]
```

The :MEASure:NDUTy? query measures and outputs the negative duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the negative pulse width to the period. The negative pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

$$\text{- duty cycle} = (\text{-pulse width}/\text{period}) * 100$$

Return Format

```
<value><NL>
<value> ::= ratio of negative pulse width to period in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:PERiod](#)" on page 457
- "[:MEASure:NWIDth](#)" on page 453
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:DUTYcycle](#)" on page 447

:MEASure:NEDGes

N (see page 1232)

Command Syntax :MEASure:NEDGes [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:NEDGes command installs a falling edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:NEDGes? [<source>]

The :MEASure:NEDGes? query measures and returns the on-screen falling edge count.

Return Format

<value><NL>

<value> ::= the falling edge count in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:NPULses

N (see page 1232)

Command Syntax `:MEASure:NPULses [<source>]`

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:NPULses command installs a falling pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax `:MEASure:NPULses? [<source>]`

The :MEASure:NPULses? query measures and returns the on-screen falling pulse count.

Return Format `<value><NL>`

```
<value> ::= the falling pulse count in NR3 format
```

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:SOURce](#)" on page 468

:MEASure:NWIDth

C (see page 1232)

Command Syntax :MEASure:NWIDth [<source>]

```
<source> ::= {<digital channels> | CHANnel<n> | FUNCTion<m> | MATH<m>
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is not specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax :MEASure:NWIDth? [<source>]

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

Return Format <value><NL>

<value> ::= negative pulse width in seconds in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:SOURce](#)" on page 468
 - "[":MEASure:PWIDth](#)" on page 461
 - "[":MEASure:PERiod](#)" on page 457

:MEASure:OVERshoot

C (see page 1232)

Command Syntax

```
:MEASure:OVERshoot [<source>]
<source> ::= {CHANnel<n> | FUNCTion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{overshoot} = ((\text{Vmax} - \text{Vtop}) / (\text{Vtop} - \text{Vbase})) \times 100$$

For a falling edge:

$$\text{overshoot} = ((\text{Vbase} - \text{Vmin}) / (\text{Vtop} - \text{Vbase})) \times 100$$

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 428
- ":MEASure:PREShoot" on page 460

- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:VMAX](#)" on page 484
- "[:MEASure:VTOP](#)" on page 490
- "[:MEASure:VBASe](#)" on page 483
- "[:MEASure:VMIN](#)" on page 485

:MEASure:PEDGes

N (see page 1232)

Command Syntax

```
:MEASure:PEDGes [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PEDGes command installs a rising edge count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PEDGes? [<source>]
```

The :MEASure:NEDGes? query measures and returns the on-screen rising edge count.

Return Format

```
<value><NL>
<value> ::= the rising edge count in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:PERiod

C (see page 1232)

Command Syntax

```
:MEASure:PERiod [<source>]
<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
              | WMEMory<r>}
<digital channels> ::= DIGItal<d> for the MSO models
<d> ::= 0 to (# digital channels - 1) in NR1 format
<n> ::= 1 to (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PERiod command installs a screen measurement and starts the period measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

Return Format

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

<value> ::= waveform period in seconds in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468
- ":MEASure:NWIDth" on page 453
- ":MEASure:PVIDth" on page 461
- ":MEASure:FREQuency" on page 449

Example Code

- "Example Code" on page 469

:MEASure:PHASe

N (see page 1232)

Command Syntax `:MEASure:PHASe [<source1> [,<source2>]`

```
<source1>, <source2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMemory<r>}
```

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax `:MEASure:PHASe? [<source1> [,<source2>]`

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

$$\text{phase} = (\text{delay} / \text{period of input 1}) \times 360$$

Return Format `<value><NL>`

`<value>` ::= the phase angle value in degrees in NR3 format

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[":MEASure:DELay](#)" on page 439
- "[":MEASure:PERiod](#)" on page 457
- "[":MEASure:SOURce](#)" on page 468

:MEASure:PPULses

N (see page 1232)

Command Syntax :MEASure:PPULses [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PPULses command installs a rising pulse count measurement on screen. If the optional source parameter is not specified, the current source is measured.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax :MEASure:PPULses? [<source>]

The :MEASure:PPULses? query measures and returns the on-screen rising pulse count.

Return Format <value><NL>

```
<value> ::= the rising pulse count in NR3 format
```

- See Also**
- "Introduction to :MEASure Commands" on page 428
 - ":MEASure:SOURce" on page 468

:MEASure:PREShoot



(see page 1232)

Command Syntax

```
:MEASure:PREShoot [<source>]
<source> ::= {CHANnel<n> | FUNCTION<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:PREShoot? [<source>]
```

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{preshoot} = ((\text{Vmin}-\text{Vbase}) / (\text{Vtop}-\text{Vbase})) \times 100$$

For a falling edge:

$$\text{preshoot} = ((\text{Vmax}-\text{Vtop}) / (\text{Vtop}-\text{Vbase})) \times 100$$

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 428
- "[":MEASure:SOURce](#)" on page 468
- "[":MEASure:VMIN](#)" on page 485
- "[":MEASure:VMAX](#)" on page 484
- "[":MEASure:VTOP](#)" on page 490
- "[":MEASure:VBASE](#)" on page 483

:MEASure:PWIDth

C (see page 1232)

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
              | WMEMemory<r>}

<digital channels> ::= DIGItal<d> for the MSO models
<d> ::= 0 to (# digital channels - 1) in NR1 format
<n> ::= 1 to (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

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

<value> ::= width of positive pulse in seconds in NR3 format

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:NWIDth](#)" on page 453
- "[:MEASure:PERiod](#)" on page 457

:MEASure:RESults

N (see page 1232)

Query Syntax

:MEASure:RESults?

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values.

If more than one measurement is running continuously, the :MEASure:RESults return values are duplicated for each continuous measurement from the first to last (top to bottom) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of 10 continuous measurements that can be continuously displayed at a time.

When no quick measurements are installed, the :MEASure:RESults? query returns nothing (empty string). When the count for any of the measurements is 0, the value of infinity (9.9E+37) is returned for the min, max, mean, and standard deviation.

Return Format

<result_list><NL>

<result_list> ::= comma-separated list of measurement results

The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measurement label	current	min	max	mean	std dev	count
-------------------	---------	-----	-----	------	---------	-------

Measurement label, current, min, max, mean, std dev, and count are only returned if :MEASure:STATistics is ON.

If :MEASure:STATistics is set to CURRent, MIN, MAX, MEAN, STDDev, or COUNT only that particular statistic value is returned for each measurement that is on.

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:STATistics" on page 470

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) = "VAMPplitude"
    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
        End Sub

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

```

:MEASure:RISetime

C (see page 1232)

Command Syntax :MEASure: RISetime [<source>]

```
<source> ::= {CHANnel<n> | FUNCTION<m> | MATH<m> | WMEMORY<r>}

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax :MEASure: RISetime? [<source>]

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

$$\text{rise time} = \text{time at upper threshold} - \text{time at lower threshold}$$

Return Format <value><NL>

```
<value> ::= rise time in seconds in NR3 format
```

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:SOURce"](#) on page 468
 - "[":MEASure:FALLtime"](#) on page 448

:MEASure:SDEViation

N (see page 1232)

Command Syntax

```
:MEASure:SDEViation [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

NOTE

This ":MEASure:VRMS DISPlay, AC" command is the preferred syntax for making standard deviation measurements.

The :MEASure:SDEViation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:SDEViation? [<source>]
```

The :MEASure:SDEViation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

Return Format

```
<value><NL>
<value> ::= calculated std deviation value in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:VRMS](#)" on page 488
- "[:MEASure:SOURce](#)" on page 468

:MEASure:SHOW

N (see page 1232)

Command Syntax :MEASure:SHOW <on_off>

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

The :MEASure:SHOW command enables markers for tracking measurements on the display.

Query Syntax :MEASure:SHOW?

The :MEASure:SHOW? query returns the current state of the markers.

This can return OFF when :MARKer:MODE selects a mode other than MEASurement.

Return Format <on_off><NL>

<on_off> ::= {1 | 0}

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MARKer:MODE" on page 401

:MEASure:SOURce

C (see page 1232)

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 428
 - ":MARKer:MODE" on page 401
 - ":MARKer:X1Y1source" on page 403
 - ":MARKer:X2Y2source" on page 405
 - ":MEASure:DELay" on page 439

- “[:MEASure:PHASE](#)” on page 458

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 41](#), “Programming Examples,” starting on page 1241

:MEASure:STATistics

N (see page 1232)

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 428
 - "[:MEASure:RESults](#)" on page 462
 - "[:MEASure:STATistics:DISPlay](#)" on page 471
 - "[:MEASure:STATistics:RESet](#)" on page 474
 - "[:MEASure:STATistics:INCReement](#)" on page 472

- Example Code**
- "[Example Code](#)" on page 462

:MEASure:STATistics:DISPlay

N (see page 1232)

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 428

- ":MEASure:RESults" on page 462
- ":MEASure:STATistics" on page 470
- ":MEASure:STATistics:MCOUNT" on page 473
- ":MEASure:STATistics:RESET" on page 474
- ":MEASure:STATistics:INCREMENT" on page 472
- ":MEASure:STATistics:RSD deviation" on page 475

:MEASure:STATistics:INCRement

N (see page 1232)

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 428
- "[:MEASure:STATistics](#)" on page 470
- "[:MEASure:STATistics:DISPlay](#)" on page 471
- "[:MEASure:STATistics:RESet](#)" on page 474
- "[:MEASure:RESults](#)" on page 462

:MEASure:STATistics:MCOUNT

N (see page 1232)

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 428
- ":MEASure:RESults" on page 462
- ":MEASure:STATistics" on page 470
- ":MEASure:STATistics:DISPlay" on page 471
- ":MEASure:STATistics:RSDDeviation" on page 475
- ":MEASure:STATistics:RESet" on page 474
- ":MEASure:STATistics:INCrement" on page 472

:MEASure:STATistics:RESet

N (see page 1232)

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 428
 - "[:MEASure:STATistics](#)" on page 470
 - "[:MEASure:STATistics:DISPlay](#)" on page 471
 - "[:MEASure:RESults](#)" on page 462
 - "[:MEASure:STATistics:INCReement](#)" on page 472

Example Code

- "[Example Code](#)" on page 462

:MEASure:STATistics:RSDeviation

N (see page 1232)

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 428
 - "[:MEASure:RESults](#)" on page 462
 - "[:MEASure:STATistics](#)" on page 470
 - "[:MEASure:STATistics:DISPlay](#)" on page 471
 - "[:MEASure:STATistics:MCount](#)" on page 473
 - "[:MEASure:STATistics:RESet](#)" on page 474
 - "[:MEASure:STATistics:INCrement](#)" on page 472

:MEASure:TEDGe

N (see page 1232)

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<m> | MATH<m> | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

<m> ::= 1-4 in NR1 format

<r> ::= 1 to (# ref waveforms) 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 477.

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 41](#), “Programming Examples,” starting on page 1241

See Also

- “[Introduction to :MEASure Commands](#)” on page 428

- "[:MEASure:TVALue](#)" on page 479
- "[:MEASure:VTIMe](#)" on page 489

:MEASure:TValue**C**

(see page 1232)

Query Syntax

```
:MEASure:TValue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross. The
          value can be volts or a math function value such as dB,
          Vs, or V/s.

<slope> ::= direction of the waveform. A rising slope is indicated
           by a plus sign (+). A falling edge is indicated by a
           minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence
                  number is one, the first crossing is reported. If
                  the number is two, the second crossing is reported,
                  etc.

<source> ::= {CHANnel<n> | FUNCTion<m> | MATH<m> | WMMemory<r>}

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

<m> ::= 1-4 in NR1 format

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

When the :MEASure:TValue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Return Format <value><NL>

<value> ::= time in seconds of the specified value crossing in
NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:TEDGe](#)" on page 476
 - "[":MEASure:VTIMe](#)" on page 489

:MEASure:VAMPlitude

C (see page 1232)

Command Syntax :MEASure:VAMPlitude [<source>]

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax :MEASure:VAMPlitude? [<source>]

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

$$\text{vertical amplitude} = \text{Vtop} - \text{Vbase}$$

Return Format <value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format

- See Also**
- "Introduction to :MEASure Commands" on page 428
 - ":MEASure:SOURce" on page 468
 - ":MEASure:VBASe" on page 483
 - ":MEASure:VTOP" on page 490
 - ":MEASure:VPP" on page 486

:MEASure:VAverage

(see page 1232)

Command Syntax

```
:MEASure:VAverage [<interval>] [,<source>]
<interval> ::= {CYCLE | DISPLAY}
<source> ::= {CHANNEL<n> | FUNCTION<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VAverage command installs a screen measurement and starts an average value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAverage? [<interval>] [,<source>]
```

The :MEASure:VAverage? query returns the average value of an integral number of periods of the signal.

Return Format

```
<value><NL>
<value> ::= calculated average value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:VBASe

C (see page 1232)

Command Syntax

```
:MEASure:VBASe [<source>]
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

```
<base_voltage><NL>
<base_voltage> ::= value at the base of the selected waveform in
NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[":MEASure:SOURce](#)" on page 468
- "[":MEASure:VTOP](#)" on page 490
- "[":MEASure:VAMplitude](#)" on page 481
- "[":MEASure:VMIN](#)" on page 485

:MEASure:VMAX

(see page 1232)

Command Syntax `:MEASure:VMAX [<source>]`

```

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format

```

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax `:MEASure:VMAX? [<source>]`

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

Return Format `<value><NL>`

```

<value> ::= maximum vertical value of the selected waveform in
NR3 format

```

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:SOURce](#)" on page 468
 - "[":MEASure:VMIN](#)" on page 485
 - "[":MEASure:VPP](#)" on page 486
 - "[":MEASure:VTOP](#)" on page 490

:MEASure:VMIN

(see page 1232)

Command Syntax `:MEASure:VMIN [<source>]`

```

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format

```

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax `:MEASure:VMIN? [<source>]`

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

Return Format `<value><NL>`

```

<value> ::= minimum vertical value of the selected waveform in
NR3 format

```

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 428
 - "[":MEASure:SOURce](#)" on page 468
 - "[":MEASure:VBASe](#)" on page 483
 - "[":MEASure:VMAX](#)" on page 484
 - "[":MEASure:VPP](#)" on page 486

:MEASure:VPP

(see page 1232)

Command Syntax

```
:MEASure:VPP [<source>]

<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

$$V_{pp} = V_{max} - V_{min}$$

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 428
- "[":MEASure:SOURce](#)" on page 468
- "[":MEASure:VMAX](#)" on page 484
- "[":MEASure:VMIN](#)" on page 485
- "[":MEASure:VAMPLitude](#)" on page 481

:MEASure:VRATio

N (see page 1232)

Command Syntax :MEASure:VRATio [<interval>] [,<source1>] [,<source2>]

```
<interval> ::= {CYCLE | DISPLAY}
<source1,2> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMMemory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VRATio command installs a ratio measurement on screen. Ratio measurements show the ratio of the ACRMS value of source1 to that of source2, expressed in dB.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

:MEASure:VRATio? [<interval>] [<source1>] [,<source2>]

The :MEASure:VRATio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

Return Format

```
<value><NL>
<value> ::= the ratio value in dB in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:VRMS" on page 488
- ":MEASure:SOURce" on page 468

:MEASure:VRMS

(see page 1232)

Command Syntax `:MEASure:VRMS [<interval>[,<type>][,<source>]`

```

<interval> ::= {CYCLE | DISPLAY}
<type> ::= {AC | DC}
<source> ::= {CHANNEL<n> | FUNCTION<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format

```

The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPLAY is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax `:MEASure:VRMS? [<interval>[,<type>][,<source>]`

The :MEASure:VRMS? query measures and outputs the RMS measurement value.

Return Format `<value><NL>`

```

<value> ::= calculated dc RMS value in NR3 format

```

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:SOURce" on page 468

:MEASure:VTIMe

N (see page 1232)

Query Syntax :MEASure:VTIMe? <vtimetime_argument>[,<source>]

```

<vtimetime_argument> ::= time from trigger in seconds

<source> ::= {<digital channels> | CHANnel<n> | FUNCtion<m> | MATH<m>
              | WMEMory<r>}

<digital channels> ::= DIGItal<d> for the MSO models

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

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

<m> ::= 1-4 in NR1 format

<r> ::= 1 to (# ref waveforms) 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 428
- ":MEASure:SOURce" on page 468
- ":MEASure:TEDGe" on page 476
- ":MEASure:TVALue" on page 479

:MEASure:VTOP

C (see page 1232)

Command Syntax

```
:MEASure:VTOP [source]
<source> ::= {CHANnel<n> | FUNCTion<m> | MATH<m>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE

This query is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VTOP? [source]
```

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format

```
<value><NL>
<value> ::= vertical value at the top of the waveform in NR3 format
```

See Also

- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:VMAX](#)" on page 484
- "[:MEASure:VAMPplitude](#)" on page 481
- "[:MEASure:VBASe](#)" on page 483

:MEASure:WINDOW

N (see page 1232)

Command Syntax :MEASure:WINDOW <type>

<type> ::= {MAIN | ZOOM | AUTO | GATE}

The :MEASure:WINDOW command lets you choose whether measurements are made in the Main window portion of the display, the Zoom window portion of the display (when the zoomed time base is displayed), or gated by the X1 and X2 cursors.

- MAIN – the measurement window is the Main window.
- ZOOM – the measurement window is the lower, Zoom window.
- AUTO – when the zoomed time base is displayed, the measurement is attempted in the lower, Zoom window; if it cannot be made there, or if the zoomed time base is not displayed, the Main window is used.
- GATE – the measurement window is between the X1 and X2 cursors. When the zoomed time base is displayed, the X1 and X2 cursors in the Zoom window portion of the display are used.

Query Syntax :MEASure:WINDOW?

The :MEASure:WINDOW? query returns the current measurement window setting.

Return Format <type><NL>

<type> ::= {MAIN | ZOOM | AUTO | GATE}

- See Also**
- "Introduction to :MEASure Commands" on page 428
 - ":MEASure:SOURce" on page 468

:MEASure:XMAX

N (see page 1232)

Command Syntax `:MEASure:XMAX [<source>]`

```
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE

:MEASure:XMAX is an alias for :MEASure:TMAX.

Query Syntax `:MEASure:XMAX? [<source>]`

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format `<value><NL>`

```
<value> ::= horizontal value of the maximum in NR3 format
```

- See Also**
- "Introduction to :MEASure Commands" on page 428
 - ":MEASure:XMIN" on page 493
 - ":MEASure:TMAX" on page 1164

:MEASure:XMIN

N (see page 1232)

Command Syntax :MEASure:XMIN [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMORY<r>}
<n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE

:MEASure:XMIN is an alias for :MEASure:TMIN.

Query Syntax

:MEASure:XMIN? [<source>]

The :MEASure:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

<value><NL>

<value> ::= horizontal value of the minimum in NR3 format

See Also

- "Introduction to :MEASure Commands" on page 428
- ":MEASure:XMAX" on page 492
- ":MEASure:TMIN" on page 1165

22

:MEASure Power Commands

These :MEASure commands are available when the DSOX4PWR power measurements and analysis application is licensed and enabled.

Table 97 :MEASure Power Commands Summary

Command	Query	Options and Query Returns
:MEASure:ANGLE [<source1>] [,<source2>] (see page 499)	:MEASure:ANGLE? [<source1>] [,<source2>] (see page 499)	<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 500)	:MEASure:APPARENT? [<source1>] [,<source2>] (see page 500)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the apparent power value in NR3 format
:MEASure:CPLoss [<source1>] [,<source2>] (see page 501)	:MEASure:CPLoss? [<source1>] [,<source2>] (see page 501)	<source1>, <source2> <source1> ::= {FUNCTION<m> MATH<m>} <source2> ::= {CHANnel<n>} <m> ::= 1-4 in NR1 format <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the switching loss per cycle watts value in NR3 format
:MEASure:CREST [<source>] (see page 502)	:MEASure:CREST? [<source>] (see page 502)	<source> ::= {CHANnel<n> FUNCTION<m> MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <return_value> ::= the crest factor value in NR3 format



Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:EFFiciency (see page 503)	:MEASure:EFFiciency? (see page 503)	<return_value> ::= percent value in NR3 format
:MEASure:ELOSS [<source>] (see page 504)	:MEASure:ELOSS? [<source>] (see page 504)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the energy loss value in NR3 format</p>
:MEASure:FACTOr [<source1> [, <source2>] (see page 505)	:MEASure:FACTOr? [<source1> [, <source2>] (see page 505)	<p><source1>, <source2> ::= {CHANnel<n>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><return_value> ::= the power factor value in NR3 format</p>
:MEASure:IPOWer (see page 506)	:MEASure:IPOWer? (see page 506)	<return_value> ::= the input power value in NR3 format
:MEASure:OFFTime [<source1> [, <source2>] (see page 507)	:MEASure:OFFTime? [<source1> [, <source2>] (see page 507)	<p><source1>, <source2> ::= {CHANnel<n>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><return_value> ::= the time in seconds in NR3 format</p>
:MEASure:ONTime [<source1> [, <source2>] (see page 508)	:MEASure:ONTIME? [<source1> [, <source2>] (see page 508)	<p><source1>, <source2> ::= {CHANnel<n>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><return_value> ::= the time in seconds in NR3 format</p>
:MEASure:OPOWer (see page 509)	:MEASure:OPOWer? (see page 509)	<return_value> ::= the output power value in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PCURrent [<source>] (see page 510)	:MEASure:PCURrent? [<source>] (see page 510)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the peak current value in NR3 format
:MEASure:PLOSS [<source>] (see page 511)	:MEASure:PLOSS? [<source>] (see page 511)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m> WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <r> ::= 1 to (# ref waveforms) in NR1 format <return_value> ::= the power loss value in NR3 format
:MEASure:REACTive [<source1>] [,<source2>] (see page 512)	:MEASure:REACTive? [<source1>] [,<source2>] (see page 512)	<source1>, <source2> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format <return_value> ::= the reactive power value in NR3 format
:MEASure:REAL [<source>] (see page 513)	:MEASure:REAL? [<source>] (see page 513)	<source> ::= {CHANnel<n> FUNCTion<m> MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format <return_value> ::= the real power value in NR3 format

Table 97 :MEASure Power Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:RIPPle [<source>] (see page 514)	:MEASure:RIPPle? [<source>] (see page 514)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= the output ripple value in NR3 format</p>
:MEASure:TRESPonse [<source>] (see page 515)	:MEASure:TRESPonse? [<source>] (see page 515)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMory<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><return_value> ::= time in seconds for the overshoot to settle back into the band in NR3 format</p>

:MEASure:ANGLE

N (see page 1232)

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 $\text{apparent_power}^2 = \text{real_power}^2 + \text{reactive_power}^2$), phase angle is the angle between the apparent power and the real power, indicating the amount of reactive power. 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 468
 - "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:APPARENT

N (see page 1232)

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 468
- "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:CPLoss

N (see page 1232)

Command Syntax :MEASure:CPLoss [<source1> [, <source2>]]
 <source1> ::= {FUNCTION<m> | MATH<m>}
 <source2> ::= {CHANnel<n>}
 <m> ::= 1-4 in NR1 format
 <n> ::= 1 to (# analog channels) in NR1 format

The :MEASure:CPLoss command installs a power loss per cycle measurement on screen.

The <source1> parameter is typically a math multiply waveform or other waveform that represents power (voltage * current). This source can also be specified by the :MEASure:SOURce command.

Power loss per cycle is $P_n = (V_{dsn} * I_{dn}) * (\text{Time range of zoom window}) * (\text{Counter measurement of the voltage of the switching signal})$, where n is each sample.

This measurement operates when in zoom mode and the counter measurement is installed on the voltage of the switching signal.

Query Syntax :MEASure:CPLoss? [<source1> [, <source2>]]

The :MEASure:CPLoss query returns the switching loss per cycle in watts.

Return Format <return_value><NL>
 <return_value> ::= the switching loss per cycle value in NR3 format

See Also

- "[:MEASure:SOURce](#)" on page 468
- "[:POWER:SWITch:APPLY](#)" on page 612

:MEASure:CRESt

N (see [page 1232](#))

Command Syntax

```
:MEASure:CRESt [<source>]
<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 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 468

- "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:EFFiciency

N (see page 1232)

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 609
 - "[:POWER:SIGNals:SOURce:CURRent<i>](#)" on page 608
 - "[:POWER:SIGNals:AUTosetup](#)" on page 591
 - "[:POWER:EFFiciency:APPLy](#)" on page 563

:MEASure:ELOSSs

N (see page 1232)

Command Syntax `:MEASure:ELOSSs [<source>]`

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:ELOSSs 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 = $\sum (V_{ds_n} * I_{d_n}) * \text{sample size}$, where n is each sample.

Query Syntax `:MEASure:ELOSSs? [<source>]`

The :MEASure:ELOSSs 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 468
- "[:POWER:SWITch:APPLY](#)" on page 612

:MEASure:FACTOr

N (see page 1232)

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 468
 • "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:IPOWer

N (see page 1232)

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 609
 - "[:POWER:SIGNals:SOURce:CURRent<i>](#)" on page 608
 - "[:POWER:SIGNals:AUTosetup](#)" on page 591
 - "[:POWER:EFFiciency:APPLy](#)" on page 563

:MEASure:OFFTime

N (see page 1232)

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 468
- "[:POWER:ONOFF:TEST](#)" on page 584
- "[:POWER:ONOFF:APPLy](#)" on page 581

:MEASure:ONTime

N (see page 1232)

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 468
 • "[:POWER:ONOFF:TEST](#)" on page 584
 • "[:POWER:ONOFF:APPLY](#)" on page 581

:MEASure:OPOWer

N (see page 1232)

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 609
 - "[:POWER:SIGNals:SOURce:CURRent<i>](#)" on page 608
 - "[:POWER:SIGNals:AUTosetup](#)" on page 591
 - "[:POWER:EFFiciency:APPLy](#)" on page 563

:MEASure:PCURrent

N (see page 1232)

Command Syntax `:MEASure:PCURrent [<source>]`

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:PCURrent command installs a peak current measurement on screen.

The <source> parameter is the channel probing the current. This source can also be specified by the :MEASure:SOURce command.

This command measures the peak current when the power supply first turned on.

Query Syntax `:MEASure:PCURrent? [<source>]`

The :MEASure:PCURrent query returns the measured peak current.

Return Format `<return_value><NL>`

```
<return_value> ::= the peak current value in NR3 format
```

See Also

- "[:MEASure:SOURce](#)" on page 468
- "[:POWer:INRush:APPLy](#)" on page 575

:MEASure:PLOSSs

N (see page 1232)

Command Syntax :MEASure:PLOSSs [<source>]

```
<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m> | WMEMory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :MEASure:PLOSSs 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 $P_n = V_{ds_n} * I_{d_n}$, where n is each sample.

Query Syntax :MEASure:PLOSSs? [<source>]

The :MEASure:PLOSSs 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 468
 - "[:POWER:SWITch:APPLY](#)" on page 612

:MEASure:REACTive

N (see page 1232)

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 $\text{apparent_power}^2 = \text{real_power}^2 + \text{reactive_power}^2$):

$$\text{Reactive Power} = \sqrt{\text{Apparent Power}^2 - \text{Real Power}^2}$$

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 468
• "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:REAL

N (see page 1232)

Command Syntax :MEASure:REAL [<source>]

```
<source> ::= {CHANnel<n>| FUNCtion<m> | MATH<m>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 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 468
 - "[:POWER:QUALITY:APPLY](#)" on page 589

:MEASure:RIPPle

N (see page 1232)

Command Syntax `:MEASure:RIPPle [<source>]`

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:RIPPle command installs an output ripple measurement on screen.

The <source> parameter is the channel probing the output voltage. This source can also be specified by the :MEASure:SOURce command.

Output ripple is: Vmax - Vmin.

Query Syntax `:MEASure:RIPPle? [<source>]`

The :MEASure:RIPPle query returns the measured output ripple.

Return Format `<return_value><NL>`

```
<return_value> ::= the output ripple value in NR3 format
```

- See Also**
- "[:MEASure:SOURce](#)" on page 468
 - "[:POWer:RIPPLE:APPLY](#)" on page 590

:MEASure:TRESPonse

N (see page 1232)

Command Syntax :MEASure:TRESPonse [<source>]

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

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

<m> ::= 1-4 in NR1 format

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

The :MEASure:TRESPonse command installs a transient response time measurement on screen.

The <source> parameter is the channel probing the output voltage. This source can also be specified by the :MEASure:SOURce command.

Transient response time = $t_2 - t_1$, where:

- t_1 = The first time a voltage waveform exits the settling band.
- t_2 = 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 468
- "[:POWER:TRANsient:APPLy](#)" on page 618

23 :MTEST Commands

The MTEST subsystem commands and queries control the mask test features. See "Introduction to :MTEST Commands" on page 519.

Table 98 :MTEST Commands Summary

Command	Query	Options and Query Returns
:MTEST:ALL {{0 OFF} {1 ON}} (see page 522)	:MTEST:ALL? (see page 522)	{0 1}
:MTEST:AMASK:CREATE (see page 523)	n/a	n/a
:MTEST:AMASK:SOURCE <source> (see page 524)	:MTEST:AMASK:SOURce? (see page 524)	<source> ::= CHANnel<n> <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
:MTEST:AMASK:UNITS <units> (see page 525)	:MTEST:AMASK:UNITS? (see page 525)	<units> ::= {CURRent DIVisions}
:MTEST:AMASK:XDELta <value> (see page 526)	:MTEST:AMASK:XDELta? (see page 526)	<value> ::= X delta value in NR3 format
:MTEST:AMASK:YDELta <value> (see page 527)	:MTEST:AMASK:YDELta? (see page 527)	<value> ::= Y delta value in NR3 format
n/a	:MTEST:COUNT:FWAVefor ms? [CHANnel<n>] (see page 528)	<failed> ::= number of failed waveforms in NR1 format
:MTEST:COUNT:RESet (see page 529)	n/a	n/a
n/a	:MTEST:COUNT:TIME? (see page 530)	<time> ::= elapsed seconds in NR3 format
n/a	:MTEST:COUNT:WAVEform s? (see page 531)	<count> ::= number of waveforms in NR1 format
:MTEST:DATA <mask> (see page 532)	:MTEST:DATA? (see page 532)	<mask> ::= data in IEEE 488.2 # format.



23 :MTEST Commands

Table 98 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:DELETE (see page 533)	n/a	n/a
:MTEST:ENABLE {{0 OFF} {1 ON}} (see page 534)	:MTEST:ENABLE? (see page 534)	{0 1}
:MTEST:LOCK {{0 OFF} {1 ON}} (see page 535)	:MTEST:LOCK? (see page 535)	{0 1}
:MTEST:RMODE <rmode> (see page 536)	:MTEST:RMODE? (see page 536)	<rmode> ::= {FORever TIME SIGMa WAVEforms}
:MTEST:RMODE:FACTion:MEASure {{0 OFF} {1 ON}} (see page 537)	:MTEST:RMODE:FACTion:MEASure? (see page 537)	{0 1}
:MTEST:RMODE:FACTion:PRINT {{0 OFF} {1 ON}} (see page 538)	:MTEST:RMODE:FACTion:PRINT? (see page 538)	{0 1}
:MTEST:RMODE:FACTion:SAVE {{0 OFF} {1 ON}} (see page 539)	:MTEST:RMODE:FACTion:SAVE? (see page 539)	{0 1}
:MTEST:RMODE:FACTion:STOP {{0 OFF} {1 ON}} (see page 540)	:MTEST:RMODE:FACTion:STOP? (see page 540)	{0 1}
:MTEST:RMODE:SIGMa <level> (see page 541)	:MTEST:RMODE:SIGMa? (see page 541)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTEST:RMODE:TIME <seconds> (see page 542)	:MTEST:RMODE:TIME? (see page 542)	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAVEforms <count> (see page 543)	:MTEST:RMODE:WAVEforms? (see page 543)	<count> ::= number of waveforms in NR1 format
:MTEST:SCALe:BIND {{0 OFF} {1 ON}} (see page 544)	:MTEST:SCALe:BIND? (see page 544)	{0 1}
:MTEST:SCALe:X1 <x1_value> (see page 545)	:MTEST:SCALe:X1? (see page 545)	<x1_value> ::= X1 value in NR3 format

Table 98 :MTEST Commands Summary (continued)

Command	Query	Options and Query Returns
:MTEST:SCALe:XDELta <xdelta_value> (see page 546)	:MTEST:SCALe:XDELta? (see page 546)	<xdelta_value> ::= X delta value in NR3 format
:MTEST:SCALe:Y1 <y1_value> (see page 547)	:MTEST:SCALe:Y1? (see page 547)	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALe:Y2 <y2_value> (see page 548)	:MTEST:SCALe:Y2? (see page 548)	<y2_value> ::= Y2 value in NR3 format
:MTEST:SOURce <source> (see page 549)	:MTEST:SOURce? (see page 549)	<source> ::= {CHANnel<n> NONE} <n> ::= {1 2 3 4} for 4ch models <n> ::= {1 2} for 2ch models
n/a	:MTEST:TITLe? (see page 550)	<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
+2.5000000E-001;YDEL +2.5000000E-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 ":OPERREGISTER: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 ":OPERREGISTER: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 1232)

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 519

:MTEST:AMASK:CREATE

N (see page 1232)

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 519
- ":MTEST:AMASK:XDELta" on page 526
- ":MTEST:AMASK:YDELta" on page 527
- ":MTEST:AMASK:UNITS" on page 525
- ":MTEST:AMASK:SOURce" on page 524
- ":MTEST:SOURce" on page 549

Example Code

- "Example Code" on page 519

:MTESt:AMASK:SOURce

N (see page 1232)

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 519
- "[:MTESt:AMASK:XDELta](#)" on page 526
- "[:MTESt:AMASK:YDELta](#)" on page 527
- "[:MTESt:AMASK:UNITS](#)" on page 525
- "[:MTESt:SOURce](#)" on page 549

Example Code

- "[Example Code](#)" on page 519

:MTESt:AMASK:UNITS

N (see page 1232)

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 519
 - ":MTEST:AMASK:XDELta" on page 526
 - ":MTEST:AMASK:YDELta" on page 527
 - ":CHANnel<n>:UNITS" on page 285
 - ":MTEST:AMASK:SOURce" on page 524
 - ":MTEST:SOURce" on page 549

Example Code

- "Example Code" on page 519

:MTESt:AMASK:XDELta

N (see page 1232)

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 519

- "[:MTESt:AMASK:UNITS](#)" on page 525
- "[:MTESt:AMASK:YDELta](#)" on page 527
- "[:MTESt:AMASK:SOURce](#)" on page 524
- "[:MTESt:SOURce](#)" on page 549

Example Code • "[Example Code](#)" on page 519

:MTEST:AMASK:YDELta

N (see page 1232)

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 519
- "[:MTEST:AMASK:UNITS](#)" on page 525
- "[:MTEST:AMASK:XDELta](#)" on page 526
- "[:MTEST:AMASK:SOURce](#)" on page 524
- "[:MTEST:SOURce](#)" on page 549

Example Code

- "["Example Code"](#) on page 519

:MTESt:COUNt:FWAVeforms

N (see page 1232)

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 519

- ":MTESt:COUNt:WAveforms" on page 531
- ":MTESt:COUNt:TIME" on page 530
- ":MTESt:COUNt:RESet" on page 529
- ":MTESt:SOURce" on page 549

Example Code • "Example Code" on page 519

:MTESt:COUNt:RESet

N (see page 1232)

Command Syntax :MTESt:COUNt:RESet

The :MTESt:COUNt:RESet command resets the mask statistics.

See Also

- "Introduction to :MTESt Commands" on page 519
- ":MTESt:COUNt:WAveforms" on page 531
- ":MTESt:COUNt:FWAveforms" on page 528
- ":MTESt:COUNt:TIME" on page 530

:MTESt:COUNt:TIME

N (see page 1232)

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 519
 - ":MTESt:COUNt:WAVeforms" on page 531
 - ":MTESt:COUNt:FWAVeforms" on page 528
 - ":MTESt:COUNt:RESet" on page 529

Example Code

- "Example Code" on page 519

:MTEST:COUNt:WAveforms

N (see page 1232)

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 519
 - ":MTEST:COUNt:FWAVEforms" on page 528
 - ":MTEST:COUNt:TIME" on page 530
 - ":MTEST:COUNt:RESet" on page 529

Example Code

- "Example Code" on page 519

:MTEST:DATA

N (see [page 1232](#))

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 645

• "[:RECall:MASK\[:STARt\]](#)" on page 628

:MTEST:DElete

N (see page 1232)

Command Syntax :MTEST:DElete

The :MTEST:DElete command clears the currently loaded mask.

See Also

- "Introduction to :MTEST Commands" on page 519
- ":MTEST:AMASK:CREATE" on page 523

:MTESt:ENABLE**N** (see page 1232)**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 519

:MTESt:LOCK

N (see page 1232)

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 519

• ":MTESt:SOURce" on page 549

:MTESt:RMODE

N (see page 1232)

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 541 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 542 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 543 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 519
- "[:MTESt:RMODE:SIGMa](#)" on page 541
- "[:MTESt:RMODE:TIME](#)" on page 542
- "[:MTESt:RMODE:WAVEforms](#)" on page 543

Example Code

- "[Example Code](#)" on page 519

:MTESt:RMODE:FACTion:MEASure

N (see page 1232)

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 519
- ":MTESt:RMODE:FACTion:PRINT" on page 538
- ":MTESt:RMODE:FACTion:SAVE" on page 539
- ":MTESt:RMODE:FACTion:STOP" on page 540

:MTESt:RMODE:FACTion:PRINt

N (see page 1232)

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 18, “:HARDcopy Commands,”](#) starting on page 375 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 519
- [“:MTESt:RMODE:FACTion:MEASure”](#) on page 537
- [“:MTESt:RMODE:FACTion:SAVE”](#) on page 539
- [“:MTESt:RMODE:FACTion:STOP”](#) on page 540

:MTEST:RMODE:FACTion:SAVE

N (see page 1232)

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 27, “:SAVE Commands,”](#) starting on page 633 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 519
- "[:MTEST:RMODE:FACTion:MEASure](#)" on page 537
- "[:MTEST:RMODE:FACTion:PRINT](#)" on page 538
- "[:MTEST:RMODE:FACTion:STOP](#)" on page 540

:MTESt:RMODE:FACTion:STOP

N (see page 1232)

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 519
• "[":MTEST:RMODE:FACTion:MEASure](#)" on page 537
• "[":MTEST:RMODE:FACTion:PRINT](#)" on page 538
• "[":MTEST:RMODE:FACTion:SAVE](#)" on page 539

:MTEST:RMODE:SIGMA

N (see page 1232)

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 519
 - ":MTEST:RMODE" on page 536

- Example Code**
- "Example Code" on page 519

:MTESt:RMODE:TIME

N (see page 1232)

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 519
- ":MTESt:RMODE" on page 536

:MTEST:RMODE:WAveforms

N (see page 1232)

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 519
- ":MTEST:RMODE" on page 536

:MTESt:SCALe:BIND

N (see page 1232)

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 519
 - "[":MTESt:SCALe:X1"](#) on page 545
 - "[":MTESt:SCALe:XDELta"](#) on page 546
 - "[":MTESt:SCALe:Y1"](#) on page 547
 - "[":MTESt:SCALe:Y2"](#) on page 548

:MTEST:SCALe:X1

N (see page 1232)

Command Syntax :MTEST:SCALe:X1 <x1_value>

<x1_value> ::= X1 value in NR3 format

The :MTEST:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the :MTEST:SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$$X = (X * \Delta X) + X1$$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

The X1 value is a time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Query Syntax :MTEST:SCALe:X1?

The :MTEST:SCALe:X1? query returns the current X1 coordinate setting.

Return Format <x1_value><NL>

<x1_value> ::= X1 value in NR3 format

- See Also**
- "Introduction to :MTEST Commands" on page 519
 - ":MTEST:SCALe:BIND" on page 544
 - ":MTEST:SCALe:XDELta" on page 546
 - ":MTEST:SCALe:Y1" on page 547
 - ":MTEST:SCALe:Y2" on page 548

:MTESt:SCALe:XDELta

N (see page 1232)

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:

$$X = (X * \Delta X) + X1$$

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 519
 - ":MTESt:SCALe:BIND" on page 544
 - ":MTESt:SCALe:X1" on page 545
 - ":MTESt:SCALe:Y1" on page 547
 - ":MTESt:SCALe:Y2" on page 548

:MTEST:SCALe:Y1

N (see page 1232)

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 519
- ":MTEST:SCALe:BIND" on page 544
- ":MTEST:SCALe:X1" on page 545
- ":MTEST:SCALe:XDELta" on page 546
- ":MTEST:SCALe:Y2" on page 548

:MTESt:SCALe:Y2

N (see [page 1232](#))

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 519
 - "[:MTESt:SCALe:BIND](#)" on page 544
 - "[:MTESt:SCALe:X1](#)" on page 545
 - "[:MTESt:SCALe:XDELta](#)" on page 546
 - "[:MTESt:SCALe:Y1](#)" on page 547

:MTEST:SOURce

N (see page 1232)

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 519
 - ":MTEST:AMASK:SOURce" on page 524

:MTESt:TITLE

N (see [page 1232](#))

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 519

24 :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "[Introduction to :POD<n> Commands](#)" on page 551.

Table 99 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 OFF} {1 ON}} (see page 553)	:POD<n>:DISPlay? (see page 553)	{0 1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see page 554)	:POD<n>:SIZE? (see page 554)	<value> ::= {SMALL MEDIUM LARGE}
:POD<n>:THreshold <type>[suffix] (see page 555)	:POD<n>:THreshold? (see page 555)	<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.



Agilent Technologies

24 :POD Commands

```
:POD1:DISP 0;THR +1.40E+00
```

:POD<n>:DISPlay

N (see page 1232)

Command Syntax	<code>:POD<n>:DISPlay <display></code>
	<code><display> ::= {{1 ON} {0 OFF}}</code>
	<code><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.</code>
	<code>POD1 ::= D0-D7</code>
	<code>POD2 ::= D8-D15</code>
	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	<code>:POD<n>:DISPlay?</code>
	The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

Return Format	<code><display><NL></code>
	<code><display> ::= {0 1}</code>

See Also	<ul style="list-style-type: none"> • "Introduction to :POD<n> Commands" on page 551 • ":DIGItal<d>:DISPlay" on page 309 • ":CHANnel<n>:DISPlay" on page 272 • ":VIEW" on page 230 • ":BLANK" on page 202 • ":STATus" on page 227
-----------------	--

:POD<n>:SIZE

N (see [page 1232](#))

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 551
 - "[:DIGital<d>:SIZE](#)" on page 312
 - "[:DIGital<d>:POStion](#)" on page 311

:POD<n>:THreshold

N (see page 1232)

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 551
- "[:DIGItal<d>:THreshold](#)" on page 313
- "[:TRIGger\[:EDGE\]:LEVel](#)" on page 986

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 41, “Programming Examples,”](#) starting on page 1241

25 :POWer Commands

These :POWer commands are available when the DSOX4PWR power measurements and analysis application is licensed and enabled.

Table 100 :POWer Commands Summary

Command	Query	Options and Query Returns
:POWer:DESKew (see page 562)	n/a	n/a
:POWer:EFFiciency:APP Ly (see page 563)	n/a	n/a
:POWer:ENABLE {{0 OFF} {1 ON}} (see page 564)	:POWer:ENABLE? (see page 564)	{0 1}
:POWer:HARMonics:APPL y (see page 565)	n/a	n/a
n/a	:POWer:HARMonics:DATA ? (see page 566)	<binary_block> ::= comma-separated data with newlines at the end of each row
:POWer:HARMonics:DISPlay <display> (see page 567)	:POWer:HARMonics:DISPlay? (see page 567)	<display> ::= {TABLE BAR OFF}
n/a	:POWer:HARMonics:FAIL count? (see page 568)	<count> ::= integer in NR1 format
:POWer:HARMonics:LINE <frequency> (see page 569)	:POWer:HARMonics:LINE ? (see page 569)	<frequency> ::= {F50 F60 F400}
n/a	:POWer:HARMonics:POWe rfactor? (see page 570)	<value> ::= Class C power factor in NR3 format
n/a	:POWer:HARMonics:RUNC ount? (see page 571)	<count> ::= integer in NR1 format
:POWer:HARMonics:STAN dard <class> (see page 572)	:POWer:HARMonics:STAN dard? (see page 572)	<class> ::= {A B C D}



Table 100 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:POWer:HARMonics:STATus? (see page 573)	<status> ::= {PASS FAIL UNTested}
n/a	:POWer:HARMonics:THD? (see page 574)	<value> ::= Total Harmonics Distortion in NR3 format
:POWer:INRush:APPLy (see page 575)	n/a	n/a
:POWer:INRush:EXIT (see page 576)	n/a	n/a
:POWer:INRush:NEXT (see page 577)	n/a	n/a
:POWer:MODulation:APPly (see page 578)	n/a	n/a
:POWer:MODulation:SOURce <source> (see page 579)	:POWer:MODulation:SOURce? (see page 579)	<source> ::= {V I}
:POWer:MODulation:TYPe <modulation> (see page 580)	:POWer:MODulation:TYPe? (see page 580)	<modulation> ::= {VAverage ACRMs VRATio PERiod FREQuency PWIDith NWIDth DUTYcycle RISetime FALLtime}
:POWer:ONOFF:APPLy (see page 581)	n/a	n/a
:POWer:ONOFF:EXIT (see page 582)	n/a	n/a
:POWer:ONOFF:NEXT (see page 583)	n/a	n/a
:POWer:ONOFF:TEST {{0 OFF} {1 ON}} (see page 584)	:POWer:ONOFF:TEST? (see page 584)	{0 1}
:POWer:PSRR:APPLy (see page 585)	n/a	n/a
:POWer:PSRR:FREQuency :MAXimum <value>[suffix] (see page 586)	:POWer:PSRR:FREQuency :MAXimum? (see page 586)	<value> ::= {10 100 1000 10000 100000 1000000 10000000 20000000} [suffix] ::= {Hz kHz MHz}
:POWer:PSRR:FREQuency :MINimum <value>[suffix] (see page 587)	:POWer:PSRR:FREQuency :MINimum? (see page 587)	<value> ::= {1 10 100 1000 10000 100000 1000000 10000000} [suffix] ::= {Hz kHz MHz}

Table 100 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:PSRR:RMAXimum <value> (see page 588)	:POWer:PSRR:RMAXimum? (see page 588)	<value> ::= Maximum ratio value in NR1 format
:POWer:QUALity:APPLy (see page 589)	n/a	n/a
:POWer:RIPPLe:APPLy (see page 590)	n/a	n/a
:POWer:SIGNals:AUTose tup <analysis> (see page 591)	n/a	<analysis> ::= {HARMonics EFFiciency RIPPLe MODulation QUALity SLEW SWITch}
:POWer:SIGNals:CYCLeS :HARMonics <count> (see page 592)	:POWer:SIGNals:CYCLeS :HARMonics? (see page 592)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:CYCLeS :QUALity <count> (see page 593)	:POWer:SIGNals:CYCLeS :QUALity? (see page 593)	<count> ::= integer in NR1 format Legal values are 1 to 100.
:POWer:SIGNals:DURati on:EFFiciency <value>[suffix] (see page 594)	:POWer:SIGNals:DURati on:EFFiciency? (see page 594)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:MODulation <value>[suffix] (see page 595)	:POWer:SIGNals:DURati on:MODulation? (see page 595)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:ONOFF:OFF <value>[suffix] (see page 596)	:POWer:SIGNals:DURati on:ONOFF:OFF? (see page 596)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:ONOFF:ON <value>[suffix] (see page 597)	:POWer:SIGNals:DURati on:ONOFF:ON? (see page 597)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:RIPPLe <value>[suffix] (see page 598)	:POWer:SIGNals:DURati on:RIPPLe? (see page 598)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}
:POWer:SIGNals:DURati on:TRANSient <value>[suffix] (see page 599)	:POWer:SIGNals:DURati on:TRANSient? (see page 599)	<value> ::= value in NR3 format [suffix] ::= {s ms us ns}

25 :POWer Commands

Table 100 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SIGNals:IEXPected <value>[suffix] (see page 600)	:POWer:SIGNals:IEXPected? (see page 600)	<value> ::= Expected current value in NR3 format [suffix] ::= {A mA}
:POWer:SIGNals:OVERshoot <percent> (see page 601)	:POWer:SIGNals:OVERshoot? (see page 601)	<percent> ::= percent of overshoot value in NR1 format [suffix] ::= {V mV}
:POWer:SIGNals:VMAXimum:INRush <value>[suffix] (see page 602)	:POWer:SIGNals:VMAXimum:INRush? (see page 602)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VMAXimum:ONOFF:OFF <value>[suffix] (see page 603)	:POWer:SIGNals:VMAXimum:ONOFF:OFF? (see page 603)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VMAXimum:ONOFF:ON <value>[suffix] (see page 604)	:POWer:SIGNals:VMAXimum:ONOFF:ON? (see page 604)	<value> ::= Maximum expected input Voltage in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTeady:ONOFF:OFF <value>[suffix] (see page 605)	:POWer:SIGNals:VSTeady:ONOFF:OFF? (see page 605)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTeady:ONOFF:ON <value>[suffix] (see page 606)	:POWer:SIGNals:VSTeady:ONOFF:ON? (see page 606)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:VSTeady:TRANSient <value>[suffix] (see page 607)	:POWer:SIGNals:VSTeady:TRANSient? (see page 607)	<value> ::= Expected steady stage output Voltage value in NR3 format [suffix] ::= {V mV}
:POWer:SIGNals:SOURce:CURRent<i> <source> (see page 608)	:POWer:SIGNals:SOURce:CURRent<i>? (see page 608)	<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 609)	:POWer:SIGNals:SOURce:VOLTage<i>? (see page 609)	<i> ::= 1, 2 in NR1 format <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:POWer:SLEW:APPLY (see page 610)	n/a	n/a

Table 100 :POWer Commands Summary (continued)

Command	Query	Options and Query Returns
:POWer:SLEW:SOURce <source> (see page 611)	:POWer:SLEW:SOURce? (see page 611)	<source> ::= {V I}
:POWer:SWITch:APPLy (see page 612)	n/a	n/a
:POWer:SWITch:CONDuct ion <conduction> (see page 613)	:POWer:SWITch:CONDuct ion? (see page 613)	<conduction> ::= {WAveform RDS VCE}
:POWer:SWITch:IREFere nce <percent> (see page 614)	:POWer:SWITch:IREFere nce? (see page 614)	<percent> ::= percent in NR1 format
:POWer:SWITch:RDS <value>[suffix] (see page 615)	:POWer:SWITch:RDS? (see page 615)	<value> ::= Rds(on) value in NR3 format [suffix] ::= {OHM mOHM}
:POWer:SWITch:VCE <value>[suffix] (see page 616)	:POWer:SWITch:VCE? (see page 616)	<value> ::= Vce(sat) value in NR3 format [suffix] ::= {V mV}
:POWer:SWITch:VREFere nce <percent> (see page 617)	:POWer:SWITch:VREFere nce? (see page 617)	<percent> ::= percent in NR1 format
:POWer:TRANSient:APPL y (see page 618)	n/a	n/a
:POWer:TRANSient:EXIT (see page 619)	n/a	n/a
:POWer:TRANSient:IINI tial <value>[suffix] (see page 620)	:POWer:TRANSient:IINI tial? (see page 620)	<value> ::= Initial current value in NR3 format [suffix] ::= {A mA}
:POWer:TRANSient:INEW <value>[suffix] (see page 621)	:POWer:TRANSient:INEW ? (see page 621)	<value> ::= New current value in NR3 format [suffix] ::= {A mA}
:POWer:TRANSient:NEXT (see page 622)	n/a	n/a

:POWer:DESKew

N (see page 1232)

Command Syntax :POWer:DESKew

The :POWer:DESKew command launches the auto deskew process on the oscilloscope.

Before sending this command:

- 1 Demagnetize and zero-adjust the current probe.
Refer to the current probe's documentation for instructions on how to do this.
- 2 Make connections to the U1880A deskew fixture as described in the oscilloscope's connection dialog or in the *DSOX4PWR Power Measurement Application User's Guide*.
- 3 Make sure the voltage probe and current probe channels are specified appropriately using the :POWer:SIGNals:SOURce:VOLTage1 and :POWer:SIGNals:SOURce:CURRent1 commands.

NOTE

Use the lowest attenuation setting on the high voltage differential probes whenever possible because the voltage levels on the deskew fixture are very small. Using a higher attenuation setting may yield inaccurate skew values (and affect the measurements made) because the noise level is magnified as well.

The deskew values are saved in the oscilloscope until a factory default or secure erase is performed. The next time you run the Power Application, you can use the saved deskew values or perform the deskew again.

Generally, you need to perform the deskew again when part of the test setup changes (for example, a different probe, different oscilloscope channel, etc.) or when the ambient temperature has changed.

See Also

- "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609
- "[:POWer:SIGNals:SOURce:CURRent<i>](#)" on page 608

:POWer:EFFiciency:APPLy

N (see page 1232)

Command Syntax :POWer:EFFiciency:APPLy

The :POWer:EFFiciency:APPLy command applies the efficiency power analysis.

Efficiency analysis tests the overall efficiency of the power supply by measuring the output power over the input power.

NOTE

Efficiency analysis requires a 4-channel oscilloscope because input voltage, input current, output voltage, and output current are measured.

See Also

- "[:MEASure:EFFiciency](#)" on page 503
- "[:MEASure:IPOWer](#)" on page 506
- "[:MEASure:OPOWer](#)" on page 509

:POWer:ENABLE

N (see [page 1232](#))

Command Syntax :POWer:ENABLE {{0 | OFF} | {1 | ON}}

The :POWer:ENABLE command enables or disables power analysis.

Query Syntax :POWer:ENABLE?

The :POWer:ENABLE query returns a 1 or a 0 showing whether power analysis is enabled or disabled, respectively.

Return Format {0 | 1}

See Also • [Chapter 22, “:MEASure Power Commands,” starting on page 495](#)

:POWer:HARMonics:APPLy

N (see page 1232)

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 566
- "[:POWer:HARMonics:DISPlay](#)" on page 567
- "[:POWer:HARMonics:FAILcount](#)" on page 568
- "[:POWer:HARMonics:LINE](#)" on page 569
- "[:POWer:HARMonics:POWERfactor](#)" on page 570
- "[:POWer:HARMonics:STANDARD](#)" on page 572
- "[:POWer:HARMonics:STATUS](#)" on page 573
- "[:POWer:HARMonics:RUNCount](#)" on page 571
- "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:DATA

N (see page 1232)

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 565
 - "[:POWer:HARMonics:DISPlay](#)" on page 567
 - "[:POWer:HARMonics:FAILcount](#)" on page 568
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:POWERfactor](#)" on page 570
 - "[:POWer:HARMonics:RUNCount](#)" on page 571
 - "[:POWer:HARMonics:STANDARD](#)" on page 572
 - "[:POWer:HARMonics:STATUS](#)" on page 573
 - "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:DISPlay

N (see page 1232)

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 565
 • "[:POWer:HARMonics:DATA](#)" on page 566
 • "[:POWer:HARMonics:FAILcount](#)" on page 568
 • "[:POWer:HARMonics:LINE](#)" on page 569
 • "[:POWer:HARMonics:POWERfactor](#)" on page 570
 • "[:POWer:HARMonics:RUNCount](#)" on page 571
 • "[:POWer:HARMonics:STANDARD](#)" on page 572
 • "[:POWer:HARMonics:STATUS](#)" on page 573
 • "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:FAILcount

(see page 1232)

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 571
 - "`:POWer:HARMonics:APPLy`" on page 565
 - "`:POWer:HARMonics:DATA`" on page 566
 - "`:POWer:HARMonics:DISPLAY`" on page 567
 - "`:POWer:HARMonics:LINE`" on page 569
 - "`:POWer:HARMonics:POWERfactor`" on page 570
 - "`:POWer:HARMonics:STANDARD`" on page 572
 - "`:POWer:HARMonics:STATUS`" on page 573
 - "`:POWer:HARMonics:THD`" on page 574

:POWer:HARMonics:LINE

N (see page 1232)

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 565
- "[:POWer:HARMonics:DATA](#)" on page 566
- "[:POWer:HARMonics:DISPlay](#)" on page 567
- "[:POWer:HARMonics:FAILcount](#)" on page 568
- "[:POWer:HARMonics:POWERfactor](#)" on page 570
- "[:POWer:HARMonics:RUNCount](#)" on page 571
- "[:POWer:HARMonics:STANDARD](#)" on page 572
- "[:POWer:HARMonics:STATUS](#)" on page 573
- "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:POWerfactor

N (see page 1232)

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 565
 - "[:POWer:HARMonics:DATA](#)" on page 566
 - "[:POWer:HARMonics:DISPLAY](#)" on page 567
 - "[:POWer:HARMonics:FAILcount](#)" on page 568
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:RUNCount](#)" on page 571
 - "[:POWer:HARMonics:STANDARD](#)" on page 572
 - "[:POWer:HARMonics:STATUS](#)" on page 573
 - "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:RUNCount

N (see page 1232)

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 568
 - "[:POWer:HARMonics:APPLy](#)" on page 565
 - "[:POWer:HARMonics:DATA](#)" on page 566
 - "[:POWer:HARMonics:DISPlay](#)" on page 567
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:POWERfactor](#)" on page 570
 - "[:POWer:HARMonics:STANDARD](#)" on page 572
 - "[:POWer:HARMonics:STATUS](#)" on page 573
 - "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:STANdard

(see page 1232)

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 565
 - "[:POWer:HARMonics:DATA](#)" on page 566
 - "[:POWer:HARMonics:DISPLAY](#)" on page 567
 - "[:POWer:HARMonics:FAILcount](#)" on page 568
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:POWERfactor](#)" on page 570
 - "[:POWer:HARMonics:RUNCount](#)" on page 571
 - "[:POWer:HARMonics:STATus](#)" on page 573
 - "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:STATus

N (see page 1232)

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 571
 - "[:POWer:HARMonics:FAILcount](#)" on page 568
 - "[:POWer:HARMonics:APPLy](#)" on page 565
 - "[:POWer:HARMonics:DATA](#)" on page 566
 - "[:POWer:HARMonics:DISPLAY](#)" on page 567
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:POWERfactor](#)" on page 570
 - "[:POWer:HARMonics:STANDARD](#)" on page 572
 - "[:POWer:HARMonics:THD](#)" on page 574

:POWer:HARMonics:THD

N (see page 1232)

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 565
 - "[:POWer:HARMonics:DATA](#)" on page 566
 - "[:POWer:HARMonics:DISPLAY](#)" on page 567
 - "[:POWer:HARMonics:FAILcount](#)" on page 568
 - "[:POWer:HARMonics:LINE](#)" on page 569
 - "[:POWer:HARMonics:POWERfactor](#)" on page 570
 - "[:POWer:HARMonics:RUNCount](#)" on page 571
 - "[:POWer:HARMonics:STANDARD](#)" on page 572
 - "[:POWer:HARMonics:STATUS](#)" on page 573

:POWer:INRush:APPLy

N (see page 1232)

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 510
 - "[:POWer:INRush:EXIT](#)" on page 576
 - "[:POWer:INRush:NEXT](#)" on page 577

:POWer:INRush:EXIT

N (see [page 1232](#))

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 575
- "[:POWer:INRush:NEXT](#)" on page 577

:POWer:INRush:NEXT

N (see page 1232)

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 575
- "[:POWer:INRush:EXIT](#)" on page 576

:POWer:MODulation:APPLy

N (see page 1232)

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 579
 - "[:POWer:MODulation:TYPE](#)" on page 580
 - "[:MEASure:VAVerage](#)" on page 482
 - "[:MEASure:VRMS](#)" on page 488
 - "[:MEASure:VRATio](#)" on page 487
 - "[:MEASure:PERiod](#)" on page 457
 - "[:MEASure:FREQuency](#)" on page 449
 - "[:MEASure:PWIDth](#)" on page 461
 - "[:MEASure:NWIDth](#)" on page 453
 - "[:MEASure:DUTYcycle](#)" on page 447
 - "[:MEASure:RISetime](#)" on page 465
 - "[:MEASure:FALLtime](#)" on page 448

:POWer:MODulation:SOURce

N (see page 1232)

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 578
• "[:POWer:MODulation:TYPE](#)" on page 580

:POWer:MODulation:TYPE

N (see page 1232)

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
- PWIDith (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 579
- "[:POWer:MODulation:APPLy](#)" on page 578
- "[:MEASure:VAVerage](#)" on page 482
- "[:MEASure:VRMS](#)" on page 488
- "[:MEASure:VRATio](#)" on page 487
- "[:MEASure:PERiod](#)" on page 457
- "[:MEASure:FREQuency](#)" on page 449
- "[:MEASure:PWIDth](#)" on page 461
- "[:MEASure:NWIDth](#)" on page 453
- "[:MEASure:DUTYcycle](#)" on page 447
- "[:MEASure:RISetime](#)" on page 465
- "[:MEASure:FALLtime](#)" on page 448

:POWer:ONOFF:APPLy

N (see page 1232)

Command Syntax

:POWer:ONOFF:APPLy

The :POWer:ONOFF:APPLy command applies the selected turn on/off analysis test (:POWer:ONOFF:TEST).

See Also

- "[:POWer:SIGNals:VSteady:ONOFF:OFF](#)" on page 605
- "[:POWer:SIGNals:VSteady:ONOFF:ON](#)" on page 606
- "[:MEASure:ONTIme](#)" on page 508
- "[:MEASure:OFFTime](#)" on page 507
- "[:POWer:ONOFF:TEST](#)" on page 584
- "[:POWer:ONOFF:EXIT](#)" on page 582
- "[:POWer:ONOFF:NEXT](#)" on page 583

:POWer:ONOFF:EXIT

N (see page 1232)

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 581
 - "[:POWer:ONOFF:NEXT](#)" on page 583
 - "[:POWer:ONOFF:TEST](#)" on page 584

:POWer:ONOFF:NEXT

N (see page 1232)

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 581
 - "[:POWer:ONOFF:EXIT](#)" on page 582
 - "[:POWer:ONOFF:TEST](#)" on page 584

:POWer:ONOFF:TEST

N (see page 1232)

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 581
- "[:POWer:ONOFF:EXIT](#)" on page 582
- "[:POWer:ONOFF:NEXT](#)" on page 583

:POWer:PSRR:APPLy

N (see page 1232)

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 586
- "[:POWer:PSRR:FREQuency:MINimum](#)" on page 587
- "[:POWer:PSRR:RMAXimum](#)" on page 588

:POWer:PSRR:FREQuency:MAXimum

N (see page 1232)

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 585
 - "[:POWer:PSRR:FREQuency:MINimum](#)" on page 587
 - "[:POWer:PSRR:RMAXimum](#)" on page 588

:POWer:PSRR:FREQuency:MINimum

N (see page 1232)

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 585
- "[:POWer:PSRR:FREQuency:MAXimum](#)" on page 586
- "[:POWer:PSRR:RMAXimum](#)" on page 588

:POWer:PSRR:RMAXimum

N (see page 1232)

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 588
 - "[:POWer:PSRR:FREQuency:MAXimum](#)" on page 586
 - "[:POWer:PSRR:FREQuency:MINimum](#)" on page 587

:POWer:QUALity:APPLy

N (see page 1232)

Command Syntax :POWer:QUALity:APPLy

The :POWer:QUALity:APPLy command applies the selected power quality analysis type (:POWer:QUALity:TYPE).

The power quality analysis shows the quality of the AC input line.

Some AC current may flow back into and back out of the load without delivering energy. This current, called reactive or harmonic current, gives rise to an "apparent" power which is larger than the actual power consumed. Power quality is gauged by these measurements: power factor, apparent power, true power, reactive power, crest factor, and phase angle of the current and voltage of the AC line.

- See Also**
- "[:MEASure:FACTOr](#)" on page 505
 - "[:MEASure:REAL](#)" on page 513
 - "[:MEASure:APPARENT](#)" on page 500
 - "[:MEASure:REACTive](#)" on page 512
 - "[:MEASure:CRESt](#)" on page 502
 - "[:MEASure:ANGLE](#)" on page 499

:POWer:RIPPLe:APPLy

N (see page 1232)

Command Syntax `:POWer:RIPPLe:APPLy`

The `:POWer:RIPPLe:APPLy` command applies the output ripple analysis.

See Also • " [":MEASure:RIPPLe](#)" on page 514

:POWer:SIGNals:AUTosetup

N (see page 1232)

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 567
- "[:POWer:EFFiciency:APPLY](#)" on page 563
- "[:POWer:RIPPLE:APPLY](#)" on page 590
- "[:POWer:MODulation:APPLY](#)" on page 578
- "[:POWer:QUALity:APPLY](#)" on page 589
- "[:POWer:SLEW:APPLY](#)" on page 610
- "[:POWer:SWITch:APPLY](#)" on page 612
- "[:POWer:SIGNals:CYCles:HARMonics](#)" on page 592
- "[:POWer:SIGNals:CYCles:QUALity](#)" on page 593
- "[:POWer:SIGNals:DURation:EFFiciency](#)" on page 594
- "[:POWer:SIGNals:DURation:MODulation](#)" on page 595
- "[:POWer:SIGNals:DURation:RIPPLE](#)" on page 598
- "[:POWer:SIGNals:IEXPected](#)" on page 600
- "[:POWer:SIGNals:OVERshoot](#)" on page 601
- "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
- "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:CYCLes:HARMonics

N (see page 1232)

Command Syntax :POWer:SIGNals:CYCLes:HARMonics <count>

<count> ::= integer in NR1 format

Legal values are 1 to 100.

The :POWer:SIGNals:CYCLes:HARMonics command specifies the number of cycles to include in the current harmonics analysis.

Query Syntax :POWer:SIGNals:CYCLes:HARMonics?

The :POWer:SIGNals:CYCLes:HARMonics query returns the number of cycles currently set.

Return Format <count><NL>

<count> ::= integer in NR1 format

- See Also**
- "[:POWer:HARMonics:DISPLAY](#)" on page 567
 - "[:POWer:HARMonics:APPLy](#)" on page 565
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:CYCLes:QUALity

N (see page 1232)

Command Syntax

```
:POWer:SIGNals:CYCLes:QUALity <count>
<count> ::= integer in NR1 format
Legal values are 1 to 100.
```

The :POWer:SIGNals:CYCLes:QUALity command specifies the number of cycles to include in the power quality analysis.

Query Syntax

```
:POWer:SIGNals:CYCLes:QUALity?
```

The :POWer:SIGNals:CYCLes:QUALity query returns the number of cycles currently set.

Return Format

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

See Also

- "[:POWer:QUALity:APPLy](#)" on page 589
- "[:POWer:SIGNals:AUTosetup](#)" on page 591
- "[:POWer:SIGNals:IEXPected](#)" on page 600
- "[:POWer:SIGNals:OVERshoot](#)" on page 601
- "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
- "[:POWer:SIGNals:SOURee:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:EFFiciency

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:EFFiciency <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:EFFiciency command specifies the duration of the efficiency analysis.

Query Syntax :POWer:SIGNals:DURation:EFFiciency?

The :POWer:SIGNals:DURation:EFFiciency query returns the set duration time value.

Return Format <value><NL>

<value> ::= value in NR3 format

See Also

- "[:POWer:EFFiciency:APPLY](#)" on page 563
- "[:POWer:SIGNals:AUTosetup](#)" on page 591
- "[:POWer:SIGNals:IEXPected](#)" on page 600
- "[:POWer:SIGNals:OVERshoot](#)" on page 601
- "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
- "[:POWer:SIGNals:SOURee:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:MODulation

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:MODulation <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:MODulation command specifies the duration of the modulation analysis.

Query Syntax :POWer:SIGNals:DURation:MODulation?

The :POWer:SIGNals:DURation:MODulation query returns the set duration time value.

Return Format <value><NL>

<value> ::= value in NR3 format

See Also

- "[:POWer:MODulation:APPLy](#)" on page 578
- "[:POWer:SIGNals:AUTosetup](#)" on page 591
- "[:POWer:SIGNals:IEXPected](#)" on page 600
- "[:POWer:SIGNals:OVERshoot](#)" on page 601
- "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
- "[:POWer:SIGNals:SOURee:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:ONOFF:OFF

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:ONOFF:OFF <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:ONOFF:OFF command specifies the duration of the turn off analysis.

Query Syntax :POWer:SIGNals:DURation:ONOFF:OFF?

The :POWer:SIGNals:DURation:ONOFF:OFF query returns the set duration time value.

Return Format <value><NL>

<value> ::= value in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLy](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VMAXimum:ONOFF:OFF](#)" on page 603
 - "[:POWer:SIGNals:VSTeady:ONOFF:OFF](#)" on page 605
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:ONOFF:ON

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:ONOFF:ON <value>[suffix]
 <value> ::= value in NR3 format
 [suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:ONOFF:ON command specifies the duration of the turn on analysis.

Query Syntax :POWer:SIGNals:DURation:ONOFF:ON?

The :POWer:SIGNals:DURation:ONOFF:ON query returns the set duration time value.

Return Format <value><NL>
 <value> ::= value in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLy](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VMAXimum:ONOFF:ON](#)" on page 604
 - "[:POWer:SIGNals:VSTeady:ONOFF:ON](#)" on page 606
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:RIPPle

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:RIPPle <value>[suffix]

<value> ::= value in NR3 format

[suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:RIPPle command specifies the duration of the output ripple analysis.

Query Syntax :POWer:SIGNals:DURation:RIPPle?

The :POWer:SIGNals:DURation:RIPPle query returns the set duration time value.

Return Format <value><NL>

<value> ::= value in NR3 format

- See Also**
- "[:POWer:RIPPLE:APPLy](#)" on page 590
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURee:VOLTage<i>](#)" on page 609

:POWer:SIGNals:DURation:TRANsient

N (see page 1232)

Command Syntax :POWer:SIGNals:DURation:TRANsient <value>[suffix]
 <value> ::= value in NR3 format
 [suffix] ::= {s | ms | us | ns}

The :POWer:SIGNals:DURation:TRANsient command specifies the duration of the transient response analysis.

Query Syntax :POWer:SIGNals:DURation:TRANsient?

The :POWer:SIGNals:DURation:TRANsient query returns the set duration time value.

Return Format <value><NL>
 <value> ::= value in NR3 format

- See Also**
- "[:POWer:TRANsient:APPLy](#)" on page 618
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VSTeady:TRANsient](#)" on page 607
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:IEXPected

N (see [page 1232](#))

Command Syntax	<code>:POWer:SIGNals:IEXPected <value>[suffix]</code>
	<code><value></code> ::= Expected current value in NR3 format
	<code>[suffix]</code> ::= {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	<code>:POWer:SIGNals:IEXPected?</code>
	The :POWer:SIGNals:IEXPected query returns the expected inrush current setting.

Return Format	<code><value><NL></code>
	<code><value></code> ::= Expected current value in NR3 format

See Also	<ul style="list-style-type: none"> • ":POWer:INRush:APPLy" on page 575 • ":POWer:SIGNals:AUTosetup" on page 591 • ":POWer:SIGNals:OVERshoot" on page 601 • ":POWer:SIGNals:VMAXimum:INRush" on page 602 • ":POWer:SIGNals:SOURce:CURRent<i>" on page 608 • ":POWer:SIGNals:SOURce:VOLTage<i>" on page 609
-----------------	---

:POWer:SIGNals:OVERshoot

N (see page 1232)

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 618
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:TRANsient](#)" on page 599
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:VSTeady:TRANsient](#)" on page 607
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VMAXimum:INRush

N (see page 1232)

Command Syntax :POWer:SIGNals:VMAXimum:INRush <value>[suffix]

<value> ::= Maximum expected input Voltage in NR3 format

[suffix] ::= {V | mV}

The :POWer:SIGNals:VMAXimum:INRush command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for inrush current analysis.

Query Syntax :POWer:SIGNals:VMAXimum:INRush?

The :POWer:SIGNals:VMAXimum:INRush query returns the expected maximum input voltage setting.

Return Format <value><NL>

<value> ::= Maximum expected input Voltage in NR3 format

- See Also**
- "[:POWer:INRush:APPLy](#)" on page 575
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:SOURce:CURRent<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VMAXimum:ONOFF:OFF

N (see page 1232)

Command Syntax :POWer:SIGNals:VMAXimum:ONOFF:OFF <value>[suffix]

<value> ::= Maximum expected input Voltage in NR3 format

[suffix] ::= {V | mV}

The :POWer:SIGNals:VMAXimum:ONOFF:OFF command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for turn off analysis.

Query Syntax :POWer:SIGNals:VMAXimum:ONOFF:OFF?

The :POWer:SIGNals:VMAXimum:ONOFF:OFF query returns the expected maximum input voltage setting.

Return Format <value><NL>

<value> ::= Maximum expected input Voltage in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLY](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:ONOFF:OFF](#)" on page 596
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VSTeady:ONOFF:OFF](#)" on page 605
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VMAXimum:ONOFF:ON

N (see page 1232)

Command Syntax :POWer:SIGNals:VMAXimum:ONOFF:ON <value>[suffix]

<value> ::= Maximum expected input Voltage in NR3 format

[suffix] ::= {V | mV}

The :POWer:SIGNals:VMAXimum:ONOFF:ON command specifies the maximum expected input voltage. This value is used to set the vertical scale of the channel probing voltage for turn on analysis.

Query Syntax :POWer:SIGNals:VMAXimum:ONOFF:ON?

The :POWer:SIGNals:VMAXimum:ONOFF:ON query returns the expected maximum input voltage setting.

Return Format <value><NL>

<value> ::= Maximum expected input Voltage in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLY](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:ONOFF:ON](#)" on page 597
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VSTeady:ONOFF:ON](#)" on page 606
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VSteady:ONOFF:OFF

N (see page 1232)

Command Syntax :POWer:SIGNals:VSteady:ONOFF:OFF <value>[suffix]
 <value> ::= Expected steady state output Voltage value in NR3 format
 [suffix] ::= {V | mV}

The :POWer:SIGNals:VSteady:ONOFF:OFF command specifies the expected steady state output DC voltage of the power supply for turn off analysis.

Query Syntax :POWer:SIGNals:VSteady:ONOFF:OFF?

The :POWer:SIGNals:VSteady:ONOFF:OFF query returns the expected steady state voltage setting.

Return Format <value><NL>
 <value> ::= Expected steady state output Voltage value in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLy](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:ONOFF:OFF](#)" on page 596
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VMAXimum:ONOFF:OFF](#)" on page 603
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VSteady:ONOFF:ON

N (see page 1232)

Command Syntax `:POWer:SIGNals:VSteady:ONOFF:ON <value>[suffix]`

`<value>` ::= Expected steady state output Voltage value in NR3 format

`[suffix]` ::= {V | mV}

The :POWer:SIGNals:VSteady:ONOFF:ON command specifies the expected steady state output DC voltage of the power supply for turn on analysis.

Query Syntax `:POWer:SIGNals:VSteady:ONOFF:ON?`

The :POWer:SIGNals:VSteady:ONOFF:ON query returns the expected steady state voltage setting.

Return Format `<value><NL>`

`<value>` ::= Expected steady state output Voltage value in NR3 format

- See Also**
- "[:POWer:ONOFF:APPLy](#)" on page 581
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:ONOFF:ON](#)" on page 597
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VMAXimum:ONOFF:ON](#)" on page 604
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:VSteady:TRANsient

N (see page 1232)

Command Syntax :POWer:SIGNals:VSteady:TRANsient <value>[suffix]
 <value> ::= Expected steady state output Voltage value in NR3 format
 [suffix] ::= {V | mV}

The :POWer:SIGNals:VSteady:TRANsient command specifies the expected steady state output DC voltage of the power supply for transient response analysis.

This value is used along with the overshoot percentage to specify the settling band for the transient response and to adjust the vertical scale of the oscilloscope.

Query Syntax :POWer:SIGNals:VSteady:TRANsient?

The :POWer:SIGNals:VSteady:TRANsient query returns the expected steady state voltage setting.

Return Format <value><NL>
 <value> ::= Expected steady state output Voltage value in NR3 format

- See Also**
- "[:POWer:TRANsient:APPLy](#)" on page 618
 - "[:POWer:SIGNals:AUTosetup](#)" on page 591
 - "[:POWer:SIGNals:DURation:TRANsient](#)" on page 599
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608
 - "[:POWer:SIGNals:SOURce:VOLTage<i>](#)" on page 609

:POWer:SIGNals:SOURce:CURRent<i>

N (see page 1232)

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 591

• [":POWer:SIGNals:CYCLeS:HARMonics"](#) on page 592

• [":POWer:SIGNals:CYCLeS:QUALity"](#) on page 593

• [":POWer:SIGNals:DURation:EFFiciency"](#) on page 594

• [":POWer:SIGNals:DURation:MODulation"](#) on page 595

• [":POWer:SIGNals:DURation:ONOOff:OFF"](#) on page 596

• [":POWer:SIGNals:DURation:ONOOff:ON"](#) on page 597

• [":POWer:SIGNals:DURation:RIPple"](#) on page 598

• [":POWer:SIGNals:DURation:TRANsient"](#) on page 599

• [":POWer:SIGNals:IEXPected"](#) on page 600

• [":POWer:SIGNals:OVERshoot"](#) on page 601

• [":POWer:SIGNals:VMAXimum:INRush"](#) on page 602

• [":POWer:SIGNals:VMAXimum:ONOOff:OFF"](#) on page 603

• [":POWer:SIGNals:VMAXimum:ONOOff:ON"](#) on page 604

• [":POWer:SIGNals:VSTeady:ONOOff:OFF"](#) on page 605

• [":POWer:SIGNals:VSTeady:ONOOff:ON"](#) on page 606

• [":POWer:SIGNals:VSTeady:TRANsient"](#) on page 607

• [":POWer:SIGNals:SOURce:VOLTage<i>"](#) on page 609

:POWer:SIGNals:SOURce:VOLTage<i>

N (see page 1232)

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 591
 - "[:POWer:SIGNals:CYCLeS:HARMonics](#)" on page 592
 - "[:POWer:SIGNals:CYCLeS:QUALity](#)" on page 593
 - "[:POWer:SIGNals:DURation:EFFiciency](#)" on page 594
 - "[:POWer:SIGNals:DURation:MODulation](#)" on page 595
 - "[:POWer:SIGNals:DURation:ONOOff:OFF](#)" on page 596
 - "[:POWer:SIGNals:DURation:ONOOff:ON](#)" on page 597
 - "[:POWer:SIGNals:DURation:RIPple](#)" on page 598
 - "[:POWer:SIGNals:DURation:TRANsient](#)" on page 599
 - "[:POWer:SIGNals:IEXPected](#)" on page 600
 - "[:POWer:SIGNals:OVERshoot](#)" on page 601
 - "[:POWer:SIGNals:VMAXimum:INRush](#)" on page 602
 - "[:POWer:SIGNals:VMAXimum:ONOOff:OFF](#)" on page 603
 - "[:POWer:SIGNals:VMAXimum:ONOOff:ON](#)" on page 604
 - "[:POWer:SIGNals:VSTeady:ONOOff:OFF](#)" on page 605
 - "[:POWer:SIGNals:VSTeady:ONOOff:ON](#)" on page 606
 - "[:POWer:SIGNals:VSTeady:TRANsient](#)" on page 607
 - "[:POWer:SIGNals:SOURce:CURREnt<i>](#)" on page 608

:POWer:SLEW:APPLy

N (see page 1232)

Command Syntax :POWer:SLEW:APPLy

The :POWer:SLEW:APPLy command applies the slew rate analysis.

See Also • "[:POWer:SLEW:SOURce](#)" on page 611

:POWer:SLEW:SOURce

N (see page 1232)

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 610

:POWer:SWITch:APPLy**N** (see page 1232)**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 613
- "[:POWer:SWITch:IREFERENCE](#)" on page 614
- "[:POWer:SWITch:RDS](#)" on page 615
- "[:POWer:SWITch:VCE](#)" on page 616
- "[:POWer:SWITch:VREFERENCE](#)" on page 617
- "[:MEASure:ELOSS](#)" on page 504
- "[:MEASure:PLOSS](#)" on page 511

:POWer:SWITch:CONDuction

N (see page 1232)

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: $P = V \times I$
- 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: $P = Id2 \times Rds(on)$
 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: $P = Vce(sat) \times Ic$
 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 612
- "[:POWer:SWITch:IREFerence](#)" on page 614
- "[:POWer:SWITch:RDS](#)" on page 615
- "[:POWer:SWITch:VCE](#)" on page 616
- "[:POWer:SWITch:VREFerence](#)" on page 617

:POWer:SWITch:IREFerence

N (see page 1232)

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 612
 - "[:POWer:SWITch:CONDuCTION](#)" on page 613
 - "[:POWer:SWITch:RDS](#)" on page 615
 - "[:POWer:SWITch:VCE](#)" on page 616
 - "[:POWer:SWITch:VREFerence](#)" on page 617

:POWer:SWITch:RDS

N (see page 1232)

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 612
 - "[:POWer:SWITch:CONDuction](#)" on page 613
 - "[:POWer:SWITch:IREFerence](#)" on page 614
 - "[:POWer:SWITch:VCE](#)" on page 616
 - "[:POWer:SWITch:VREFerence](#)" on page 617

:POWer:SWITch:VCE

N (see page 1232)

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 612
 - "[:POWer:SWITch:CONDuction](#)" on page 613
 - "[:POWer:SWITch:IREference](#)" on page 614
 - "[:POWer:SWITch:RDS](#)" on page 615
 - "[:POWer:SWITch:VREference](#)" on page 617

:POWer:SWITch:VREFerence

N (see page 1232)

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 612
- "[:POWer:SWITch:CONDuction](#)" on page 613
- "[:POWer:SWITch:IREference](#)" on page 614
- "[:POWer:SWITch:RDS](#)" on page 615
- "[:POWer:SWITch:VCE](#)" on page 616

:POWer:TRANsient:APPLy

N (see page 1232)

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 619
- "[:POWer:TRANsient:IINitial](#)" on page 620
- "[:POWer:TRANsient:INEW](#)" on page 621
- "[:POWer:TRANsient:NEXT](#)" on page 622
- "[:MEASure:TRESPonse](#)" on page 515

:POWer:TRANsient:EXIT

N (see page 1232)

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 618
 - "[:POWer:TRANsient:IINitial](#)" on page 620
 - "[:POWer:TRANsient:INEW](#)" on page 621
 - "[:POWer:TRANsient:NEXT](#)" on page 622

:POWer:TRANsient:IINitial

N (see page 1232)

Command Syntax `:POWer:TRANsient:IINitial <value>[suffix]`
`<value> ::= Initial current value in NR3 format`
`[suffix] ::= {A | mA}`

The :POWer:TRANsient:IINitial command to specify the initial load current value. The initial load current will be used as a reference and to trigger the oscilloscope.

Query Syntax `:POWer:TRANsient:IINitial?`

The :POWer:TRANsient:IINitial query returns the initial load current value.

Return Format `<value><NL>`
`<value> ::= Initial current value in NR3 format`

See Also • "[:POWer:SIGNals:VSTeady:TRANsient](#)" on page 607
• "[:POWer:TRANsient:APPLy](#)" on page 618
• "[:POWer:TRANsient:EXIT](#)" on page 619
• "[:POWer:TRANsient:INEW](#)" on page 621
• "[:POWer:TRANsient:NEXT](#)" on page 622

:POWer:TRANsient:INEW

N (see page 1232)

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 618
 - "[:POWer:TRANsient:EXIT](#)" on page 619
 - "[:POWer:TRANsient:IINitial](#)" on page 620
 - "[:POWer:TRANsient:NEXT](#)" on page 622

:POWer:TRANsient:NEXT

N (see page 1232)

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 618
- "[:POWer:TRANsient:EXIT](#)" on page 619
- "[:POWer:TRANsient:IINitial](#)" on page 620
- "[:POWer:TRANsient:INEW](#)" on page 621

26 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

Table 101 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:ARbitrary[:START] [<file_spec> [, <column>] [, <wavegen_id>] (see page 625)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <column> ::= Column in CSV file to load. Column number starts from 1. <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= {WGEN1 WGEN2}
:RECall:DBC[:START] [<file_name>] [, <serialbus>] (see page 626)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".dbc". <serialbus> ::= {SBUS<n>} <n> ::= 1 to (# of serial bus) in NR1 format
:RECall:FILEname <base_name> (see page 627)	:RECall:FILEname? (see page 627)	<base_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see page 628)	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 629)	:RECall:PWD? (see page 629)	<path_name> ::= quoted ASCII string



Table 101 :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:SETup [:STARt] [<file_spec>] (see page 630)	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 631)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

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 1232)

Command Syntax

```
:RECall:ARBitrary[:STARt] [<file_spec> [, <column>] [, <wavegen_id>]
<file_spec> ::= {<internal_loc> | <file_name>}
<column> ::= Column in CSV file to load. Column number starts from 1.
<wavegen_id> ::= {WGEN1 | WGEN2} - specifies which wavegen
<internal_loc> ::= 0-3; an integer in NR1 format
<file_name> ::= quoted ASCII string
```

The :RECall:ARBitrary[:STARt] command recalls an arbitrary waveform.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".csv".

For internal locations, the <column> parameter is ignored.

For external (USB storage device) files, the column parameter is optional. If no <column> parameter is entered, and it is a 2-column file, the 2nd column (assumed to be voltage) is automatically selected. If the <column> parameter is entered, and that column does not exist in the file, the operation fails.

When recalling arbitrary waveforms (from an external USB storage device) that were not saved from the oscilloscope, be aware that the oscilloscope uses a maximum of 8192 points for an arbitrary waveform. For more efficient recalls, make sure your arbitrary waveforms are 8192 points or less.

The <wavegen_id> parameter specifies which waveform generator to recall the arbitrary waveform into.

See Also

- "[Introduction to :RECall Commands](#)" on page 624
- "[":RECall:FILENAME"](#) on page 627
- "[":SAVE:ARBitrary\[:STARt\]"](#) on page 636

:RECall:DBC[:STARt]

N (see page 1232)

Command Syntax `:RECall:DBC[:STARt] [<file_name> [, <serialbus>]]`

`<file_name>` ::= quoted ASCII string

`<serialbus>` ::= {SBUS<n>}

`<n>` ::= 1 to (# of serial bus) in NR1 format

The :RECall:DBC[:STARt] command loads a CAN DBC (communication database) symbolic data file into the oscilloscope.

NOTE

If a file extension is provided as part of a specified `<file_name>`, it must be ".dbc".

The `<serialbus>` parameter specifies which serial decode waveform the CAN symbolic data will be loaded for.

See Also

- "[Introduction to :RECall Commands](#)" on page 624
- "[:RECall:FILEname](#)" on page 627
- "[:RECall:DBC\[:STARt\]](#)" on page 626
- "[:SBUS<n>:CAN:TRIGger](#)" on page 692
- "[:SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge](#)" on page 699
- "[:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNAl](#)" on page 700
- "[:SBUS<n>:CAN:TRIGger:SYMBolic:VALue](#)" on page 701
- "[:SEARch:SERial:CAN:MODE](#)" on page 868
- "[:SEARch:SERial:CAN:SYMBolic:MESSAge](#)" on page 874
- "[:SEARch:SERial:CAN:SYMBolic:SIGNAl](#)" on page 875
- "[:SEARch:SERial:CAN:SYMBolic:VALue](#)" on page 876

:RECall:FILEname

N (see page 1232)

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 624
• "[":RECall:SETup\[:STARt\]](#)" on page 630
• "[":SAVE:FILEname](#)" on page 638

:RECall:MASK[:STARt]

N (see page 1232)

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 624
- ":RECall:FILEname" on page 627
- ":SAVE:MASK[:STARt]" on page 645
- ":MTESt:DATA" on page 532

:RECall:PWD

N (see [page 1232](#))

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 624
• "[":SAVE:PWD"](#) on page 648

:RECall:SETup[:STARt]

N (see page 1232)

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 624
- ":RECall:FILEname" on page 627
- ":SAVE:SETup[:STARt]" on page 649

:RECall:WMEMory<r>[:STARt]

N (see page 1232)

Command Syntax :RECall:WMEMory<r>[:STARt] [<file_name>]
<r> ::= 1 to (# ref waveforms) 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 624
 - "[:RECall:FILEname](#)" on page 627
 - "[:SAVE:WMEMory\[:STARt\]](#)" on page 656

27 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "[Introduction to :SAVE Commands](#)" on page 635.

Table 102 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:ARBitrary[:STARt] [<file_spec>] [, <wavegen_id>] (see page 636)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string <wavegen_id> ::= {WGEN1 WGEN2}
:SAVE:COMpliance:USB[:START] [<file_name>] (see page 637)	n/a	<file_name> ::= quoted ASCII string
:SAVE:FILEname <base_name> (see page 638)	:SAVE:FILEname? (see page 638)	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_name>] (see page 639)	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGE:FACTors {{0 OFF} {1 ON}} (see page 640)	:SAVE:IMAGE:FACTors? (see page 640)	{0 1}
:SAVE:IMAGE:FORMAT <format> (see page 641)	:SAVE:IMAGE:FORMAT? (see page 641)	<format> ::= {{BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGE:INKSaver {{0 OFF} {1 ON}} (see page 642)	:SAVE:IMAGE:INKSaver? (see page 642)	{0 1}
:SAVE:IMAGE:PALETTE <palette> (see page 643)	:SAVE:IMAGE:PALETTE? (see page 643)	<palette> ::= {COLOR GRAYSCALE}



27 :SAVE Commands

Table 102 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:LISTER[:START] [<file_name>] (see page 644)	n/a	<file_name> ::= quoted ASCII string
:SAVE:MASK[:START] [<file_spec>] (see page 645)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:MULTi[:START] [<file_name>] (see page 646)	n/a	<file_name> ::= quoted ASCII string
:SAVE:POWer[:START] [<file_name>] (see page 647)	n/a	<file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 648)	:SAVE:PWD? (see page 648)	<path_name> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see page 649)	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 650)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMAT <format> (see page 651)	:SAVE:WAVeform:FORMAT? (see page 651)	<format> ::= {ASCIixy CSV BINary NONE}
:SAVE:WAVeform:LENGTH <length> (see page 652)	:SAVE:WAVeform:LENGTH? (see page 652)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:LENGTH :MAX {{0 OFF} {1 ON}} (see page 653)	:SAVE:WAVeform:LENGTH :MAX? (see page 653)	{0 1}
:SAVE:WAVeform:SEGmented <option> (see page 654)	:SAVE:WAVeform:SEGmented? (see page 654)	<option> ::= {ALL CURRent}

Table 102 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WMEMORY:SOURce <source> (see page 655)	:SAVE:WMEMORY:SOURce? (see page 655)	<p><source> ::= {CHANnel<n> FUNCTION<m> MATH<m> WMEMORY<r>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p>NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.</p> <p><return_value> ::= <source></p>
:SAVE:WMEMORY[:START] [<file_name>] (see page 656)	n/a	<p><file_name> ::= quoted ASCII string</p> <p>If extension included in file name, it must be ".h5".</p>

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 1232)

Command Syntax

```
:SAVE:ARBitrary[:STARt] [<file_spec> [, <wavegen_id>]
<file_spec> ::= {<internal_loc> | <file_name>}
<internal_loc> ::= 0-3; an integer in NR1 format
<file_name> ::= quoted ASCII string
<wavegen_id> ::= {WGEN1 | WGEN2}
```

The :SAVE:ARBitrary[:STARt] command saves the current arbitrary waveform to an internal location or a file on a USB storage device.

The <wavegen_id> parameter specifies which waveform generator to save the arbitrary waveform from.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".csv".

See Also

- "[Introduction to :SAVE Commands](#)" on page 635
- "[":SAVE:FILENAME](#)" on page 638
- "[":RECALL:ARBITRARY\[:STARt\]](#)" on page 625

:SAVE:COMPliance:USB[:STARt]

N (see page 1232)

Command Syntax :SAVE:COMPliance:USB[:STARt] [<file_name>]
<file_name> ::= quoted ASCII string

The :SAVE:COMPliance:USB[:STARt] command saves USB 2.0 signal quality test results to a file.

NOTE

If a file extension is provided as part of a specified <file_name>, it must be ".html".

-
- See Also**
- "Introduction to :SAVE Commands" on page 635
 - ":SAVE:FILENAME" on page 638
 - ":SAVE:PWD" on page 648
 - Chapter 11, "COMPliance Commands," starting on page 287

:SAVE:FILEname

N (see page 1232)

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 635
 - "[":SAVE:IMAGe\[:STARt\]](#)" on page 639
 - "[":SAVE:SETup\[:STARt\]](#)" on page 649
 - "[":SAVE:WAveform\[:STARt\]](#)" on page 650
 - "[":SAVE:PWD](#)" on page 648
 - "[":RECall:FILEname](#)" on page 627

:SAVE:IMAGe[:STARt]

N (see page 1232)

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 635
- "[:SAVE:IMAGe:FACTors](#)" on page 640
- "[:SAVE:IMAGe:FORMat](#)" on page 641
- "[:SAVE:IMAGe:INKSaver](#)" on page 642
- "[:SAVE:IMAGe:PAlette](#)" on page 643
- "[:SAVE:FILEname](#)" on page 638

:SAVE:IMAGe:FACTors

N (see page 1232)

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 635

- "[":SAVE:IMAGe\[:STARt\]](#)" on page 639
- "[":SAVE:IMAGe:FORMat](#)" on page 641
- "[":SAVE:IMAGe:INKSaver](#)" on page 642
- "[":SAVE:IMAGe:PALETTE](#)" on page 643

:SAVE:IMAGe:FORMat

N (see page 1232)

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 635
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 639
- "[":SAVE:IMAGe:FACTors](#)" on page 640
- "[":SAVE:IMAGe:INKSaver](#)" on page 642
- "[":SAVE:IMAGe:PAlette](#)" on page 643
- "[":SAVE:WAveform:FORMat](#)" on page 651

:SAVE:IMAGe:INKSaver

N (see page 1232)

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 635
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 639
- "[":SAVE:IMAGe:FACTors](#)" on page 640
- "[":SAVE:IMAGe:FORMAT](#)" on page 641
- "[":SAVE:IMAGe:PAlette](#)" on page 643

:SAVE:IMAGe:PAlette

N (see page 1232)

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 635
• ":SAVE:IMAGe[:STARt]" on page 639
• ":SAVE:IMAGe:FACTors" on page 640
• ":SAVE:IMAGe:FORMAT" on page 641
• ":SAVE:IMAGe:INKSaver" on page 642

:SAVE:LISTER[:STARt]

N (see [page 1232](#))

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 635
 - "[":SAVE:FILENAME](#)" on page 638
 - [Chapter 19, “:LISTER Commands,” starting on page 393](#)

:SAVE:MASK[:STARt]

N (see page 1232)

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 635
- "[:SAVE:FILEname](#)" on page 638
- "[:RECall:MASK\[:STARt\]](#)" on page 628
- "[:MTESt:DATA](#)" on page 532

:SAVE:MULTi[:STARt]

N (see [page 1232](#))

Command Syntax `:SAVE:MULTi[:STARt] [<file_name>]`
`<file_name>` ::= quoted ASCII string

The `:SAVE:MULTi[:STARt]` command saves multi-channel waveform data to a file. This file can be opened by the N8900A InfiniiView oscilloscope analysis software.

NOTE

If a file extension is provided as part of a specified `<file_name>`, it must be ".h5".

-
- See Also**
- "[Introduction to :SAVE Commands](#)" on page 635
 - "[":SAVE:FILEname](#)" on page 638
 - "[":SAVE:PWD](#)" on page 648

:SAVE:POWer[:STARt]

N (see page 1232)

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 635
 - ":SAVE:FILENAME" on page 638
 - Chapter 25, “:POWer Commands,” starting on page 557

:SAVE:PWD

N (see [page 1232](#))

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 635
- "[":SAVE:FILENAME](#)" on page 638
- "[":RECALL:PWD](#)" on page 629

:SAVE:SETup[:STARt]

N (see page 1232)

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 635
- "[":SAVE:FILEname](#)" on page 638
- "[":RECall:SETup\[:STARt\]](#)" on page 630

:SAVE:WAVeform[:STARt]**N** (see page 1232)**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 635
- "[":SAVE:WAVeform:FORMAT](#)" on page 651
- "[":SAVE:WAVeform:LENGTH](#)" on page 652
- "[":SAVE:FILEname](#)" on page 638
- "[":RECall:SETup\[:STARt\]](#)" on page 630

:SAVE:WAVeform:FORMat

N (see [page 1232](#))

Command Syntax :SAVE:WAVeform:FORMat <format>

<format> ::= {ASCIixy | CSV | BINARY}

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ASCIixy – creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV – creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINary – creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

Query Syntax :SAVE:WAVeform:FORMat?

The :SAVE:WAVeform:FORMat? query returns the selected waveform data format type.

Return Format <format><NL>

<format> ::= {ASC | CSV | BIN | NONE}

When NONE is returned, it indicates that an image file format is currently selected.

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 635
 - "[":SAVE:WAVeform\[:START\]](#)" on page 650
 - "[":SAVE:WAVeform:LENGTH](#)" on page 652
 - "[":SAVE:IMAGe:FORMAT](#)" on page 641

:SAVE:WAVeform:LENGth

N (see page 1232)

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 635

• "[:SAVE:WAVeform:LENGth:MAX](#)" on page 653

• "[:SAVE:WAVeform\[:START\]](#)" on page 650

• "[:WAVeform:POINts](#)" on page 1056

• "[:SAVE:WAVeform:FORMAT](#)" on page 651

:SAVE:WAVeform:LENGth:MAX

N (see page 1232)

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 635
- ":SAVE:WAVeform[:STARt]" on page 650
- ":SAVE:WAVeform:LENGth" on page 652

:SAVE:WAveform:SEGmented

N (see [page 1232](#))

Command Syntax `:SAVE:WAveform:SEGmented <option>`
`<option> ::= {ALL | CURR}`

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.
- CURR – 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 635
- "[":SAVE:WAveform\[:START\]" on page 650](#)
- "[":SAVE:WAveform:FORMAT" on page 651](#)
- "[":SAVE:WAveform:LENGTH" on page 652](#)

:SAVE:WMEMORY:SOURce

N (see page 1232)

Command Syntax

```
:SAVE:WMEMORY:SOURce <source>
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m> | WMEMory<r>}
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :SAVE:WMEMORY:SOURce command selects the source to be saved as a reference waveform file.

NOTE

Only ADD or SUBtract math operations can be saved as reference waveforms.

NOTE

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

Query Syntax

```
:SAVE:WMEMORY:SOURce?
```

The :SAVE:WMEMORY:SOURce? query returns the source to be saved as a reference waveform file.

Return Format

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

See Also

- "[Introduction to :SAVE Commands](#)" on page 635
- "[":SAVE:WMEMORY\[:START\]" on page 656](#)
- "[":RECall:WMEMORY<r>\[:START\]" on page 631](#)

:SAVE:WMEMORY[:STARt]

N (see [page 1232](#))

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 635
 - "[":SAVE:WMEMORY:SOURce](#)" on page 655
 - "[":RECall:WMEMORY<r>\[:STARt\]" on page 631](#)

28

:SBUS<n> Commands

Control the modes and parameters for each serial bus decode/trigger type.
See:

- "[Introduction to :SBUS<n> Commands](#)" on page 657
- "[General :SBUS<n> Commands](#)" on page 659
- "[:SBUS<n>:A429 Commands](#)" on page 662
- "[:SBUS<n>:CAN Commands](#)" on page 680
- "[:SBUS<n>:FLEXray Commands](#)" on page 702
- "[:SBUS<n>:I2S Commands](#)" on page 721
- "[:SBUS<n>:IIC Commands](#)" on page 740
- "[:SBUS<n>:LIN Commands](#)" on page 750
- "[:SBUS<n>:M1553 Commands](#)" on page 764
- "[:SBUS<n>:SPI Commands](#)" on page 771
- "[:SBUS<n>:UART Commands](#)" on page 787
- "[:SBUS<n>:USB Commands](#)" on page 809

**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 968).

- **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.
- **USB 2.0 triggering** (with Option USF or Option U2H) — lets you trigger on USB 2.0 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 103 General :SBUS<n> Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 660)	:SBUS<n>:DISPlay? (see page 660)	{0 1}
:SBUS<n>:MODE <mode> (see page 661)	:SBUS<n>:MODE? (see page 661)	<mode> ::= {A429 CAN FLEXray I2S IIC LIN M1553 SPI UART USB}

:SBUS<n>:DISPlay

N (see page 1232)

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}

- "-241, Hardware missing" on page 1193

See Also

- "Introduction to :SBUS<n> Commands" on page 657
- ":CHANnel<n>:DISPlay" on page 272
- ":DIGItal<d>:DISPlay" on page 309
- ":POD<n>:DISPlay" on page 553
- ":VIEW" on page 230
- ":BLANK" on page 202
- ":STATus" on page 227

:SBUS<n>:MODE

N (see page 1232)

Command Syntax :SBUS<n>:MODE <mode>

```
<mode> ::= {A429 | FLEXray | CAN | I2S | IIC | LIN | M1553 | SPI
             | UART | USB}
```

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
             | USB | NONE}
```

Errors • "-241, Hardware missing" on page 1193

See Also

- "Introduction to :SBUS<n> Commands" on page 657
- ":SBUS<n>:A429 Commands" on page 662
- ":SBUS<n>:CAN Commands" on page 680
- ":SBUS<n>:FLEXray Commands" on page 702
- ":SBUS<n>:I2S Commands" on page 721
- ":SBUS<n>:IIC Commands" on page 740
- ":SBUS<n>:LIN Commands" on page 750
- ":SBUS<n>:M1553 Commands" on page 764
- ":SBUS<n>:SPI Commands" on page 771
- ":SBUS<n>:UART Commands" on page 787
- ":SBUS<n>:USB Commands" on page 809

:SBUS<n>:A429 Commands

NOTE

These commands are valid when the DSOX4AERO MIL-STD-1553 and ARINC 429 triggering and serial decode option (Option AERO) has been licensed.

Table 104 :SBUS<n>:A429 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:A429:AUTOset up (see page 664)	n/a	n/a
:SBUS<n>:A429:BASE <base> (see page 665)	:SBUS<n>:A429:BASE? (see page 665)	<base> ::= {BINary HEX}
n/a	:SBUS<n>:A429:COUNT:E RRor? (see page 666)	<error_count> ::= integer in NR1 format
:SBUS<n>:A429:COUNT:R ESet (see page 667)	n/a	n/a
n/a	:SBUS<n>:A429:COUNT:W ORD? (see page 668)	<word_count> ::= integer in NR1 format
:SBUS<n>:A429:FORMAT <format> (see page 669)	:SBUS<n>:A429:FORMAT? (see page 669)	<format> ::= {LDSDi LDSSm LDAData}
:SBUS<n>:A429:SIGNAl <signal> (see page 670)	:SBUS<n>:A429:SIGNAl? (see page 670)	<signal> ::= {A B DIFFerential}
:SBUS<n>:A429:SOURce <source> (see page 671)	:SBUS<n>:A429:SOURce? (see page 671)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:A429:SPEEd <speed> (see page 672)	:SBUS<n>:A429:SPEEd? (see page 672)	<speed> ::= {LOW HIGH}
:SBUS<n>:A429:TRIGger :LABel <value> (see page 673)	:SBUS<n>:A429:TRIGger :LABel? (see page 673)	<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 104 :SBUS<n>:A429 Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:A429:TRIGger :PATTern:DATA <string> (see page 674)	:SBUS<n>:A429:TRIGger :PATTern:DATA? (see page 674)	<string> ::= "nn...n" where n ::= {0 1 X}, length depends on FORMat
:SBUS<n>:A429:TRIGger :PATTern:SDI <string> (see page 675)	:SBUS<n>:A429:TRIGger :PATTern:SDI? (see page 675)	<string> ::= "nn" where n ::= {0 1 X}, length always 2 bits
:SBUS<n>:A429:TRIGger :PATTern:SSM <string> (see page 676)	:SBUS<n>:A429:TRIGger :PATTern:SSM? (see page 676)	<string> ::= "nn" where n ::= {0 1 X}, length always 2 bits
:SBUS<n>:A429:TRIGger :RANGE <min>,<max> (see page 677)	:SBUS<n>:A429:TRIGger :RANGE? (see page 677)	<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 678)	:SBUS<n>:A429:TRIGger :TYPE? (see page 678)	<condition> ::= {WSTArt WSTOP LABel LBITS PERRor WERRor GERRor WGERRors ALLerrors LRANGE ABITS AOBits AZBits}

:SBUS<n>:A429:AUTosetup

(see page 1232)

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 1193

- See Also** • ":SBUS<n>:A429:BASE" on page 665
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SBUS<n>:A429:TRIGger:TYPE" on page 678
 • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:A429 Commands" on page 662

:SBUS<n>:A429:BASE

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:A429:FORMAT" on page 669

:SBUS<n>:A429:COUNt:ERRor

N (see page 1232)

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 1193

See Also • ":SBUS<n>:A429:COUNt:RESet" on page 667
• ":SBUS<n>:A429:COUNt:WORD" on page 668
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:A429 Commands" on page 662

:SBUS<n>:A429:COUNt:RESet

N (see page 1232)

Command Syntax :SBUS<n>:A429:COUNt:RESet

Resets the word and error counters.

Errors • "-241, Hardware missing" on page 1193

See Also • ":SBUS<n>:A429:COUNt:WORD" on page 668
• ":SBUS<n>:A429:COUNt:ERRor" on page 666
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:A429 Commands" on page 662

:SBUS<n>:A429:COUNt:WORD**N**

(see page 1232)

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 1193**See Also** • ":SBUS<n>:A429:COUNt:RESet" on page 667
• ":SBUS<n>:A429:COUNt:ERRor" on page 666
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:A429 Commands" on page 662

:SBUS<n>:A429:FORMAT

N (see page 1232)

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 | LDATA}

Errors • "-241, Hardware missing" on page 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:A429:TRIGger:PATTern:DATA" on page 674
 • ":SBUS<n>:A429:TRIGger:PATTern:SDI" on page 675
 • ":SBUS<n>:A429:TRIGger:PATTern:SSM" on page 676
 • ":SBUS<n>:A429:TRIGger:TYPE" on page 678
 • ":SBUS<n>:A429:SIGNal" on page 670
 • ":SBUS<n>:A429:SPEed" on page 672
 • ":SBUS<n>:A429:BASE" on page 665
 • ":SBUS<n>:A429:SOURce" on page 671

:SBUS<n>:A429:SIGNAl**N** (see page 1232)**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 1193**See Also** • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SBUS<n>:A429:SPEed" on page 672
 • ":SBUS<n>:A429:SOURce" on page 671

:SBUS<n>:A429:SOURce

N (see page 1232)

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 966
- "[:TRIGger:LEVel:LOW](#)" on page 967
- "[:TRIGger:MODE](#)" on page 968
- "[:SBUS<n>:MODE](#)" on page 661
- "[:SBUS<n>:A429:TRIGger:TYPE](#)" on page 678
- "[:SBUS<n>:A429:SIGNal](#)" on page 670
- "[:SBUS<n>:A429:SPEed](#)" on page 672
- "[:SBUS<n>:A429:FORMAT](#)" on page 669
- "[Introduction to :TRIGger Commands](#)" on page 959

:SBUS<n>:A429:SPEed

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:A429:SIGNal" on page 670
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SBUS<n>:A429:SOURce" on page 671

:SBUS<n>:A429:TRIGger:LABel

N (see page 1232)

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 1193

See Also

- "Introduction to :TRIGger Commands" on page 959
- ":SBUS<n>:A429:TRIGger:TYPE" on page 678

:SBUS<n>:A429:TRIGger:PATTERn:DATA

N (see page 1232)

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 1193

See Also

- "Introduction to :TRIGger Commands" on page 959
- ":SBUS<n>:A429:TRIGger:TYPE" on page 678
- ":SBUS<n>:A429:TRIGger:PATTERn:SDI" on page 675
- ":SBUS<n>:A429:TRIGger:PATTERn:SSM" on page 676

:SBUS<n>:A429:TRIGger:PATTERn:SDI

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SBUS<n>:A429:TRIGger:TYPE" on page 678
 • ":SBUS<n>:A429:TRIGger:PATTERn:DATA" on page 674
 • ":SBUS<n>:A429:TRIGger:PATTERn:SSM" on page 676

:SBUS<n>:A429:TRIGger:PATTERn:SSM

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SBUS<n>:A429:TRIGger:TYPE" on page 678
 • ":SBUS<n>:A429:TRIGger:PATTERn:DATA" on page 674
 • ":SBUS<n>:A429:TRIGger:PATTERn:SDI" on page 675

:SBUS<n>:A429:TRIGger:RANGE

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
• ":SBUS<n>:A429:TRIGger:TYPE" on page 678

:SBUS<n>:A429:TRIGger:TYPE

N (see page 1232)

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.
- AOBits – triggers on any bit with a value of zero.
- AZBits – 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 1193

See Also • ["Introduction to :SBUS<n> Commands"](#) on page 657

- [":SBUS<n>:MODE"](#) on page 661
- [":SBUS<n>:A429:TRIGger:LABel"](#) on page 673
- [":SBUS<n>:A429:TRIGger:PATTern:DATA"](#) on page 674
- [":SBUS<n>:A429:TRIGger:PATTern:SDI"](#) on page 675
- [":SBUS<n>:A429:TRIGger:PATTern:SSM"](#) on page 676
- [":SBUS<n>:A429:TRIGger:RANGE"](#) on page 677

- "[:SBUS<n>:A429:SOURce](#)" on page 671

:SBUS<n>:CAN Commands

NOTE

These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 105 :SBUS<n>:CAN Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS<n>:CAN:COUNT:ER Ror? (see page 682)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:OV ERload? (see page 683)	<frame_count> ::= integer in NR1 format
:SBUS<n>:CAN:COUNT:RE Set (see page 684)	n/a	n/a
n/a	:SBUS<n>:CAN:COUNT:TO Tal? (see page 685)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:CAN:COUNT:UT ILization? (see page 686)	<percent> ::= floating-point in NR3 format
:SBUS<n>:CAN:DISPlay <type> (see page 687)	:SBUS<n>:CAN:DISPlay? (see page 687)	<type> ::= {HEXadecimal SYMBolic}
:SBUS<n>:CAN:SAMPLEpo int <value> (see page 688)	:SBUS<n>:CAN:SAMPLEpo int? (see page 688)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:CAN:SIGNAL:B AUDrate <baudrate> (see page 689)	:SBUS<n>:CAN:SIGNAL:B AUDrate? (see page 689)	<baudrate> ::= integer from 10000 to 4000000 in 100 b/s increments, or 5000000
:SBUS<n>:CAN:SIGNAL:D EFinition <value> (see page 690)	:SBUS<n>:CAN:SIGNAL:D EFinition? (see page 690)	<value> ::= {CANH CANL RX TX DIFFerential DIFL DIFH}
:SBUS<n>:CAN:SOURCE <source> (see page 691)	:SBUS<n>:CAN:SOURce? (see page 691)	<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 105 :SBUS<n>:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:CAN:TRIGger <condition> (see page 692)	:SBUS<n>:CAN:TRIGger? (see page 693)	<condition> ::= {SOF DATA ERRor IDData IDEither IDRmote ALLerrors OVERload ACKerror MESSage MSIGnal}
:SBUS<n>:CAN:TRIGger: PATtern:DATA <string> (see page 695)	:SBUS<n>:CAN:TRIGger: PATtern:DATA? (see page 695)	<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 696)	:SBUS<n>:CAN:TRIGger: PATtern:DATA:LENGTH? (see page 696)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:CAN:TRIGger: PATtern:ID <string> (see page 697)	:SBUS<n>:CAN:TRIGger: PATtern:ID? (see page 697)	<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 698)	:SBUS<n>:CAN:TRIGger: PATtern:ID:MODE? (see page 698)	<value> ::= {STANDARD EXTended}
:SBUS<n>:CAN:TRIGger: SYMBOLic:MESSAge <name> (see page 699)	:SBUS<n>:CAN:TRIGger: SYMBOLic:MESSAge? (see page 699)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger: SYMBOLic:SIGNAl <name> (see page 700)	:SBUS<n>:CAN:TRIGger: SYMBOLic:SIGNAl? (see page 700)	<name> ::= quoted ASCII string
:SBUS<n>:CAN:TRIGger: SYMBOLic:VALue <data> (see page 701)	:SBUS<n>:CAN:TRIGger: SYMBOLic:VALue? (see page 701)	<data> ::= value in NR3 format

:SBUS<n>:CAN:COUNt:ERRQ

N (see page 1232)

Query Syntax :SBUS<n>:CAN:COUNt:ERRQ?

Returns the error frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • "-241, Hardware missing" on page 1193

See Also • ":SBUS<n>:CAN:COUNt:RESet" on page 684
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:CAN Commands" on page 680

:SBUS<n>:CAN:COUNt:OVERload

N (see page 1232)

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 1193

See Also • ":SBUS<n>:CAN:COUNt:RESet" on page 684
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:CAN Commands" on page 680

:SBUS<n>:CAN:COUNt:RESet**N**

(see page 1232)

Command Syntax :SBUS<n>:CAN:COUNt:RESet

Resets the frame counters.

Errors • "-241, Hardware missing" on page 1193**See Also** • ":SBUS<n>:CAN:COUNt:ERRor" on page 682

• ":SBUS<n>:CAN:COUNt:OVERload" on page 683

• ":SBUS<n>:CAN:COUNt:TOTal" on page 685

• ":SBUS<n>:CAN:COUNt:UTILization" on page 686

• "Introduction to :SBUS<n> Commands" on page 657

• ":SBUS<n>:MODE" on page 661

• ":SBUS<n>:CAN Commands" on page 680

:SBUS<n>:CAN:COUNt:TOTal

N (see page 1232)

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 1193

See Also • ":SBUS<n>:CAN:COUNt:RESet" on page 684
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:CAN Commands" on page 680

:SBUS<n>:CAN:COUNt:UTILization

N (see [page 1232](#))

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 1193

See Also • "[:SBUS<n>:CAN:COUNt:RESet](#)" on page 684
• "[Introduction to :SBUS<n> Commands](#)" on page 657
• "[:SBUS<n>:MODE](#)" on page 661
• "[:SBUS<n>:CAN Commands](#)" on page 680

:SBUS<n>:CAN:DISPlay

N (see page 1232)

Command Syntax :SBUS<n>:CAN:DISPlay <type>
<type> ::= {HEXadecimal | SYMBolic}

The :SBUS<n>:CAN:DISPlay command specifies, when CAN symbolic data is loaded into the oscilloscope, whether symbolic values (from the DBC file) or hexadecimal values are displayed in the decode waveform and the Lister window.

Query Syntax :SBUS<n>:CAN:DISPlay?

The :SBUS<n>:CAN:DISPlay? query returns the CAN decode display type.

Return Format <type><NL>
<type> ::= {HEX | SYMB}

See Also • "[:RECall:DBC\[:STARt\]](#)" on page 626

:SBUS<n>:CAN:SAMPLEpoint**N** (see page 1232)**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 959
• "[:SBUS<n>:MODE](#)" on page 661
• "[:SBUS<n>:CAN:TRIGger](#)" on page 692

:SBUS<n>:CAN:SIGNAl:BAUDrate

N (see page 1232)

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 959
- ":SBUS<n>:MODE" on page 661
- ":SBUS<n>:CAN:TRIGger" on page 692
- ":SBUS<n>:CAN:SIGNAl:DEFinition" on page 690
- ":SBUS<n>:CAN:SOURce" on page 691

:SBUS<n>:CAN:SIGNAl:DEFinition

N (see page 1232)

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 959
- "[:SBUS<n>:MODE](#)" on page 661
- "[:SBUS<n>:CAN:SIGNAl:BAUDrate](#)" on page 689
- "[:SBUS<n>:CAN:SOURce](#)" on page 691
- "[:SBUS<n>:CAN:TRIGger](#)" on page 692

:SBUS<n>:CAN:SOURce

N (see page 1232)

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 959
 - ":SBUS<n>:MODE" on page 661
 - ":SBUS<n>:CAN:TRIGGER" on page 692
 - ":SBUS<n>:CAN:SIGNAl:DEFinition" on page 690

:SBUS<n>:CAN:TRIGger

N (see page 1232)

Command Syntax `:SBUS<n>:CAN:TRIGger <condition>`

```
<condition> ::= {SOF | DATA | ERROR | IDData | IDEither | IDRemote |
    ALLerrors | OVERload | ACKerror | MESSAGE | MSIGNAL}
```

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.
- MESSAGE – when symbolic data is loaded into the oscilloscope (see "[:RECall:DBC\[:STARt\]](#)" on page 626), you can trigger on a symbolic message. To specify the message, use "[":SBUS<n>:CAN:TRIGger:SYMBOLic:MESSAge](#)" on page 699.
- MSIGNAL – when symbolic data is loaded into the oscilloscope, you can trigger on a symbolic message and a signal value. To specify the signal and signal value, use "[":SBUS<n>:CAN:TRIGger:SYMBOLic:SIGNAl](#)" on page 700 and "[":SBUS<n>:CAN:TRIGger:SYMBOLic:VALUe](#)" on page 701.

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

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
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
MESSage	Message
MSIGnal	Message and Signal

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
| MESS | MSIG}
```

- "-241, Hardware missing" on page 1193

See Also

- "Introduction to :SBUS<n> Commands" on page 657
- ":SBUS<n>:MODE" on page 661
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA" on page 695
- ":SBUS<n>:CAN:TRIGger:PATTern:DATA:LENGth" on page 696
- ":SBUS<n>:CAN:TRIGger:PATTern:ID" on page 697
- ":SBUS<n>:CAN:TRIGger:PATTern:ID:MODE" on page 698
- ":SBUS<n>:CAN:SIGNal:DEFinition" on page 690
- ":SBUS<n>:CAN:SOURce" on page 691
- ":RECall:DBC[:STARt]" on page 626
- ":SBUS<n>:CAN:TRIGger:SYMBolic:MESSage" on page 699
- ":SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal" on page 700

- "[:SBUS<n>:CAN:TRIGger:SYMBolic:VALue](#)" on page 701

:SBUS<n>:CAN:TRIGger:PATTERn:DATA

N (see page 1232)

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

- "-241, Hardware missing" on page 1193

See Also

- "Introduction to :TRIGger Commands" on page 959
- ":SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth" on page 696
- ":SBUS<n>:CAN:TRIGger:PATTERn:ID" on page 697

:SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGTH

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:CAN:TRIGger:PATTERn:DATA" on page 695
 • ":SBUS<n>:CAN:SOURce" on page 691

:SBUS<n>:CAN:TRIGger:PATTERn:ID

N (see page 1232)

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 1193

See Also

- "Introduction to :TRIGger Commands" on page 959
- ":SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE" on page 698
- ":SBUS<n>:CAN:TRIGger:PATTERn:DATA" on page 695

:SBUS<n>:CAN:TRIGger:PATTERn:ID:MODE

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:MODE" on page 661
 • ":SBUS<n>:CAN:TRIGger:PATTERn:DATA" on page 695
 • ":SBUS<n>:CAN:TRIGger:PATTERn:DATA:LENGth" on page 696
 • ":SBUS<n>:CAN:TRIGger:PATTERn:ID" on page 697

:SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge

N (see page 1232)

Command Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge <name>
 <name> ::= quoted ASCII string

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge command specifies the message to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MESSage or MSIGnal.

Query Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge?

The :SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge? query returns the specified message.

Return Format <name><NL>
 <name> ::= quoted ASCII string

- See Also**
- "[:RECall:DBC\[:STARt\]](#)" on page 626
 - "[:SBUS<n>:CAN:TRIGger](#)" on page 692
 - "[:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNAl](#)" on page 700
 - "[:SBUS<n>:CAN:TRIGger:SYMBolic:VALue](#)" on page 701

:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal

N (see page 1232)

Command Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal <name>
 <name> ::= quoted ASCII string

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal command specifies the signal to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

Query Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal?

The :SBUS<n>:CAN:TRIGger:SYMBolic:SIGNal? query returns the specified signal.

Return Format <name><NL>
 <name> ::= quoted ASCII string

See Also • "[:RECall:DBC\[:STARt\]](#)" on page 626
 • "[:SBUS<n>:CAN:TRIGger](#)" on page 692
 • "[:SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge](#)" on page 699
 • "[:SBUS<n>:CAN:TRIGger:SYMBolic:VALUe](#)" on page 701

:SBUS<n>:CAN:TRIGger:SYMBolic:VALue

N (see page 1232)

Command Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:VALue <data>
 <data> ::= value in NR3 format

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue command specifies the signal value to trigger on when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN trigger mode is set to MSIGnal.

NOTE

Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax :SBUS<n>:CAN:TRIGger:SYMBolic:VALue?

The :SBUS<n>:CAN:TRIGger:SYMBolic:VALue? query returns the specified signal value.

Return Format <data><NL>

<data> ::= value in NR3 format

- See Also**
- "[:RECall:DBC\[:STARt\]](#)" on page 626
 - "[:SBUS<n>:CAN:TRIGger](#)" on page 692
 - "[:SBUS<n>:CAN:TRIGger:SYMBolic:MESSAge](#)" on page 699
 - "[:SBUS<n>:CAN:TRIGger:SYMBolic:SIGNAl](#)" on page 700

:SBUS<n>:FLEXray Commands

NOTE

These commands are only valid when the FLEXray triggering and serial decode option (Option FLEX) has been licensed.

Table 106 :SBUS<n>:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:AUTOsetup (see page 704)	n/a	n/a
:SBUS<n>:FLEXray:BAUDrate <baudrate> (see page 705)	:SBUS<n>:FLEXray:BAUDrate? (see page 705)	<baudrate> ::= {2500000 5000000 10000000}
:SBUS<n>:FLEXray:CHANNEL <channel> (see page 706)	:SBUS<n>:FLEXray:CHANNEL? (see page 706)	<channel> ::= {A B}
n/a	:SBUS<n>:FLEXray:COUNT:NULL? (see page 707)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:COUNT:RESET (see page 708)	n/a	n/a
n/a	:SBUS<n>:FLEXray:COUNT:SYNC? (see page 709)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:FLEXray:COUNT:TOTAl? (see page 710)	<frame_count> ::= integer in NR1 format
:SBUS<n>:FLEXray:SOURCE <source> (see page 711)	:SBUS<n>:FLEXray:SOURCE? (see page 711)	<source> ::= {CHANNEL<n>} <n> ::= 1-2 or 1-4 in NR1 format
:SBUS<n>:FLEXray:TRIGGER <condition> (see page 712)	:SBUS<n>:FLEXray:TRIGGER? (see page 712)	<condition> ::= {FRAMe ERRor EVENT}
:SBUS<n>:FLEXray:TRIGGER:ERROR:TYPE <error_type> (see page 713)	:SBUS<n>:FLEXray:TRIGGER:ERROR:TYPE? (see page 713)	<error_type> ::= {ALL HCRC FCRC}
:SBUS<n>:FLEXray:TRIGGER:EVENT:AUTOSET (see page 714)	n/a	n/a

Table 106 :SBUS<n>:FLEXray Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:FLEXray:TRIG ger:EVENT:BSS:ID <frame_id> (see page 715)	:SBUS<n>:FLEXray:TRIG ger:EVENT:BSS:ID? (see page 715)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIG ger:EVENT:TYPE <event> (see page 716)	:SBUS<n>:FLEXray:TRIG ger:EVENT:TYPE? (see page 716)	<event> ::= {WAKEup TSS {FES DTS} BSS}
:SBUS<n>:FLEXray:TRIG ger:FRAMe:CCBase <cycle_count_base> (see page 717)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:CCBase? (see page 717)	<cycle_count_base> ::= integer from 0-63
:SBUS<n>:FLEXray:TRIG ger:FRAMe:CCRepetitio n <cycle_count_repetiti on> (see page 718)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:CCRepetitio n? (see page 718)	<cycle_count_repetition> ::= {ALL <rep #>} <rep #> ::= integer values 2, 4, 8, 16, 32, or 64
:SBUS<n>:FLEXray:TRIG ger:FRAMe:ID <frame_id> (see page 719)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:ID? (see page 719)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SBUS<n>:FLEXray:TRIG ger:FRAMe:TYPE <frame_type> (see page 720)	:SBUS<n>:FLEXray:TRIG ger:FRAMe:TYPE? (see page 720)	<frame_type> ::= {NORMAL STARtup NULL SYNC NSTArtup NNULl NSYNC ALL}

:SBUS<n>:FLEXray:AUTosetup

(see page 1232)

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 959
- "[":SBUS<n>:FLEXray:TRIGger](#)" on page 712
- "[":SBUS<n>:FLEXray:BAUDrate](#)" on page 705
- "[":TRIGger\[:EDGE\]:LEVel](#)" on page 986
- "[":SBUS<n>:FLEXray:SOURce](#)" on page 711

:SBUS<n>:FLEXray:BAUDrate

N (see page 1232)

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 959
• ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:CHANnel

N (see page 1232)

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 959
 • ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:COUNt:NULL

N (see page 1232)

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 1193

See Also • ":SBUS<n>:FLEXray:COUNt:RESet" on page 708

• ":SBUS<n>:FLEXray:COUNt:TOTal" on page 710

• ":SBUS<n>:FLEXray:COUNt:SYNC" on page 709

• "Introduction to :SBUS<n> Commands" on page 657

• ":SBUS<n>:MODE" on page 661

• ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:COUNt:RESet

N (see page 1232)

Command Syntax :SBUS<n>:FLEXray:COUNT:RESET

Resets the FlexRay frame counters.

Errors • "-241, Hardware missing" on page 1193

See Also • ":SBUS<n>:FLEXray:COUNT:NULL" on page 707

• ":SBUS<n>:FLEXray:COUNT:TOTal" on page 710

• ":SBUS<n>:FLEXray:COUNT:SYNC" on page 709

• "Introduction to :SBUS<n> Commands" on page 657

• ":SBUS<n>:MODE" on page 661

• ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:COUNt:SYNC

N (see page 1232)

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 1193

See Also • ":SBUS<n>:FLEXray:COUNt:RESet" on page 708
• ":SBUS<n>:FLEXray:COUNt:TOTal" on page 710
• ":SBUS<n>:FLEXray:COUNt:NULL" on page 707
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:COUNt:TOTal

N (see page 1232)

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 1193

See Also • ":SBUS<n>:FLEXray:COUNt:RESet" on page 708
• ":SBUS<n>:FLEXray:COUNt:TOTal" on page 710
• ":SBUS<n>:FLEXray:COUNt:NULL" on page 707
• ":SBUS<n>:FLEXray:COUNt:SYNC" on page 709
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:FLEXray Commands" on page 702

:SBUS<n>:FLEXray:SOURce

N (see page 1232)

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 959
 - ":SBUS<n>:FLEXray:TRIGger" on page 712
 - ":SBUS<n>:FLEXray:TRIGger:EVENT:TYPE" on page 716
 - ":SBUS<n>:FLEXray:AUTosetup" on page 704

:SBUS<n>:FLEXray:TRIGger

N (see page 1232)

Command Syntax `:SBUS<n>:FLEXray:TRIGger <condition>`
`<condition> ::= {FRAMe | ERRor | EVENT}`

The `:SBUS<n>:FLEXray:TRIGger` command sets the FLEXray trigger on condition:

- FRAMe – triggers on specified frames (without errors).
- ERRor – triggers on selected active error frames and unknown bus conditions.
- 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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:FLEXray:TRIGger:ERRor:TYPE" on page 713
- ":SBUS<n>:FLEXray:TRIGger:EVENT:AUToset" on page 714
- ":SBUS<n>:FLEXray:TRIGger:EVENT:BSS:ID" on page 715
- ":SBUS<n>:FLEXray:TRIGger:EVENT:TYPE" on page 716
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase" on page 717
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition" on page 718
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:ID" on page 719
- ":SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE" on page 720

:SBUS<n>:FLEXray:TRIGger:ERRor:TYPE

N (see page 1232)

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 959
 • ":SBUS<n>:FLEXray:TRIGger" on page 712

:SBUS<n>:FLEXray:TRIGger:EVENT:AUToSet

N (see page 1232)

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 959
- ":SBUS<n>:FLEXray:TRIGger:EVENT:TYPE" on page 716
- ":SBUS<n>:FLEXray:TRIGger:EVENT:BSS:ID" on page 715
- ":SBUS<n>:FLEXray:TRIGger" on page 712
- ":SBUS<n>:FLEXray:BAUDrate" on page 705
- ":TRIGger[:EDGE]:LEVel" on page 986
- ":SBUS<n>:FLEXray:SOURce" on page 711

:SBUS<n>:FLEXray:TRIGger:EVENT:BSS:ID

N (see page 1232)

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 959
 • "[:SBUS<n>:FLEXray:TRIGger:EVENT:TYPE](#)" on page 716
 • "[:SBUS<n>:FLEXray:TRIGger:EVENT:AUToset](#)" on page 714
 • "[:TRIGger:MODE](#)" on page 968
 • "[:SBUS<n>:FLEXray:TRIGger](#)" on page 712

:SBUS<n>:FLEXray:TRIGger:EVENT:TYPE

N (see page 1232)

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 959
 • ":SBUS<n>:FLEXray:TRIGger:EVENT:AUToset" on page 714
 • ":SBUS<n>:FLEXray:TRIGger:EVENT:BSS:ID" on page 715
 • ":SBUS<n>:FLEXray:TRIGger" on page 712
 • ":SBUS<n>:FLEXray:AUTosetup" on page 704
 • ":SBUS<n>:FLEXray:SOURce" on page 711

:SBUS<n>:FLEXray:TRIGger:FRAMe:CCBase

N (see page 1232)

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 959
 • ":SBUS<n>:FLEXray:TRIGger" on page 712

:SBUS<n>:FLEXray:TRIGger:FRAMe:CCRepetition

N (see page 1232)

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 959
• ":SBUS<n>:FLEXray:TRIGger" on page 712

:SBUS<n>:FLEXray:TRIGger:FRAMe:ID

N (see page 1232)

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 959
 • "[:TRIGger:MODE](#)" on page 968
 • "[:SBUS<n>:FLEXray:TRIGger](#)" on page 712

:SBUS<n>:FLEXray:TRIGger:FRAMe:TYPE

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":SBUS<n>:FLEXray:TRIGger" on page 712

:SBUS<n>:I2S Commands

NOTE

These commands are only valid when the I2S serial decode option (Option SND) has been licensed.

Table 107 :SBUS<n>:I2S Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:I2S:ALIGnment <setting> (see page 723)	:SBUS<n>:I2S:ALIGnment? (see page 723)	<setting> ::= {I2S LJ RJ}
:SBUS<n>:I2S:BASE <base> (see page 724)	:SBUS<n>:I2S:BASE?	<base> ::= {DECimal HEX}
:SBUS<n>:I2S:CLOCK:SL OPe <slope> (see page 725)	:SBUS<n>:I2S:CLOCK:SL OPe?	<slope> ::= {NEGative POSitive}
:SBUS<n>:I2S:RWIDTH <receiver> (see page 726)	:SBUS<n>:I2S:RWIDTH?	<receiver> ::= 4-32 in NR1 format
:SBUS<n>:I2S:SOURce:COLock <source> (see page 727)	:SBUS<n>:I2S:SOURce:COLock? (see page 727)	<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:D ATA <source> (see page 728)	:SBUS<n>:I2S:SOURce:D ATA?	<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:W SElect <source> (see page 729)	:SBUS<n>:I2S:SOURce:W SElect? (see page 729)	<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 107 :SBUS<n>:I2S Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:I2S:TRIGger <operator> (see page 730)	:SBUS<n>:I2S:TRIGger? (see page 730)	<operator> ::= {EQUAL NOTEQUAL LESSthan GREATERthan INRange OUTRange INCReasing DECReasing}
:SBUS<n>:I2S:TRIGger: AUDIO <audio_ch> (see page 732)	:SBUS<n>:I2S:TRIGger: AUDIO? (see page 732)	<audio_ch> ::= {RIGHT LEFT EITHer}
:SBUS<n>:I2S:TRIGger: PATtern:DATA <string> (see page 733)	:SBUS<n>:I2S:TRIGger: PATtern:DATA? (see page 734)	<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 735)	:SBUS<n>:I2S:TRIGger: PATtern:FORMAT? (see page 735)	<base> ::= {BINary HEX DECimal}
:SBUS<n>:I2S:TRIGger: RANGE <lower>,<upper> (see page 736)	:SBUS<n>:I2S:TRIGger: RANGE? (see page 736)	<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 738)	:SBUS<n>:I2S:TWIDth? (see page 738)	<word_size> ::= 4-32 in NR1 format
:SBUS<n>:I2S:WSLow <low_def> (see page 739)	:SBUS<n>:I2S:WSLow? (see page 739)	<low_def> ::= {LEFT RIGHT}

:SBUS<n>:I2S:ALIGnment

N (see page 1232)

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 959
- ":SBUS<n>:I2S:CLOCK:SLOPe" on page 725
- ":SBUS<n>:I2S:RWIDth" on page 726
- ":SBUS<n>:I2S:TWIDth" on page 738
- ":SBUS<n>:I2S:WSLow" on page 739

:SBUS<n>:I2S:BASE

N (see page 1232)

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 1193](#)

See Also • ["Introduction to :SBUS<n> Commands" on page 657](#)
 • [":SBUS<n>:I2S Commands" on page 721](#)

:SBUS<n>:I2S:CLOCK:SLOPe

N (see page 1232)

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 959
 • ":SBUS<n>:I2S:ALIGNment" on page 723
 • ":SBUS<n>:I2S:RWIDth" on page 726
 • ":SBUS<n>:I2S:TWIDth" on page 738
 • ":SBUS<n>:I2S:WSLow" on page 739

:SBUS<n>:I2S:RWIDth

N (see page 1232)

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 959
 - "[:SBUS<n>:I2S:ALIGNment](#)" on page 723
 - "[:SBUS<n>:I2S:CLOCK:SLOPe](#)" on page 725
 - "[:SBUS<n>:I2S:TWIDth](#)" on page 738
 - "[:SBUS<n>:I2S:WSLow](#)" on page 739

:SBUS<n>:I2S:SOURce:CLOCK

N (see page 1232)

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 959
- ":SBUS<n>:I2S:SOURce:DATA" on page 728
- ":SBUS<n>:I2S:SOURce:WSELect" on page 729

:SBUS<n>:I2S:SOURce:DATA

N (see page 1232)

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 959
 - ":SBUS<n>:I2S:SOURce:CLOCK" on page 727
 - ":SBUS<n>:I2S:SOURce:WSELect" on page 729

:SBUS<n>:I2S:SOURce:WSELect

N (see page 1232)

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 959
- ":SBUS<n>:I2S:SOURce:CLOCK" on page 727
- ":SBUS<n>:I2S:SOURce:DATA" on page 728

:SBUS<n>:I2S:TRIGger

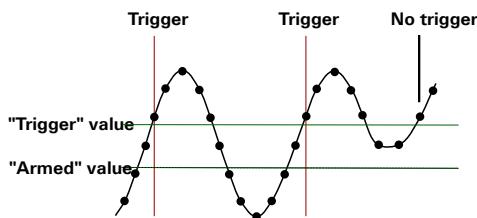
N (see page 1232)

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 decrease 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 959
 - ":SBUS<n>:I2S:TRIGger:AUDio" on page 732
 - ":SBUS<n>:I2S:TRIGger:RANGE" on page 736
 - ":SBUS<n>:I2S:TRIGger:PATTERn:FORMAT" on page 735

:SBUS<n>:I2S:TRIGger:AUDio

N (see page 1232)

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 959
 • ":SBUS<n>:I2S:TRIGger" on page 730

:SBUS<n>:I2S:TRIGger:PATTERn:DATA

N (see page 1232)

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 959
- "[:SBUS<n>:I2S:TRIGger:PATTERn:FORMat](#)" on page 735
- "[:SBUS<n>:I2S:TRIGger](#)" on page 730
- "[:SBUS<n>:I2S:RWIDth](#)" on page 726
- "[:SBUS<n>:I2S:TWIDth](#)" on page 738
- "[:SBUS<n>:I2S:TRIGger:AUDio](#)" on page 732

:SBUS<n>:I2S:TRIGger:PATTERn:FORMAT

N (see page 1232)

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 959
 • ":SBUS<n>:I2S:TRIGger:AUDio" on page 732
 • ":SBUS<n>:I2S:TRIGger" on page 730

:SBUS<n>:I2S:TRIGger:RANGE

N (see page 1232)

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 959

- "[:SBUS<n>:I2S:TRIGger](#)" on page 730
- "[:SBUS<n>:I2S:RWIDth](#)" on page 726
- "[:SBUS<n>:I2S:TWIDth](#)" on page 738
- "[:SBUS<n>:I2S:WSLow](#)" on page 739

:SBUS<n>:I2S:TWIDth

N (see page 1232)

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 959
 • "[:SBUS<n>:I2S:ALIGNment](#)" on page 723
 • "[:SBUS<n>:I2S:CLOCK:SLOPe](#)" on page 725
 • "[:SBUS<n>:I2S:RWIDth](#)" on page 726
 • "[:SBUS<n>:I2S:WSLow](#)" on page 739

:SBUS<n>:I2S:WSLow

N (see page 1232)

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 959
 • "[:SBUS<n>:I2S:ALIGNment](#)" on page 723
 • "[:SBUS<n>:I2S:CLOCK:SLOPe](#)" on page 725
 • "[:SBUS<n>:I2S:RWIDth](#)" on page 726
 • "[:SBUS<n>:I2S:TWIDth](#)" on page 738

: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 108 :SBUS<n>:IIC Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:IIC:ASIZE <size> (see page 741)	:SBUS<n>:IIC:ASIZE? (see page 741)	<size> ::= {BIT7 BIT8}
:SBUS<n>:IIC[:SOURce] :CLOCk <source> (see page 742)	:SBUS<n>:IIC[:SOURce] :CLOCk? (see page 742)	<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 743)	:SBUS<n>:IIC[:SOURce] :DATA? (see page 743)	<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 744)	:SBUS<n>:IIC:TRIGger: PATTern:ADDRess? (see page 744)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATTern:DATA <value> (see page 745)	:SBUS<n>:IIC:TRIGger: PATTern:DATA? (see page 745)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: PATTern:DATA2 <value> (see page 746)	:SBUS<n>:IIC:TRIGger: PATTern:DATA2? (see page 746)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SBUS<n>:IIC:TRIGger: QUALifier <value> (see page 747)	:SBUS<n>:IIC:TRIGger: QUALifier? (see page 747)	<value> ::= {EQUAL NOTEqual LESSthan GREaterthan}
:SBUS<n>:IIC:TRIGger[: TYPE] <type> (see page 748)	:SBUS<n>:IIC:TRIGger[: TYPE]? (see page 748)	<type> ::= {START STOP READ7 READEprom WRITE7 WRITE10 NACKnowledge ANACK R7Data2 W7Data2 REStart}

:SBUS<n>:IIC:ASIZE

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:IIC Commands" on page 740

:SBUS<n>:IIC[:SOURce]:CLOCK

N (see page 1232)

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 959
 - ":SBUS<n>:IIC[:SOURce]:DATA" on page 743

:SBUS<n>:IIC[:SOURce]:DATA

N (see page 1232)

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 959
- ":SBUS<n>:IIC[:SOURce]:CLOCK" on page 742

:SBUS<n>:IIC:TRIGger:PATTERn:ADDResS

N (see page 1232)

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 959
- "[:SBUS<n>:IIC:TRIGger:PATTERn:DATA](#)" on page 745
- "[:SBUS<n>:IIC:TRIGger:PATTERn:DATA2](#)" on page 746
- "[:SBUS<n>:IIC:TRIGger\[:TYPE\]](#)" on page 748

:SBUS<n>:IIC:TRIGger:PATTERn:DATA

N (see page 1232)

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 959
 - "[:SBUS<n>:IIC:TRIGger:PATTERn:ADDResS](#)" on page 744
 - "[:SBUS<n>:IIC:TRIGger:PATTERn:DATA2](#)" on page 746
 - "[:SBUS<n>:IIC:TRIGger\[:TYPE\]](#)" on page 748

:SBUS<n>:IIC:TRIGger:PATTERn:DATA2

N (see page 1232)

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 959
 - "[":SBUS<n>:IIC:TRIGger:PATTERn:ADDResS"](#) on page 744
 - "[":SBUS<n>:IIC:TRIGger:PATTERn:DATA"](#) on page 745
 - "[":SBUS<n>:IIC:TRIGger\[:TYPE\]"](#) on page 748

:SBUS<n>:IIC:TRIGger:QUALifier

N (see page 1232)

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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:IIC:TRIGger[:TYPE]" on page 748

:SBUS<n>:IIC:TRIGger[:TYPE]

N (see [page 1232](#))

Command Syntax :SBUS<n>:IIC:TRIGger[:TYPE] <value>

<value> ::= {STAR | 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 1234](#)).

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 959
• "[":TRIGger:MODE"](#) on page 968

- "[:SBUS<n>:IIC:TRIGger:PATTern:ADDResS](#)" on page 744
- "[:SBUS<n>:IIC:TRIGger:PATTern:DATA](#)" on page 745
- "[:SBUS<n>:IIC:TRIGger:PATTern:DATA2](#)" on page 746
- "[:SBUS<n>:IIC:TRIGger:QUALifier](#)" on page 747
- "["Long Form to Short Form Truncation Rules"](#) on page 1234

:SBUS<n>:LIN Commands

NOTE

These commands are valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Table 109 :SBUS<n>:LIN Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:LIN:PARity { {0 OFF} {1 ON} } (see page 752)	:SBUS<n>:LIN:PARity? (see page 752)	{0 1}
:SBUS<n>:LIN:SAMPLEpo int <value> (see page 753)	:SBUS<n>:LIN:SAMPLEpo int? (see page 753)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:SBUS<n>:LIN:SIGNAL:B AUDrate <baudrate> (see page 754)	:SBUS<n>:LIN:SIGNAL:B AUDrate? (see page 754)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:SBUS<n>:LIN:SOURce <source> (see page 755)	:SBUS<n>:LIN:SOURce? (see page 755)	<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 756)	:SBUS<n>:LIN:STANDARD ? (see page 756)	<std> ::= {LIN13 LIN20}
:SBUS<n>:LIN:SYNCbrea k <value> (see page 757)	:SBUS<n>:LIN:SYNCbrea k? (see page 757)	<value> ::= integer = {11 12 13}
:SBUS<n>:LIN:TRIGger <condition> (see page 758)	:SBUS<n>:LIN:TRIGger? (see page 758)	<condition> ::= {SYNCbreak ID DATA}
:SBUS<n>:LIN:TRIGger: ID <value> (see page 759)	:SBUS<n>:LIN:TRIGger: ID? (see page 759)	<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 109 :SBUS<n>:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:LIN:TRIGger: PATtern:DATA <string> (see page 760)	:SBUS<n>:LIN:TRIGger: PATtern:DATA? (see page 760)	<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 762)	:SBUS<n>:LIN:TRIGger: PATtern:DATA:LENGTH? (see page 762)	<length> ::= integer from 1 to 8 in NR1 format
:SBUS<n>:LIN:TRIGger: PATtern:FORMAT <base> (see page 763)	:SBUS<n>:LIN:TRIGger: PATtern:FORMAT? (see page 763)	<base> ::= {BINary HEX DECimal}

:SBUS<n>:LIN:PARity**N** (see page 1232)**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 1193](#)**See Also** • ["Introduction to :SBUS<n> Commands" on page 657](#)
• [":SBUS<n>:LIN Commands" on page 750](#)

:SBUS<n>:LIN:SAMPLEpoint

N (see page 1232)

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 959

• "[:TRIGger:MODE](#)" on page 968

• "[:SBUS<n>:LIN:TRIGger](#)" on page 758

:SBUS<n>:LIN:SIGNAl:BAUDrate

N (see page 1232)

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 959
 • ":TRIGger:MODE" on page 968
 • ":SBUS<n>:LIN:TRIGger" on page 758
 • ":SBUS<n>:LIN:SIGNAl:DEFinition" on page 1185
 • ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:LIN:SOURce

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":SBUS<n>:LIN:TRIGger" on page 758
 - ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1185

:SBUS<n>:LIN:STANDARD

N (see page 1232)

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 959
 • ":TRIGGER:MODE" on page 968
 • ":SBUS<n>:LIN:SIGNAl:DEFinition" on page 1185
 • ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:LIN:SYNCbreak

N (see page 1232)

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 959
 - "[:TRIGger:MODE](#)" on page 968
 - "[:SBUS<n>:LIN:SIGNal:DEFinition](#)" on page 1185
 - "[:SBUS<n>:LIN:SOURce](#)" on page 755

:SBUS<n>:LIN:TRIGger

N (see page 1232)

Command Syntax `:SBUS<n>:LIN:TRIGger <condition>`

`<condition> ::= {SYNCbreak | ID | DATA | PARityerror | CSUMerror}`

The :SBUS<n>:LIN:TRIGger command sets the LIN trigger condition to be:

- SYNCbreak – Sync Break.
- ID – Frame ID.

Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.

- DATA – Frame ID and Data.

Use the :SBUS<n>:LIN:TRIGger:ID command to specify the frame ID.

Use the :SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth and :SBUS<n>:LIN:TRIGger:PATTERn:DATA commands to specify the data string length and value.

- PARityerror – parity errors.
- CSUMerror – checksum errors.

Query Syntax `:SBUS<n>:LIN:TRIGger?`

The :SBUS<n>:LIN:TRIGger? query returns the current LIN trigger value.

Return Format `<condition><NL>`

`<condition> ::= {SYNC | ID | DATA | PAR | CSUM}`

- "-241, Hardware missing" on page 1193

See Also • "Introduction to :TRIGger Commands" on page 959

- ":TRIGger:MODE" on page 968
- ":SBUS<n>:LIN:TRIGger:ID" on page 759
- ":SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth" on page 762
- ":SBUS<n>:LIN:TRIGger:PATTERn:DATA" on page 760
- ":SBUS<n>:LIN:SIGNAL:DEFinition" on page 1185
- ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:LIN:TRIGger:ID

N (see page 1232)

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 "0xFF" 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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":TRIGger:MODE" on page 968
 • ":SBUS<n>:LIN:TRIGger" on page 758
 • ":SBUS<n>:LIN:SIGNal:DEFinition" on page 1185
 • ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:LIN:TRIGger:PATTERn:DATA

N (see page 1232)

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 959
- ":SBUS<n>:LIN:TRIGger:PATTern:FORMAT" on page 763
- ":SBUS<n>:LIN:TRIGger" on page 758
- ":SBUS<n>:LIN:TRIGger:PATTern:DATA:LENGTH" on page 762

:SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGTH

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:LIN:TRIGger:PATTERn:DATA" on page 760
 • ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:LIN:TRIGger:PATTERn:FORMAT

N (see page 1232)

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 959
 • ":SBUS<n>:LIN:TRIGger:PATTERn:DATA" on page 760
 • ":SBUS<n>:LIN:TRIGger:PATTERn:DATA:LENGth" on page 762

:SBUS<n>:M1553 Commands

NOTE

These commands are valid when the DSOX4AERO MIL-STD-1553 and ARINC 429 triggering and serial decode option (Option AERO) has been licensed.

Table 110 :SBUS<n>:M1553 Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:M1553:AUTose tup (see page 765)	n/a	n/a
:SBUS<n>:M1553:BASE <base> (see page 766)	:SBUS<n>:M1553:BASE? (see page 766)	<base> ::= {BINary HEX}
:SBUS<n>:M1553:SOURce <source> (see page 767)	:SBUS<n>:M1553:SOURce? (see page 767)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:M1553:TRIGge r:PATTERn:DATA <string> (see page 768)	:SBUS<n>:M1553:TRIGge r:PATTERn:DATA? (see page 768)	<string> ::= "nn...n" where n ::= {0 1 X}
:SBUS<n>:M1553:TRIGge r:RTA <value> (see page 769)	:SBUS<n>:M1553:TRIGge r:RTA? (see page 769)	<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:TRIGge r:TYPE <type> (see page 770)	:SBUS<n>:M1553:TRIGge r:TYPE? (see page 770)	<type> ::= {DSTArt DSTOp CSTArt CSTOp RTA PERRor SERRor MERRor RTA11}

:SBUS<n>:M1553:AUTosetup

N (see page 1232)

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 $\pm 1/3$ 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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:M1553:SOURce" on page 767

:SBUS<n>:M1553:BASE

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:M1553 Commands" on page 764

:SBUS<n>:M1553:SOURce

N (see page 1232)

Command Syntax

```
:SBUS<n>:M1553:SOURce <source>
<source> ::= {CHANnel<n>}
```

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

The :SBUS<n>:M1553:SOURce command sets the source of the MIL-STD 1553 signal.

Use the :TRIGger:LEVel:HIGH and :TRIGger:LEVel:LOW commands to set the threshold levels for the selected source.

Query Syntax

```
:SBUS<n>:M1553:TRIGger:SOURce?
```

The :SBUS<n>:M1553:SOURce? query returns the currently set source of the MIL-STD 1553 signal.

Return Format

```
<source><NL>
<source> ::= {CHAN<n>}
```

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

See Also

- "[:TRIGger:LEVel:HIGH](#)" on page 966
- "[:TRIGger:LEVel:LOW](#)" on page 967
- "[:TRIGger:MODE](#)" on page 968
- "[Introduction to :TRIGger Commands](#)" on page 959

:SBUS<n>:M1553:TRIGger:PATTERn:DATA

N (see page 1232)

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 959
 • "[:SBUS<n>:M1553:TRIGger:TYPE](#)" on page 770
 • "[:SBUS<n>:M1553:TRIGger:RTA](#)" on page 769

:SBUS<n>:M1553:TRIGger:RTA

N (see page 1232)

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 (0XX), 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 959
- ":SBUS<n>:M1553:TRIGger:TYPE" on page 770

:SBUS<n>:M1553:TRIGger:TYPE

N (see page 1232)

Command Syntax :SBUS<n>:M1553:TRIGger:TYPE <type>

```
<type> ::= {DSTA | DSTO | CSTA | CSTO | RTA | PERR | SERR
            | MERR | 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 specified as a hex value, and the remaining 11 bits can be specified as a 1, 0, or X (don't care).
- PERRor – (Parity Error) triggers if the (odd) parity bit is incorrect for the data in the word.
- MERRor – (Manchester Error) triggers if a Manchester encoding error is detected.
- SERRor – (Sync Error) triggers if an invalid Sync pulse is found.

Query Syntax :SBUS<n>:M1553:TRIGger:TYPE?

The :SBUS<n>:M1553:TRIGger:TYPE? query returns the currently set MIL-STD- 1553 trigger type.

Return Format <type><NL>

```
<type> ::= {DSTA | DSTO | CSTA | CSTO | RTA | PERR | SERR
            | MERR | RTA11}
```

See Also

- "[Introduction to :TRIGger Commands](#)" on page 959
- "[:SBUS<n>:M1553:TRIGger:RTA](#)" on page 769
- "[:SBUS<n>:M1553:TRIGger:PATTERn:DATA](#)" on page 768
- "[:TRIGger:MODE](#)" on page 968

: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 111 :SBUS<n>:SPI Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:SPI:BITorder <order> (see page 773)	:SBUS<n>:SPI:BITorder ? (see page 773)	<order> ::= {LSBFFirst MSBFFirst}
:SBUS<n>:SPI:CLOCK:SL OPe <slope> (see page 774)	:SBUS<n>:SPI:CLOCK:SL OPe? (see page 774)	<slope> ::= {NEGative POSitive}
:SBUS<n>:SPI:CLOCK:TI Meout <time_value> (see page 775)	:SBUS<n>:SPI:CLOCK:TI Meout? (see page 775)	<time_value> ::= time in seconds in NR3 format
:SBUS<n>:SPI:FRAMing <value> (see page 776)	:SBUS<n>:SPI:FRAMing? (see page 776)	<value> ::= {CHIPselect {NCHipselect NOTC} TIMeout}
:SBUS<n>:SPI:SOURce:CO LOCK <source> (see page 777)	:SBUS<n>:SPI:SOURce:CO LOCK? (see page 777)	<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:FR AME <source> (see page 778)	:SBUS<n>:SPI:SOURce:FR AME? (see page 778)	<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:MO ISO <source> (see page 779)	:SBUS<n>:SPI:SOURce:MO ISO? (see page 779)	<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 111 :SBUS<n>:SPI Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:SPI:SOURce:MOSI <source> (see page 780)	:SBUS<n>:SPI:SOURce:MOSI? (see page 780)	<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 781)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA? (see page 781)	<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 782)	:SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTH? (see page 782)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:PATTern:MOsi:DATA <string> (see page 783)	:SBUS<n>:SPI:TRIGger:PATTern:MOsi:DATA? (see page 783)	<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 784)	:SBUS<n>:SPI:TRIGger:PATTern:MOsi:WIDTH? (see page 784)	<width> ::= integer from 4 to 64 in NR1 format
:SBUS<n>:SPI:TRIGger:TYPE <value> (see page 785)	:SBUS<n>:SPI:TRIGger:TYPE? (see page 785)	<value> ::= {MOSI MISO}
:SBUS<n>:SPI:WIDTH <word_width> (see page 786)	:SBUS<n>:SPI:WIDTH? (see page 786)	<word_width> ::= integer 4-16 in NR1 format

:SBUS<n>:SPI:BITorder

N (see page 1232)

Command Syntax :SBUS<n>:SPI:BITorder <order>

<order> ::= {LSBFFirst | MSBFFirst}

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657

• ":SBUS<n>:MODE" on page 661

• ":SBUS<n>:SPI Commands" on page 771

:SBUS<n>:SPI:CLOCK:SLOPe**N** (see page 1232)**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 959
 • ":SBUS<n>:SPI:CLOCK:TIMEout" on page 775
 • ":SBUS<n>:SPI:SOURce:CLOCK" on page 777

:SBUS<n>:SPI:CLOCK:TIMEout

N (see page 1232)

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 959
 • ":SBUS<n>:SPI:CLOCK:SLOPe" on page 774
 • ":SBUS<n>:SPI:SOURce:CLOCK" on page 777
 • ":SBUS<n>:SPI:FRAMing" on page 776

:SBUS<n>:SPI:FRAMing

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":SBUS<n>:SPI:CLOCK:TIMEout" on page 775
 - ":SBUS<n>:SPI:SOURce:FRAMe" on page 778

:SBUS<n>:SPI:SOURce:CLOCK

N (see page 1232)

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 959
- "[:SBUS<n>:SPI:CLOCK:SLOPe](#)" on page 774
- "[:SBUS<n>:SPI:CLOCK:TIMEout](#)" on page 775
- "[:SBUS<n>:SPI:SOURce:FRAMe](#)" on page 778
- "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 780
- "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 779

:SBUS<n>:SPI:SOURce:FRAMe

N (see page 1232)

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 959
- ":SBUS<n>:SPI:SOURce:CLOCK" on page 777
- ":SBUS<n>:SPI:SOURce:MOSI" on page 780
- ":SBUS<n>:SPI:SOURce:MISO" on page 779
- ":SBUS<n>:SPI:FRAMing" on page 776

:SBUS<n>:SPI:SOURce:MISO

N (see page 1232)

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 959
 - ":SBUS<n>:SPI:SOURce:MOSI" on page 780
 - ":SBUS<n>:SPI:SOURce:CLOCK" on page 777
 - ":SBUS<n>:SPI:SOURce:FRAMe" on page 778
 - ":SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA" on page 781
 - ":SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA" on page 783
 - ":SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh" on page 782
 - ":SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh" on page 784

:SBUS<n>:SPI:SOURce:MOSI

N (see page 1232)

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.

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 959
- "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 779
- "[:SBUS<n>:SPI:SOURce:CLOCK](#)" on page 777
- "[:SBUS<n>:SPI:SOURce:FRAMe](#)" on page 778
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA](#)" on page 781
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA](#)" on page 783
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh](#)" on page 782
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh](#)" on page 784

:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA

N (see page 1232)

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 959
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh](#)" on page 782
- "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 779

:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh

N (see page 1232)

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 959
- ":SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA" on page 781
- ":SBUS<n>:SPI:SOURce:MISO" on page 779

:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA

N (see page 1232)

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 959
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh](#)" on page 784
- "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 780

:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh**N** (see page 1232)**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 959
- ":SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA" on page 783
- ":SBUS<n>:SPI:SOURce:MOSI" on page 780

:SBUS<n>:SPI:TRIGger:TYPE

N (see page 1232)

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 959
 - ":SBUS<n>:SPI:SOURce:MOSI" on page 780
 - ":SBUS<n>:SPI:SOURce:MISO" on page 779
 - ":SBUS<n>:SPI:TRIGger:PATTern:MISO:DATA" on page 781
 - ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:DATA" on page 783
 - ":SBUS<n>:SPI:TRIGger:PATTern:MISO:WIDTh" on page 782
 - ":SBUS<n>:SPI:TRIGger:PATTern:MOSI:WIDTh" on page 784
 - ":TRIGger:MODE" on page 968

:SBUS<n>:SPI:WIDTH

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657

• ":SBUS<n>:MODE" on page 661

• ":SBUS<n>:SPI Commands" on page 771

: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 112 :SBUS<n>:UART Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:UART:BASE <base> (see page 790)	:SBUS<n>:UART:BASE? (see page 790)	<base> ::= {ASCII BINARY HEX}
:SBUS<n>:UART:BAUDrate <baudrate> (see page 791)	:SBUS<n>:UART:BAUDrate? (see page 791)	<baudrate> ::= integer from 100 to 8000000
:SBUS<n>:UART:BITorder <bitorder> (see page 792)	:SBUS<n>:UART:BITorder? (see page 792)	<bitorder> ::= {LSBFIRST MSBFIRST}
n/a	:SBUS<n>:UART:COUNT:EROR? (see page 793)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:COUNT:RESet (see page 794)	n/a	n/a
n/a	:SBUS<n>:UART:COUNT:RXFRAMES? (see page 795)	<frame_count> ::= integer in NR1 format
n/a	:SBUS<n>:UART:COUNT:TXFRAMES? (see page 796)	<frame_count> ::= integer in NR1 format
:SBUS<n>:UART:FRAMing <value> (see page 797)	:SBUS<n>:UART:FRAMing? (see page 797)	<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 798)	:SBUS<n>:UART:PARity? (see page 798)	<parity> ::= {EVEN ODD NONE}
:SBUS<n>:UART:POLaritY <polarity> (see page 799)	:SBUS<n>:UART:POLaritY? (see page 799)	<polarity> ::= {HIGH LOW}

Table 112 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:SOURce: RX <source> (see page 800)	:SBUS<n>:UART:SOURce: RX? (see page 800)	<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 801)	:SBUS<n>:UART:SOURce: TX? (see page 801)	<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 802)	:SBUS<n>:UART:TRIGger :BASE? (see page 802)	<base> ::= {ASCii HEX}
:SBUS<n>:UART:TRIGger :BURSt <value> (see page 803)	:SBUS<n>:UART:TRIGger :BURSt? (see page 803)	<value> ::= {OFF 1 to 4096 in NR1 format}
:SBUS<n>:UART:TRIGger :DATA <value> (see page 804)	:SBUS<n>:UART:TRIGger :DATA? (see page 804)	<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 805)	:SBUS<n>:UART:TRIGger :IDLE? (see page 805)	<time_value> ::= time from 1 us to 10 s in NR3 format
:SBUS<n>:UART:TRIGger :QUALifier <value> (see page 806)	:SBUS<n>:UART:TRIGger :QUALifier? (see page 806)	<value> ::= {EQUAL NOTEqual GREaterthan LESSthan}

Table 112 :SBUS<n>:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:UART:TRIGger :TYPE <value> (see page 807)	:SBUS<n>:UART:TRIGger :TYPE? (see page 807)	<value> ::= {RSTArt RSTOP RDATA RD1 RD0 RDX PARityerror TSTAArt TSTOP TDATa TD1 TD0 TDX}
:SBUS<n>:UART:WIDTH <width> (see page 808)	:SBUS<n>:UART:WIDTH? (see page 808)	<width> ::= {5 6 7 8 9}

:SBUS<n>:UART:BASE**N** (see page 1232)**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 1193**See Also** • "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:BAUDrate

N (see page 1232)

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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:UART:TRIGGER:TYPE" on page 807

:SBUS<n>:UART:BITorder

N (see page 1232)

Command Syntax `:SBUS<n>:UART:BITorder <bitorder>`
`<bitorder> ::= {LSBFFirst | MSBFFirst}`

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. MSBFFirst 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 959
 - "[:TRIGGER:MODE](#)" on page 968
 - "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807
 - "[:SBUS<n>:UART:SOURce:RX](#)" on page 800
 - "[:SBUS<n>:UART:SOURce:TX](#)" on page 801

:SBUS<n>:UART:COUNt:ERRor

N (see page 1232)

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 1193

See Also • ":SBUS<n>:UART:COUNt:RESet" on page 794
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:COUNt:RESet

N (see page 1232)

Command Syntax :SBUS<n>:UART:COUNt:RESet

Resets the UART frame counters.

Errors • "-241, Hardware missing" on page 1193

See Also • ":SBUS<n>:UART:COUNt:ERRor" on page 793
• ":SBUS<n>:UART:COUNt:RXFRAMES" on page 795
• ":SBUS<n>:UART:COUNt:TXFRAMES" on page 796
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:COUNt:RXFRAMES

N (see page 1232)

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 1193

See Also • ":SBUS<n>:UART:COUNt:RESet" on page 794
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:COUNt:TXFRames

N (see page 1232)

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 1193

See Also • ":SBUS<n>:UART:COUNt:RESet" on page 794
• "Introduction to :SBUS<n> Commands" on page 657
• ":SBUS<n>:MODE" on page 661
• ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:FRAMing

N (see page 1232)

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 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:UART Commands" on page 787

:SBUS<n>:UART:PARity**N** (see page 1232)

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 959
 • ":TRIGger:MODE" on page 968
 • ":SBUS<n>:UART:TRIGger:TYPE" on page 807

:SBUS<n>:UART:POLarity

N (see page 1232)

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 959
• ":TRIGger:MODE" on page 968
• ":SBUS<n>:UART:TRIGger:TYPE" on page 807

:SBUS<n>:UART:SOURce:RX

N (see page 1232)

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 959
 - ":TRIGGER:MODE" on page 968
 - ":SBUS<n>:UART:TRIGGER:TYPE" on page 807
 - ":SBUS<n>:UART:BITorder" on page 792

:SBUS<n>:UART:SOURce:TX

N (see page 1232)

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 959
 - ":TRIGGER:MODE" on page 968
 - ":SBUS<n>:UART:TRIGger:TYPE" on page 807
 - ":SBUS<n>:UART:BITorder" on page 792

:SBUS<n>:UART:TRIGger:BASE

(see page 1232)

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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:UART:TRIGger:DATA" on page 804

:SBUS<n>:UART:TRIGger:BURSt

N (see page 1232)

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 959
 - "[:TRIGger:MODE](#)" on page 968
 - "[:SBUS<n>:UART:TRIGger:IDLE](#)" on page 805
 - "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807

:SBUS<n>:UART:TRIGger:DATA

N (see page 1232)

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 959
- "[:TRIGger:MODE](#)" on page 968
- "[:SBUS<n>:UART:TRIGger:BASE](#)" on page 802
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807

:SBUS<n>:UART:TRIGger:IDLE

N (see page 1232)

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 959
- "[:TRIGger:MODE](#)" on page 968
- "[:SBUS<n>:UART:TRIGger:BURSt](#)" on page 803
- "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807

:SBUS<n>:UART:TRIGger:QUALifier**N** (see page 1232)**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 959• "[:TRIGger:MODE](#)" on page 968• "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807

:SBUS<n>:UART:TRIGger:TYPE

N (see page 1232)

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 959
 - "[:TRIGger:MODE](#)" on page 968
 - "[:SBUS<n>:UART:TRIGger:DATA](#)" on page 804
 - "[:SBUS<n>:UART:TRIGger:QUALifier](#)" on page 806
 - "[:SBUS<n>:UART:WIDTH](#)" on page 808

:SBUS<n>:UART:WIDTH**N** (see page 1232)

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 959
 • "[:TRIGger:MODE](#)" on page 968
 • "[:SBUS<n>:UART:TRIGger:TYPE](#)" on page 807

:SBUS<n>:USB Commands

NOTE

These commands are only valid when a USB 2.0 triggering and serial decode option (Option USF or Option U2H) has been licensed.

Table 113 :SBUS<n>:USB Commands Summary

Command	Query	Options and Query Returns
:SBUS<n>:USB:BASE <base> (see page 812)	:SBUS<n>:USB:BASE? (see page 812)	<base> ::= {ASCii BINary DECimal HEX}
:SBUS<n>:USB:SOURce:D MINus <source> (see page 813)	:SBUS<n>:USB:SOURce:D MINus? (see page 813)	<source> ::= {CHANnel<n>} 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>:USB:SOURce:D PLus <source> (see page 814)	:SBUS<n>:USB:SOURce:D PLus? (see page 814)	<source> ::= {CHANnel<n>} 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>:USB:SOURce:D IFFerential <source> (see page 815)	:SBUS<n>:USB:SOURce:D IFFerential? (see page 815)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:SBUS<n>:USB:SPEed <speed> (see page 816)	:SBUS<n>:USB:SPEed? (see page 816)	<speed> ::= {LOW FULL HIGH}
:SBUS<n>:USB:TRIGger <condition> (see page 817)	:SBUS<n>:USB:TRIGger? (see page 817)	<condition> ::= {SOP EOP ENTersuspend EXITsuspend RESet TOKen DATA HANDshake SPECIAL ALLerrors PIDerror CRC5error CRC16error GLITcher STUFFerror SE1error}
:SBUS<n>:USB:TRIGger: ADDRess <string> (see page 818)	:SBUS<n>:USB:TRIGger: ADDRess? (see page 818)	<string> ::= "nnnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}

Table 113 :SBUS<n>:USB Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:USB:TRIGger: CRC <string> (see page 819)	:SBUS<n>:USB:TRIGger: CRC? (see page 819)	<string> ::= "nnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: DATA <string> (see page 820)	:SBUS<n>:USB:TRIGger: DATA? (see page 820)	<string> ::= "nnnn..." where n ::= {0 1 X} <string> ::= "0xn..." where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: DATA:LENGTH <value> (see page 821)	:SBUS<n>:USB:TRIGger: DATA:LENGTH? (see page 821)	<length> ::= data length between 1-20
:SBUS<n>:USB:TRIGger: ENDPoint <string> (see page 822)	:SBUS<n>:USB:TRIGger: ENDPoint? (see page 822)	<string> ::= "nnnn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: ET <string> (see page 823)	:SBUS<n>:USB:TRIGger: ET? (see page 823)	<string> ::= "nn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 2 3 X \$}
:SBUS<n>:USB:TRIGger: FRAMe <string> (see page 824)	:SBUS<n>:USB:TRIGger: FRAMe? (see page 824)	<string> ::= "nnnnnnnnnnnn" where n ::= {0 1 X} <string> ::= "0xnnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: HADDress <string> (see page 825)	:SBUS<n>:USB:TRIGger: HADDress? (see page 825)	<string> ::= "nnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: PCHeck <string> (see page 826)	:SBUS<n>:USB:TRIGger: PCHeck? (see page 826)	<string> ::= "nnnn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGger: PID:DATA <pid> (see page 827)	:SBUS<n>:USB:TRIGger: PID:DATA? (see page 827)	<pid> ::= {DATA0 DATA1 DATA2 MDATA}
:SBUS<n>:USB:TRIGger: PID:HANDshake <pid> (see page 828)	:SBUS<n>:USB:TRIGger: PID:HANDshake? (see page 828)	<pid> ::= {ACK NAK STALL NYET}
:SBUS<n>:USB:TRIGger: PID:SPECial <pid> (see page 829)	:SBUS<n>:USB:TRIGger: PID:SPECIAL? (see page 829)	<pid> ::= {PING PRE ERR SPLIt}

Table 113 :SBUS<n>:USB Commands Summary (continued)

Command	Query	Options and Query Returns
:SBUS<n>:USB:TRIGger: PID:TOKen <pid> (see page 830)	:SBUS<n>:USB:TRIGger: PID:TOKen? (see page 830)	<pid> ::= {OUT IN SETup SOF}
:SBUS<n>:USB:TRIGGER: PORT <string> (see page 831)	:SBUS<n>:USB:TRIGger: PORT? (see page 831)	<string> ::= "nnnnnnn" where n ::= {0 1 X} <string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SBUS<n>:USB:TRIGGER: SC <string> (see page 832)	:SBUS<n>:USB:TRIGger: SC? (see page 832)	<string> ::= "n" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 X \$}
:SBUS<n>:USB:TRIGGER: SEU <string> (see page 833)	:SBUS<n>:USB:TRIGger: SEU? (see page 833)	<string> ::= "nn" where n ::= {0 1 X} <string> ::= "0xn" where n ::= {0 1 2 3 X \$}

:SBUS<n>:USB:BASE

N (see page 1232)

Command Syntax :SBUS<n>:USB:BASE <base>

<base> ::= {ASCii | BINary | DECimal | HEX}

The :SBUS<n>:USB:BASE command determines the base to use for the USB decode and Lister display, controlling how the data or payload field is displayed.

All other fields are displayed in hex

Query Syntax :SBUS<n>:USB:BASE?

The :SBUS<n>:USB:BASE? query returns the currently specified base.

Return Format <base><NL>

<base> ::= {ASC | BIN | DEC | HEX}

:SBUS<n>:USB:SOURce:DMINus

N (see page 1232)

Command Syntax

```
:SBUS<n>:USB:SOURce:DMINus <source>
<source> ::= {CHANnel<n>} 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 :SBUS<n>:USB:SOURce:DMINus command specifies which signal is used as the USB D- source for the selected bus.

The D- source is only used for Low and Full Speed USB.

Query Syntax

```
:SBUS<n>:USB:SOURce:DMINus?
```

The :SBUS<n>:USB:SOURce:DMINus? query returns the specified D- source.

Return Format

```
<source><NL>
<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "[:SBUS<n>:USB:SOURce:DPLus](#)" on page 814
- "[:SBUS<n>:USB:SPEed](#)" on page 816

:SBUS<n>:USB:SOURce:DPLus

N (see page 1232)

Command Syntax

```
:SBUS<n>:USB:SOURce:DPLus <source>
<source> ::= {CHANnel<n>} 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 :SBUS<n>:USB:SOURce:DPLus command specifies which signal is used as the USB D+ source for the selected bus.

The D+ source is only used for Low and Full Speed USB.

Query Syntax

```
:SBUS<n>:USB:SOURce:DPLus?
```

The :SBUS<n>:USB:SOURce:DPLus? query returns the specified D+ source.

Return Format

```
<source><NL>
<source> ::= {CHAN<n> | DIG<d>}
```

See Also

- "[:SBUS<n>:USB:SOURce:DMINus](#)" on page 813
- "[:SBUS<n>:USB:SPEed](#)" on page 816

:SBUS<n>:USB:SOURce:DIFFerential

N (see page 1232)

Command Syntax :SBUS<n>:USB:SOURce:DIFFerential <source>

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

The :SBUS<n>:USB:SOURce:DIFFerential command specifies which signal is used as the differential source for the selected bus.

Differential sources are only used for High Speed USB.

Query Syntax :SBUS<n>:USB:SOURce:DIFFerential?

The :SBUS<n>:USB:SOURce:DIFFerential? query returns the specified differential source.

Return Format <source><NL>

```
<source> ::= {CHAN<n>}
```

See Also • "[:SBUS<n>:USB:SPEed](#)" on page 816

:SBUS<n>:USB:SPEed

N (see page 1232)

Command Syntax :SBUS<n>:USB:SPEed <speed>

<speed> ::= {LOW | FULL | HIGH}

The :SBUS<n>:USB:SPEed command specifies the speed of the USB interface for the selected bus.

Query Syntax :SBUS<n>:USB:SPEed?

The :SBUS<n>:USB:SPEed? query returns the speed setting.

Return Format <speed><NL>

<speed> ::= {LOW | FULL | HIGH}

- See Also**
- "[:SBUS<n>:USB:SOURce:DPLus](#)" on page 814
 - "[:SBUS<n>:USB:SOURce:DMINus](#)" on page 813
 - "[:SBUS<n>:USB:SOURce:DIFFerential](#)" on page 815

:SBUS<n>:USB:TRIGger

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger <condition>

```
<condition> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet
                  | TOKen | DATA | HANDshake | SPECIAL | ALLerrors
                  | PIDerror | CRC5error | CRC16error | GLITcherror
                  | STUFFerror | SE1error}
```

The :SBUS<n>:USB:TRIGger command specifies the USB trigger mode for the selected bus.

Query Syntax :SBUS<n>:USB:TRIGger?

The :SBUS<n>:USB:TRIGger? query returns the specified USB trigger mode.

Return Format <condition><NL>

```
<condition> ::= {SOP | EOP | ENT | EXIT | RES | TOK | DATA | HAND
                  | SPEC | ALL | PID | CRC5 | CRC16 | GLIT | STUFF
                  | SE1}
```

See Also • "[:SBUS<n>:USB:TRIGger:PID:DATA](#)" on page 827

• "[:SBUS<n>:USB:TRIGger:PID:HANDshake](#)" on page 828

• "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829

• "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:ADDRess

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:ADDRess <string>

<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | x | \$}

The :SBUS<n>:USB:TRIGger:ADDRess command specifies the 7-bit Address portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:ADDRess?

The :SBUS<n>:USB:TRIGger:ADDRess? query returns the specified Address portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}

See Also

- "[:SBUS<n>:USB:TRIGger:PID:SPEcial](#)" on page 829
- "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:CRC

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:CRC <string>

<string> ::= "nnnnn" where n ::= {0 | 1 | x}

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | x | \$}

The :SBUS<n>:USB:TRIGger:CRC command specifies the CRC portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:CRC?

The :SBUS<n>:USB:TRIGger:CRC? query returns the specified CRC portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnnn" where n ::= {0 | 1 | x}

See Also

- "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829
- "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:DATA

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:DATA <string>

<string> ::= "nnnn..." where n ::= {0 | 1 | X}

<string> ::= "0xn..." where n ::= {0,...,9 | A,...,F | X | \$}

The :SBUS<n>:USB:TRIGger:DATA command specifies the Data portion of the trigger for the selected bus, in binary or hex.

See the :SBUS<n>:USB:TRIGger:DATA:LENGTH command for setting the length of the data value.

Query Syntax :SBUS<n>:USB:TRIGger:DATA?

The :SBUS<n>:USB:TRIGger:DATA? query returns the specified Data portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnn..." where n ::= {0 | 1 | X}

See Also • "[:SBUS<n>:USB:TRIGger:DATA:LENGTH](#)" on page 821

:SBUS<n>:USB:TRIGger:DATA:LENGTH

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:DATA:LENGTH <length>
<length> ::= data length between 1-20

The :SBUS<n>:USB:TRIGger:DATA:LENGTH command specifies the data length in bytes.

Query Syntax :SBUS<n>:USB:TRIGger:DATA:LENGTH?

The :SBUS<n>:USB:TRIGger:DATA:LENGTH? query returns the specified data length.

Return Format <length><NL>
<length> ::= data length between 1-20

See Also • "[:SBUS<n>:USB:TRIGger:DATA](#)" on page 820

:SBUS<n>:USB:TRIGger:ENDPoint

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:ENDPoint <string>

<string> ::= "nnnn" where n ::= {0 | 1 | x}

<string> ::= "0xn" where n ::= {0,...,9 | A,...,F | x | \$}

The :SBUS<n>:USB:TRIGger:ENDPoint command specifies the 4-bit Endpoint portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:ENDPoint?

The :SBUS<n>:USB:TRIGger:ENDPoint? query returns the specified Endpoint portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnn" where n ::= {0 | 1 | x}

See Also • "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829
• "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:ET

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:ET <string>

<string> ::= "nn" where n ::= {0 | 1 | X}

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | X | \$}

The :SBUS<n>:USB:TRIGger:ET command specifies the 2-bit ET portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:ET?

The :SBUS<n>:USB:TRIGger:ET? query returns the specified ET portion of the trigger.

Return Format <string><NL>

<string> ::= "nn" where n ::= {0 | 1 | X}

See Also • "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829

:SBUS<n>:USB:TRIGger:FRAMe

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:FRAMe <string>

```
<string> ::= "nnnnnnnnnnm" where n ::= {0 | 1 | X}  
<string> ::= "0xnnn" where n ::= {0,...,9 | A,...,F | X | $}
```

The :SBUS<n>:USB:TRIGger:FRAMe command specifies the 11-bit Frame portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:FRAMe?

The :SBUS<n>:USB:TRIGger:FRAMe? query returns the specified Frame portion of the trigger.

Return Format <string><NL>

```
<string> ::= "nnnnnnnnnnn" where n ::= {0 | 1 | X}
```

See Also • "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:HADDress

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:HADDress <string>

<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}

<string> ::= "0xnm" where n ::= {0,...,9 | A,...,F | x | \$}

The :SBUS<n>:USB:TRIGger:HADDress command specifies the 7-bit Hub Address portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:HADDress?

The :SBUS<n>:USB:TRIGger:HADDress? query returns the specified Hub Address portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}

See Also • "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829

:SBUS<n>:USB:TRIGger:PCHeck

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:PCHeck <string>

<string> ::= "nnnn" where n ::= {0 | 1 | X}

<string> ::= "0xn" where n ::= {0,...,9 | A,...,F | X | \$}

The :SBUS<n>:USB:TRIGger:PCHeck command specifies the 4-bit PID check portion of the trigger for the selected bus, in binary or hex.

See the :SBUS<n>:USB:TRIGger:PID commands for setting the PID.

Query Syntax :SBUS<n>:USB:TRIGger:PCHeck?

The :SBUS<n>:USB:TRIGger:PCHeck? query returns the specified PID check portion of the trigger.

Return Format <string><NL>

<string> ::= "nnnn" where n ::= {0 | 1 | X}

- See Also**
- "[:SBUS<n>:USB:TRIGger:PID:DATA](#)" on page 827
 - "[:SBUS<n>:USB:TRIGger:PID:HANDshake](#)" on page 828
 - "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829
 - "[:SBUS<n>:USB:TRIGger:PID:TOKen](#)" on page 830

:SBUS<n>:USB:TRIGger:PID:DATA

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:PID:DATA <pid>

<pid> ::= {DATA0 | DATA1 | DATA2 | MDATA}

The :SBUS<n>:USB:TRIGger:PID:DATA command specifies the USB data PID to trigger on for the selected bus.

The specified PID does not include the PID check value, which is specified using the :SBUS<n>:USB:TRIGger:PIDCheck command.

The DATA USB trigger mode can be selected using the :SBUS<n>:USB:TRIGger command.

Query Syntax :SBUS<n>:USB:TRIGger:PID:DATA?

The :SBUS<n>:USB:TRIGger:PID:DATA? query returns the specified data PID.

Return Format <pid><NL>

<pid> ::= {DATA0 | DATA1 | DATA2 | MDAT}

See Also

- "[:SBUS<n>:USB:TRIGger:PCheck](#)" on page 826
- "[:SBUS<n>:USB:TRIGger](#)" on page 817

:SBUS<n>:USB:TRIGger:PID:HANDshake

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:PID:HANDshake <pid>
 <pid> ::= {ACK | NAK | STALL | NYET}

The :SBUS<n>:USB:TRIGger:PID:HANDshake command specifies the USB handshake PID to trigger on for the selected bus.

The specified PID does not include the PID check value, which is specified using the :SBUS<n>:USB:TRIGger:PIDCheck command.

The HANDshake USB trigger mode can be selected using the :SBUS<n>:USB:TRIGger command.

Query Syntax :SBUS<n>:USB:TRIGger:PID:HANDshake?

The :SBUS<n>:USB:TRIGger:PID:HANDshake? query returns the specified handshake PID.

Return Format <pid><NL>
 <pid> ::= {ACK | NAK | STAL | NYET}

See Also • "[:SBUS<n>:USB:TRIGger:PCheck](#)" on page 826
 • "[:SBUS<n>:USB:TRIGger](#)" on page 817

:SBUS<n>:USB:TRIGger:PID:SPECial

N (see page 1232)

Command Syntax :SBUS<n>:USB:TRIGger:PID:SPECial <pid>
 <pid> ::= {PING | PRE | ERR | SPL}

The :SBUS<n>:USB:TRIGger:PID:SPECial command specifies the USB special PID to trigger on for the selected bus.

The specified PID does not include the PID check value, which is specified using the :SBUS<n>:USB:TRIGger:PIDCheck command.

The SPECial USB trigger mode can be selected using the :SBUS<n>:USB:TRIGger command.

Query Syntax :SBUS<n>:USB:TRIGger:PID:SPECial?

The :SBUS<n>:USB:TRIGger:PID:SPECial? query returns the specified special PID.

Return Format <pid><NL>
 <pid> ::= {PING | PRE | ERR | SPL}

See Also • "[:SBUS<n>:USB:TRIGger:PCheck](#)" on page 826
 • "[:SBUS<n>:USB:TRIGger](#)" on page 817

:SBUS<n>:USB:TRIGger:PID:TOKen

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:PID:TOKen <pid>
 <pid> ::= {OUT | IN | SETUP | SOF}

The :SBUS<n>:USB:TRIGger:PID:TOKen command specifies the USB token PID to trigger on for the selected bus.

The specified PID does not include the PID check value, which is specified using the :SBUS<n>:USB:TRIGger:PIDCheck command.

The TOKen USB trigger mode can be selected using the :SBUS<n>:USB:TRIGger command.

Query Syntax :SBUS<n>:USB:TRIGger:PID:TOKen?

The :SBUS<n>:USB:TRIGger:PID:TOKen? query returns the specified token PID.

Return Format <pid><NL>
 <pid> ::= {OUT | IN | SETUP | SOF}

See Also • "[:SBUS<n>:USB:TRIGger:PCheck](#)" on page 826
 • "[:SBUS<n>:USB:TRIGger](#)" on page 817

:SBUS<n>:USB:TRIGger:PORT

N (see page 1232)

Command Syntax

```
:SBUS<n>:USB:TRIGger:PORT <string>
<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}
<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | x | $}
```

The :SBUS<n>:USB:TRIGger:PORT command specifies the 7-bit Port portion of the trigger for the selected bus, in binary or hex.

Query Syntax

```
:SBUS<n>:USB:TRIGger:PORT?
```

The :SBUS<n>:USB:TRIGger:PORT? query returns the specified Port portion of the trigger.

Return Format

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

```
<string> ::= "nnnnnnn" where n ::= {0 | 1 | x}
```

See Also

- "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829

:SBUS<n>:USB:TRIGger:SC

N (see page 1232)

Command :SBUS<n>:USB:TRIGger:SC <string>

SyntaxS

```
<string> ::= "n" where n ::= {0 | 1 | X}
<string> ::= "0xn" where n ::= {0 | 1 | X | $}
```

The :SBUS<n>:USB:TRIGger:SC command specifies the 1-bit SC portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:SC?

The :SBUS<n>:USB:TRIGger:SC? query returns the specified SC portion of the trigger.

Return Format <string><NL>

```
<string> ::= "nnnn" where n ::= {0 | 1 | X}
```

See Also • "[:SBUS<n>:USB:TRIGger:PID:SPECial](#)" on page 829

:SBUS<n>:USB:TRIGger:SEU

N (see [page 1232](#))

Command Syntax :SBUS<n>:USB:TRIGger:SEU <string>

<string> ::= "nn" where n ::= {0 | 1 | x}

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | x | \$}

The :SBUS<n>:USB:TRIGger:SEU command specifies the 2-bit S and E or U portion of the trigger for the selected bus, in binary or hex.

Query Syntax :SBUS<n>:USB:TRIGger:SEU?

The :SBUS<n>:USB:TRIGger:SEU? query returns the specified S and E or U portion of the trigger.

Return Format <string><NL>

<string> ::= "nn" where n ::= {0 | 1 | x}

See Also • [":SBUS<n>:USB:TRIGger:PID:SPECial"](#) on page 829

29 **:SEARch Commands**

Control the event search modes and parameters for each search type. See:

- ["General :SEARch Commands" on page 836](#)
- [":SEARch:EDGE Commands" on page 841](#)
- [":SEARch:GLITch Commands" on page 844 \(Pulse Width search\)](#)
- [":SEARch:RUNT Commands" on page 851](#)
- [":SEARch:TRANSition Commands" on page 856](#)
- [":SEARch:SERial:A429 Commands" on page 861](#)
- [":SEARch:SERial:CAN Commands" on page 867](#)
- [":SEARch:SERial:FLEXray Commands" on page 877](#)
- [":SEARch:SERial:I2S Commands" on page 883](#)
- [":SEARch:SERial:IIC Commands" on page 889](#)
- [":SEARch:SERial:LIN Commands" on page 896](#)
- [":SEARch:SERial:M1553 Commands" on page 902](#)
- [":SEARch:SERial:SPI Commands" on page 906](#)
- [":SEARch:SERial:UART Commands" on page 910](#)
- [":SEARch:SERial:USB Commands" on page 914](#)



General :SEARch Commands

Table 114 General :SEARch Commands Summary

Command	Query	Options and Query Returns
n/a	:SEARch:COUNT? (see page 837)	<count> ::= an integer count value
:SEARch:EVENT <event_number> (see page 838)	:SEARch:EVENT? (see page 838)	<event_number> ::= the integer number of a found search event
:SEARch:MODE <value> (see page 839)	:SEARch:MODE? (see page 839)	<value> ::= {EDGE GLITch RUNT TRANSition SERial{1 2}}
:SEARch:STATE <value> (see page 840)	:SEARch:STATE? (see page 840)	<value> ::= {{0 OFF} {1 ON}}

:SEARCh:COUNT

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
 - “:SEARCh:EVENT” on page 838
 - “:SEARCh:STATE” on page 840
 - “:SEARCh:MODE” on page 839

:SEARCh:EVENT

N (see page 1232)

Command Syntax :SEARCh:EVENT <event_number>

<event_number> ::= the integer number of a found search event

The :SEARCh:EVENT command navigates to a found search event.

If the :SEARCh:STATe is ON, the horizontal position is changed so that the specified event is located at the time reference.

Query Syntax :SEARCh:EVENT?

The :SEARCh:EVENT? query returns the currently selected event number.

Return Format <event_number><NL>

<event_number> ::= the integer number of a found search event

- See Also**
- [Chapter 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:COUNt" on page 837](#)
 - [":SEARCh:STATe" on page 840](#)
 - [":SEARCh:MODE" on page 839](#)

:SEARCh:MODE

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:STATe" on page 840](#)
 - [":SEARCh:COUNT" on page 837](#)
 - [":SEARCh:EVENT" on page 838](#)

:SEARch:STATE

N (see page 1232)

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 the current setting.

Return Format `<value><NL>`

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

- See Also**
- [Chapter 29, “:SEARch Commands,” starting on page 835](#)
 - [":SEARch:MODE" on page 839](#)
 - [":SEARch:COUNT" on page 837](#)
 - [":SEARch:EVENT" on page 838](#)

:SEARch:EDGE Commands

Table 115 :SEARch:EDGE Commands Summary

Command	Query	Options and Query Returns
:SEARch:EDGE:SLOPe <slope> (see page 842)	:SEARch:EDGE:SLOPe? (see page 842)	<slope> ::= {POSitive NEGative EITHer}
:SEARch:EDGE:SOURce <source> (see page 843)	:SEARch:EDGE:SOURce? (see page 843)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:EDGE:SLOPe

N (see [page 1232](#))

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 29, “:SEARch Commands,” starting on page 835](#)

:SEARCh:EDGE:SOURce

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)

:SEARCh:GLITCh Commands

Table 116 :SEARCh:GLITCh Commands Summary

Command	Query	Options and Query Returns
:SEARCh:GLITCh:GREaterthan <greater_than_time>[suffix] (see page 845)	:SEARCh:GLITCh:GREaterthan? (see page 845)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:SEARCh:GLITCh:LESSthan <less_than_time>[suffix] (see page 846)	:SEARCh:GLITCh:LESSthan? (see page 846)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:SEARCh:GLITCh:POLarity <polarity> (see page 847)	:SEARCh:GLITCh:POLarity? (see page 847)	<polarity> ::= {POSitive NEGative}
:SEARCh:GLITCh:QUALifier <qualifier> (see page 848)	:SEARCh:GLITCh:QUALifier? (see page 848)	<qualifier> ::= {GREaterthan LESSthan RANGE}
:SEARCh:GLITCh:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 849)	:SEARCh:GLITCh:RANGE? (see page 849)	<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 850)	:SEARCh:GLITCh:SOURce? (see page 850)	<source> ::= CHANNEL<n> <n> ::= 1 to (# analog channels) in NR1 format

:SEARch:GLITch:GREaterthan

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:GLITch:SOURce” on page 850
- “:SEARch:GLITch:QUALifier” on page 848
- “:SEARch:MODE” on page 839

:SEARch:GLITch:LESSthan

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
• [":SEARch:GLITch:SOURce" on page 850](#)
• [":SEARch:GLITch:QUALifier" on page 848](#)
• [":SEARch:MODE" on page 839](#)

:SEARch:GLITch:POLarity

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
 • [":SEARch:MODE" on page 839](#)
 • [":SEARch:GLITch:SOURce" on page 850](#)

:SEARch:GLITch:QUALifier**N** (see page 1232)**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 29, “:SEARch Commands,” starting on page 835](#)
 - [":SEARch:GLITch:SOURce"](#) on page 850
 - [":SEARch:MODE"](#) on page 839

:SEARch:GLITch:RANGe

N (see page 1232)

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

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 29, “:SEARch Commands,” starting on page 835](#)
 - [":SEARch:GLITch:SOURce" on page 850](#)
 - [":SEARch:GLITch:QUALifier" on page 848](#)
 - [":SEARch:MODE" on page 839](#)

:SEARch:GLITch:SOURce**N** (see page 1232)**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 29, “:SEARch Commands,” starting on page 835](#)
- [Chapter 29, “:SEARch Commands,” starting on page 835](#)
- [“:SEARch:MODE” on page 839](#)
- [“:SEARch:GLITch:POLarity” on page 847](#)
- [“:SEARch:GLITch:QUALifier” on page 848](#)
- [“:SEARch:GLITch:RANGE” on page 849](#)

:SEARCh:RUNT Commands

Table 117 :SEARCh:RUNT Commands Summary

Command	Query	Options and Query Returns
:SEARCh:RUNT:POLarity <polarity> (see page 852)	:SEARCh:RUNT:POLarity? ? (see page 852)	<polarity> ::= {POSitive NEGative EITHer}
:SEARCh:RUNT:QUALifie r <qualifier> (see page 853)	:SEARCh:RUNT:QUALifie r? ? (see page 853)	<qualifier> ::= {GREaterthan LESSthan NONE}
:SEARCh:RUNT:SOURce <source> (see page 854)	:SEARCh:RUNT:SOURce? (see page 854)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARCh:RUNT:TIME <time>[suffix] (see page 855)	:SEARCh:RUNT:TIME? (see page 855)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

:SEARch:RUNT:POLarity**N** (see page 1232)**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 29, “:SEARch Commands,” starting on page 835
 - “:SEARch:MODE” on page 839
 - “:SEARch:RUNT:SOURce” on page 854

:SEARCh:RUNT:QUALifier

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [“:SEARCh:MODE” on page 839](#)

:SEARch:RUNT:SOURce

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
• [“:SEARch:RUNT:POLarity” on page 852](#)

:SEARCh:RUNT:TIME

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:RUNT:QUALifier"](#) on page 853

:SEARch:TRANsition Commands

Table 118 :SEARch:TRANsition Commands Summary

Command	Query	Options and Query Returns
:SEARch:TRANsition:QUALifier <qualifier> (see page 857)	:SEARch:TRANsition:QUALifier? (see page 857)	<qualifier> ::= {GREaterthan LESSthan}
:SEARch:TRANsition:SLOPe <slope> (see page 858)	:SEARch:TRANsition:SLOPe? (see page 858)	<slope> ::= {NEGative POSitive}
:SEARch:TRANsition:SOURCE <source> (see page 859)	:SEARch:TRANsition:SOURCE? (see page 859)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:SEARch:TRANsition:TIME <time>[suffix] (see page 860)	:SEARch:TRANsition:TIME? (see page 860)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

:SEARch:TRANSition:QUALifier

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
• [“:SEARch:MODE” on page 839](#)
• [“:SEARch:TRANSition:TIME” on page 860](#)

:SEARch:TRANSition:SLOPe**N** (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
- [":SEARch:MODE" on page 839](#)
- [":SEARch:TRANSition:SOURce" on page 859](#)
- [":SEARch:TRANSition:TIME" on page 860](#)

:SEARch:TRANSition:SOURce

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
 - “:SEARch:MODE” on page 839
 - “:SEARch:TRANSition:SLOPe” on page 858

:SEARch:TRANSition:TIME

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
 - [":SEARch:TRANSition:QUALifier" on page 857](#)

:SEARch:SERial:A429 Commands

Table 119 :SEARch:SERial:A429 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:A429:L ABel <value> (see page 862)	:SEARch:SERial:A429:L ABel? (see page 862)	<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:M ODE <condition> (see page 863)	:SEARch:SERial:A429:M ODE? (see page 863)	<condition> ::= {LABEL LBITS PERRor WERRor GERRor WGERRors ALLerrors}
:SEARch:SERial:A429:P ATTern:DATA <string> (see page 864)	:SEARch:SERial:A429:P ATTern:DATA? (see page 864)	<string> ::= "nn...n" where n ::= {0 1}, length depends on FORMat
:SEARch:SERial:A429:P ATTern:SDI <string> (see page 865)	:SEARch:SERial:A429:P ATTern:SDI? (see page 865)	<string> ::= "nn" where n ::= {0 1}, length always 2 bits
:SEARch:SERial:A429:P ATTern:SSM <string> (see page 866)	:SEARch:SERial:A429:P ATTern:SSM? (see page 866)	<string> ::= "nn" where n ::= {0 1}, length always 2 bits

:SEARch:SERial:A429:LABEL

N (see [page 1232](#))

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 1193](#)

See Also • ["Introduction to :TRIGger Commands" on page 959](#)
 • [":SEARch:SERial:A429:MODE" on page 863](#)

:SEARCh:SERial:A429:MODE

N (see page 1232)

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 | WGERR | ALL}
```

Errors • "-241, Hardware missing" on page 1193

See Also • "Introduction to :SBUS<n> Commands" on page 657
 • ":SBUS<n>:MODE" on page 661
 • ":SEARCh:SERial:A429:LABEL" on page 862
 • ":SEARCh:SERial:A429:PATTern:DATA" on page 864
 • ":SEARCh:SERial:A429:PATTern:SDI" on page 865
 • ":SEARCh:SERial:A429:PATTern:SSM" on page 866
 • ":SBUS<n>:A429:SOURce" on page 671

:SEARch:SERial:A429:PATTern:DATA

N (see page 1232)

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 1193

See Also

- "Introduction to :TRIGger Commands" on page 959
- ":SEARch:SERial:A429:MODE" on page 863
- ":SEARch:SERial:A429:PATTern:SDI" on page 865
- ":SEARch:SERial:A429:PATTern:SSM" on page 866

:SEARCh:SERial:A429:PATTern:SDI

N (see page 1232)

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 1193

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":SBUS<n>:A429:FORMAT" on page 669
 • ":SEARCh:SERial:A429:MODE" on page 863
 • ":SEARCh:SERial:A429:PATTern:DATA" on page 864
 • ":SEARCh:SERial:A429:PATTern:SSM" on page 866

:SEARch:SERial:A429:PATTern:SSM

N (see [page 1232](#))

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 1193](#)

See Also • ["Introduction to :TRIGger Commands" on page 959](#)

• [":SBUS<n>:A429:FORMat" on page 669](#)

• [":SEARch:SERial:A429:MODE" on page 863](#)

• [":SEARch:SERial:A429:PATTern:DATA" on page 864](#)

• [":SEARch:SERial:A429:PATTern:SDI" on page 865](#)

:SEARCh:SERial:CAN Commands

Table 120 :SEARCh:SERial:CAN Commands Summary

Command	Query	Options and Query Returns
:SEARCh:SERial:CAN:MO DE <value> (see page 868)	:SEARCh:SERial:CAN:MO DE? (see page 868)	<value> ::= {DATA IDData IDEither IDRmote ALLerrors OVERload ERRor MESSage MSignal}
:SEARCh:SERial:CAN:PA TTern:DATA <string> (see page 870)	:SEARCh:SERial:CAN:PA TTern:DATA? (see page 870)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARCh:SERial:CAN:PA TTern:DATA:LENGTH <length> (see page 871)	:SEARCh:SERial:CAN:PA TTern:DATA:LENGTH? (see page 871)	<length> ::= integer from 1 to 8 in NR1 format
:SEARCh:SERial:CAN:PA TTern:ID <string> (see page 872)	:SEARCh:SERial:CAN:PA TTern:ID? (see page 872)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X} for hexadecimal
:SEARCh:SERial:CAN:PA TTern:ID:MODE <value> (see page 873)	:SEARCh:SERial:CAN:PA TTern:ID:MODE? (see page 873)	<value> ::= {STANDARD EXTENDED}
:SEARCh:SERial:CAN:SY MBolic:MESSage <name> (see page 874)	:SEARCh:SERial:CAN:SY MBolic:MESSAge? (see page 874)	<name> ::= quoted ASCII string
:SEARCh:SERial:CAN:SY MBolic:SIGNal <name> (see page 875)	:SEARCh:SERial:CAN:SY MBolic:SIGNal? (see page 875)	<name> ::= quoted ASCII string
:SEARCh:SERial:CAN:SY MBolic:VALue <data> (see page 876)	:SEARCh:SERial:CAN:SY MBolic:VALue? (see page 876)	<data> ::= value in NR3 format

:SEARCh:SERial:CAN:MODE

N (see page 1232)

Command Syntax :SEARCh:SERial:CAN:MODE <value>

```
<value> ::= {DATA | IDD | IDE | IDR | ALL | OVER | ERR | MESS | MSIG}
```

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).
- IDD - searches for CAN frames matching the specified ID of a Data frame.
- IDE - searches for the specified ID, regardless if it is a Remote frame or a Data frame.
- IDR - searches for CAN frames matching the specified ID of a Remote frame.
- ALL - searches for CAN active error frames and unknown bus conditions.
- OVER - searches for CAN overload frames.
- ERR - searches for CAN Error frame.
- MESS - when symbolic data is loaded into the oscilloscope (see "[:RECall:DBC\[:STARt\]](#)" on page 626), you can search for a symbolic message. To specify the message, use "[":SEARCh:SERial:CAN:SYMBolic:MESSAge](#)" on page 874.
- MSIG - when symbolic data is loaded into the oscilloscope, you can search for a symbolic message and a signal value. To specify the signal and signal value, use "[":SEARCh:SERial:CAN:SYMBolic:SIGNAl](#)" on page 875 and "[":SEARCh:SERial:CAN:SYMBolic:VALue](#)" on page 876.

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 | MESS | MSIG}
```

See Also

- [Chapter 29, ".SEARCh Commands,"](#) starting on page 835
- "[":SEARCh:SERial:CAN:PATTern:DATA](#)" on page 870
- "[":SEARCh:SERial:CAN:PATTern:ID](#)" on page 872
- "[":RECall:DBC\[:STARt\]](#)" on page 626
- "[":SEARCh:SERial:CAN:SYMBolic:MESSAge](#)" on page 874

- "[:SEARch:SERial:CAN:SYMBolic:SIGNal](#)" on page 875
- "[:SEARch:SERial:CAN:SYMBolic:VALUe](#)" on page 876

:SEARCh:SERial:CAN:PATTERn:DATA

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
- [":SEARCh:SERial:CAN:MODE" on page 868](#)
- [":SEARCh:SERial:CAN:PATTERn:DATA:LENGth" on page 871](#)

:SEARch:SERial:CAN:PATTern:DATA:LENGth

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:SERial:CAN:MODE” on page 868
- “:SEARch:SERial:CAN:PATTern:DATA” on page 870

:SEARch:SERial:CAN:PATTern:ID

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
- [":SEARch:SERial:CAN:MODE" on page 868](#)
- [":SEARch:SERial:CAN:PATTern:ID:MODE" on page 873](#)

:SEARch:SERial:CAN:PATTern:ID:MODE

N (see page 1232)

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 29](#), “:SEARch Commands,” starting on page 835
 • [":SEARch:SERial:CAN:MODE"](#) on page 868
 • [":SEARch:SERial:CAN:PATTern:ID"](#) on page 872

:SEARch:SERial:CAN:SYMBolic:MESSage**N** (see page 1232)

Command Syntax :SEARch:SERial:CAN:SYMBolic:MESSage <name>
 <name> ::= quoted ASCII string

The :SEARch:SERial:CAN:SYMBolic:MESSAge command specifies the message to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MESSage or MSIGnal.

Query Syntax :SEARch:SERial:CAN:SYMBolic:MESSAge?

The :SEARch:SERial:CAN:SYMBolic:MESSAge? query returns the specified message.

Return Format <name><NL>
 <name> ::= quotes ASCII string

See Also • "[:RECall:DBC\[:STARt\]](#)" on page 626
 • "[:SEARch:SERial:CAN:MODE](#)" on page 868
 • "[:SEARch:SERial:CAN:SYMBolic:SIGNAl](#)" on page 875
 • "[:SEARch:SERial:CAN:SYMBolic:VALue](#)" on page 876

:SEARCh:SERial:CAN:SYMBolic:SIGNal

N (see page 1232)

Command Syntax :SEARCh:SERial:CAN:SYMBolic:SIGNal <name>
 <name> ::= quoted ASCII string

The :SEARCh:SERial:CAN:SYMBolic:SIGNal command specifies the signal to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnal.

Query Syntax :SEARCh:SERial:CAN:SYMBolic:SIGNal?

The :SEARCh:SERial:CAN:SYMBolic:SIGNal? query returns the specified signal.

Return Format <name><NL>
 <name> ::= quoted ASCII string

See Also

- "[:RECall:DBC\[:STARt\]](#)" on page 626
- "[:SEARCh:SERial:CAN:MODE](#)" on page 868
- "[:SEARCh:SERial:CAN:SYMBolic:MESSAge](#)" on page 874
- "[:SEARCh:SERial:CAN:SYMBolic:VALue](#)" on page 876

:SEARCh:SERial:CAN:SYMBolic:VALue

N (see page 1232)

Command Syntax `:SEARCh:SERial:CAN:SYMBolic:VALue <data>`
`<data> ::= value in NR3 format`

The :SEARCh:SERial:CAN:SYMBolic:VALue command specifies the signal value to search for when CAN symbolic data has been loaded (recalled) into the oscilloscope and the CAN serial search mode is set to MSIGnAl.

NOTE

Encoded signal values are not supported in the remote interface (even though they can be used in the front panel graphical interface).

Query Syntax `:SEARCh:SERial:CAN:SYMBolic:VALue?`

The :SEARCh:SERial:CAN:SYMBolic:VALue? query returns the specified signal value.

Return Format `<data><NL>`
`<data> ::= value in NR3 format`

- See Also**
- "[:RECall:DBC\[:STARt\]](#)" on page 626
 - "[:SEARCh:SERial:CAN:MODE](#)" on page 868
 - "[:SEARCh:SERial:CAN:SYMBolic:MESSAge](#)" on page 874
 - "[:SEARCh:SERial:CAN:SYMBolic:SIGNAl](#)" on page 875

:SEARCh:SERial:FLEXray Commands

Table 121 :SEARCh:SERial:FLEXray Commands Summary

Command	Query	Options and Query Returns
:SEARCh:SERial:FLEXray:CYCLE <cycle> (see page 878)	:SEARCh:SERial:FLEXray:CYCLE? (see page 878)	<cycle> ::= {ALL <cycle #>} <cycle #> ::= integer from 0-63
:SEARCh:SERial:FLEXray:DATA <string> (see page 879)	:SEARCh:SERial:FLEXray:DATA? (see page 879)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARCh:SERial:FLEXray:DATA:LENGTH <length> (see page 880)	:SEARCh:SERial:FLEXray:DATA:LENGTH? (see page 880)	<length> ::= integer from 1 to 12 in NR1 format
:SEARCh:SERial:FLEXray:FRAMe <frame id> (see page 881)	:SEARCh:SERial:FLEXray:FRAMe? (see page 881)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:SEARCh:SERial:FLEXray:MODE <value> (see page 882)	:SEARCh:SERial:FLEXray:MODE? (see page 882)	<value> ::= {FRAMe CYCLE DATA HERRor FERRor AERRor}

:SEARch:SERial:FLEXray:CYCLe**N** (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
 • [“:SEARch:SERial:FLEXray:MODE” on page 882](#)

:SEARch:SERial:FLEXray:DATA

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:SERial:FLEXray:MODE” on page 882
- “:SEARch:SERial:FLEXray:DATA:LENGth” on page 880

:SEARch:SERial:FLEXray:DATA:LENGTH

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
 • [":SEARch:SERial:FLEXray:MODE" on page 882](#)
 • [":SEARch:SERial:FLEXray:DATA" on page 879](#)

:SEARch:SERial:FLEXray:FRAMe

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
• [“:SEARch:SERial:FLEXray:MODE” on page 882](#)

:SEARch:SERial:FLEXray:MODE

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
- [":SEARch:SERial:FLEXray:FRAMe" on page 881](#)
- [":SEARch:SERial:FLEXray:CYCLe" on page 878](#)
- [":SEARch:SERial:FLEXray:DATA" on page 879](#)

:SEARCh:SERial:I2S Commands

Table 122 :SEARCh:SERial:I2S Commands Summary

Command	Query	Options and Query Returns
:SEARCh:SERial:I2S:AU Dio <audio_ch> (see page 884)	:SEARCh:SERial:I2S:AU Dio? (see page 884)	<audio_ch> ::= {RIGHT LEFT EITHer}
:SEARCh:SERial:I2S:MO DE <value> (see page 885)	:SEARCh:SERial:I2S:MO DE? (see page 885)	<value> ::= {EQUAL NOTequal LESSthan GREaterthan INRange OUTRange}
:SEARCh:SERial:I2S:PA TTern:DATA <string> (see page 886)	:SEARCh:SERial:I2S:PA TTern:DATA? (see page 886)	<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:PA TTern:FORMAT <base> (see page 887)	:SEARCh:SERial:I2S:PA TTern:FORMAT? (see page 887)	<base> ::= {BINary HEX DECimal}
:SEARCh:SERial:I2S:RA NGe <lower>, <upper> (see page 888)	:SEARCh:SERial:I2S:RA NGe? (see page 888)	<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 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
• [":SEARch:SERial:I2S:MODE"](#) on page 885

:SEARCh:SERial:I2S:MODE

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
 - “:SEARCh:SERial:I2S:PATTERn:DATA” on page 886
 - “:SEARCh:SERial:I2S:RANGE” on page 888
 - “:SEARCh:SERial:I2S:AUDIO” on page 884

:SEARch:SERial:I2S:PATTern:DATA

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:SERial:I2S:MODE” on page 885
- “:SEARch:SERial:I2S:PATTern:FORMat” on page 887

:SEARCh:SERial:I2S:PATTern:FORMAT

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
• [":SEARCh:SERial:I2S:PATTern:DATA"](#) on page 886

:SEARch:SERial:I2S:RANGE

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
- [“:SEARch:SERial:I2S:MODE” on page 885](#)

:SEARCh:SERial:IIC Commands

Table 123 :SEARCh:SERial:IIC Commands Summary

Command	Query	Options and Query Returns
:SEARCh:SERial:IIC:MO DE <value> (see page 890)	:SEARCh:SERial:IIC:MO DE? (see page 890)	<value> ::= { READ7 WRITE7 NACKnowledge ANACK R7Data2 W7Data2 RESTart READEprom}
:SEARCh:SERial:IIC:PA TTern:ADDRESS <value> (see page 892)	:SEARCh:SERial:IIC:PA TTern:ADDRess? (see page 892)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARCh:SERial:IIC:PA TTern:DATA <value> (see page 893)	:SEARCh:SERial:IIC:PA TTern:DATA? (see page 893)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARCh:SERial:IIC:PA TTern:DATA2 <value> (see page 894)	:SEARCh:SERial:IIC:PA TTern:DATA2? (see page 894)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:SEARCh:SERial:IIC:QU ALifier <value> (see page 895)	:SEARCh:SERial:IIC:QU ALifier? (see page 895)	<value> ::= { EQUAL NOTequal LESSthan GREaterthan}

:SEARCh:SERial:IIC:MODE

N (see [page 1232](#))

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 1234](#)).

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 29, “:SEARCh Commands,” starting on page 835](#)

- "[:SEARch:SERial:IIC:PATTern:ADDResS](#)" on page 892
- "[:SEARch:SERial:IIC:PATTern:DATA](#)" on page 893
- "[:SEARch:SERial:IIC:PATTern:DATA2](#)" on page 894
- "[:SEARch:SERial:IIC:QUALifier](#)" on page 895

:SEARCh:SERial:IIC:PATTERn:ADDResS

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
 • “:SEARCh:SERial:IIC:MODE” on page 890

:SEARCh:SERial:IIC:PATTERn:DATA

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 • [":SEARCh:SERial:IIC:MODE" on page 890](#)
 • [":SEARCh:SERial:IIC:QUALifier" on page 895](#)
 • [":SEARCh:SERial:IIC:PATTERn:DATA2" on page 894](#)

:SEARch:SERial:IIC:PATTERn:DATA2

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
 • [":SEARch:SERial:IIC:MODE" on page 890](#)
 • [":SEARch:SERial:IIC:PATTERn:DATA" on page 893](#)

:SEARCh:SERial:IIC:QUALifier

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:SERial:IIC:MODE" on page 890](#)
 - [":SEARCh:SERial:IIC:PATTern:DATA" on page 893](#)

:SEARch:SERial:LIN Commands

Table 124 :SEARch:SERial:LIN Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:LIN:ID <value> (see page 897)	:SEARch:SERial:LIN:ID ? (see page 897)	<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:MO DE <value> (see page 898)	:SEARch:SERial:LIN:MO DE? (see page 898)	<value> ::= {ID DATA ERRor}
:SEARch:SERial:LIN: TTern:DATA <string> (see page 899)	:SEARch:SERial:LIN: TTern:DATA? (see page 899)	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: TTern:DATA:LENGTH <length> (see page 900)	:SEARch:SERial:LIN: TTern:DATA:LENGTH? (see page 900)	<length> ::= integer from 1 to 8 in NR1 format
:SEARch:SERial:LIN: TTern:FORMAT <base> (see page 901)	:SEARch:SERial:LIN: TTern:FORMAT? (see page 901)	<base> ::= {HEX DECimal}

:SEARCh:SERial:LIN:ID

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:SERial:LIN:MODE"](#) on page 898

:SEARCh:SERial:LIN:MODE

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 • [":SEARCh:SERial:LIN:ID" on page 897](#)
 • [":SEARCh:SERial:LIN:PATTERn:DATA" on page 899](#)

:SEARCh:SERial:LIN:PATTERn:DATA

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
- [":SEARCh:SERial:LIN:MODE" on page 898](#)
- [":SEARCh:SERial:LIN:PATTERn:FORMAT" on page 901](#)
- [":SEARCh:SERial:LIN:PATTERn:DATA:LENGth" on page 900](#)

:SEARch:SERial:LIN:PATTERn:DATA:LENGth

N (see page 1232)

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 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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:SERial:LIN:PATTERn:DATA” on page 899

:SEARCh:SERial:LIN:PATTERn:FORMat

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
• [“:SEARCh:SERial:LIN:PATTERn:DATA” on page 899](#)

:SEARch:SERial:M1553 Commands

Table 125 :SEARch:SERial:M1553 Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:M1553: MODE <value> (see page 903)	:SEARch:SERial:M1553: MODE? (see page 903)	<value> ::= {DSTARt CSTArt RTA RTA11 PERRor SERRor MERRor}
:SEARch:SERial:M1553: PATTern:DATA <string> (see page 904)	:SEARch:SERial:M1553: PATTern:DATA? (see page 904)	<string> ::= "nn...n" where n ::= {0 1}
:SEARch:SERial:M1553: RTA <value> (see page 905)	:SEARch:SERial:M1553: RTA? (see page 905)	<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 1232)

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 29, “:SEARCh Commands,” starting on page 835
- “:SEARCh:SERial:M1553:RTA” on page 905
- “:SEARCh:SERial:M1553:PATTern:DATA” on page 904

:SEARch:SERial:M1553:PATTERn:DATA

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835](#)
- [":SEARch:SERial:M1553:MODE" on page 903](#)

:SEARCh:SERial:M1553:RTA

N (see page 1232)

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 29](#), “:SEARCh Commands,” starting on page 835

:SEARch:SERial:SPI Commands

Table 126 :SEARch:SERial:SPI Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:SPI:MO DE <value> (see page 907)	:SEARch:SERial:SPI:MO DE? (see page 907)	<value> ::= {MOSI MISO}
:SEARch:SERial:SPI:PA TTern:DATA <string> (see page 908)	:SEARch:SERial:SPI:PA TTern:DATA? (see page 908)	<string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F X}
:SEARch:SERial:SPI:PA TTern:WIDTH <width> (see page 909)	:SEARch:SERial:SPI:PA TTern:WIDTH? (see page 909)	<width> ::= integer from 1 to 10

:SEARCh:SERial:SPI:MODE

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
 • “:SEARCh:SERial:SPI:PATTERn:DATA” on page 908

:SEARch:SERial:SPI:PATTERn:DATA

N (see page 1232)

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 29, “:SEARch Commands,” starting on page 835
- “:SEARch:SERial:SPI:PATTERn:WIDTH” on page 909

:SEARCh:SERial:SPI:PATTERn:WIDTh

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
• “:SEARCh:SERial:SPI:PATTERn:DATA” on page 908

:SEARch:SERial:UART Commands

Table 127 :SEARch:SERial:UART Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:UART:D ATA <value> (see page 911)	:SEARch:SERial:UART:D ATA? (see page 911)	<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:M ODE <value> (see page 912)	:SEARch:SERial:UART:M ODE? (see page 912)	<value> ::= {RDATa RD1 RD0 RDX TDATA TD1 TD0 TDX PARityerror AERRor}
:SEARch:SERial:UART:Q UALifier <value> (see page 913)	:SEARch:SERial:UART:Q UALifier? (see page 913)	<value> ::= {EQUal NOTequal GREaterthan LESSthan}

:SEARCh:SERial:UART:DATA

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835
- “:SEARCh:SERial:UART:MODE” on page 912
- “:SEARCh:SERial:UART:QUALifier” on page 913

:SEARCh:SERial:UART:MODE

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
 - [":SEARCh:SERial:UART:DATA" on page 911](#)
 - [":SEARCh:SERial:UART:QUALifier" on page 913](#)

:SEARCh:SERial:UART:QUALifier

N (see page 1232)

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 29, “:SEARCh Commands,” starting on page 835](#)
• [":SEARCh:SERial:UART:DATA" on page 911](#)

:SEARch:SERial:USB Commands

Table 128 :SEARch:SERial:USB Commands Summary

Command	Query	Options and Query Returns
:SEARch:SERial:USB:MO DE <condition> (see page 916)	:SEARch:SERial:USB:MO DE? (see page 916)	<event> ::= {TOKen DATA HANDshake SPECial ALLerrors PIDerror CRC5error CRC16error GLITCHerror STUFFerror SE1error}
:SEARch:SERial:USB:AD DRess <string> (see page 917)	:SEARch:SERial:USB:AD DRess? (see page 917)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:CR C <string> (see page 918)	:SEARch:SERial:USB:CR C? (see page 918)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:DA TA <string> (see page 919)	:SEARch:SERial:USB:DA TA? (see page 919)	<string> ::= "0xn..." where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:DA TA:LENGTH <value> (see page 920)	:SEARch:SERial:USB:DA TA:LENGTH? (see page 920)	<length> ::= data length between 1-20
:SEARch:SERial:USB:EN DPoint <string> (see page 921)	:SEARch:SERial:USB:EN DPoint? (see page 921)	<string> ::= "0xn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:ET <string> (see page 922)	:SEARch:SERial:USB:ET ? (see page 922)	<string> ::= "0xn" where n ::= {0 1 2 3 X \$}
:SEARch:SERial:USB:FR AMe <string> (see page 923)	:SEARch:SERial:USB:FR AMe? (see page 923)	<string> ::= "0xnnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:HA DDress <string> (see page 924)	:SEARch:SERial:USB:HA DDress? (see page 924)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARch:SERial:USB:PI D:DATA <pid> (see page 925)	:SEARch:SERial:USB:PI D:DATA? (see page 925)	<pid> ::= {DATA0 DATA1 DATA2 MDATA}
:SEARch:SERial:USB:PI D:HANDshake <pid> (see page 926)	:SEARch:SERial:USB:PI D:HANDshake? (see page 926)	<pid> ::= {ACK NAK STALL NYET}
:SEARch:SERial:USB:PI D:SPECial <pid> (see page 927)	:SEARch:SERial:USB:PI D:SPECial? (see page 927)	<pid> ::= {PING PRE ERR SPLIt}

Table 128 :SEARCh:SERial:USB Commands Summary (continued)

Command	Query	Options and Query Returns
:SEARCh:SERial:USB:PI D:TOKen <pid> (see page 928)	:SEARCh:SERial:USB:PI D:TOKen? (see page 928)	<pid> ::= {OUT IN SETup SOF}
:SEARCh:SERial:USB:PO RT <string> (see page 929)	:SEARCh:SERial:USB:PO RT? (see page 929)	<string> ::= "0xnn" where n ::= {0,...,9 A,...,F X \$}
:SEARCh:SERial:USB:SC <string> (see page 930)	:SEARCh:SERial:USB:SC ? (see page 930)	<string> ::= "0xn" where n ::= {0 1 X \$}
:SEARCh:SERial:USB:SE U <string> (see page 931)	:SEARCh:SERial:USB:SE U? (see page 931)	<string> ::= "0xn" where n ::= {0 1 2 3 X \$}

:SEARCh:SERial:USB:MODE

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:MODE <mode>

```
<mode> ::= {TOKen | DATA | HANDshake | SPECial | ALLerrors
             | PIDerror | CRC5error | CRC16error | GLITcherror
             | STUFFerror | SE1error}
```

The :SEARCh:SERial:USB:MODE command specifies the USB search mode.

Query Syntax :SEARCh:SERial:USB:MODE?

The :SEARCh:SERial:USB:MODE? query returns the specified USB search mode.

Return Format <mode><NL>

```
<mode> ::= {TOK | DATA | HAND | SPEC | ALL | PID | CRC5
             | CRC16 | GLIT | STUFF | SE1}
```

- See Also**
- "[:SEARCh:SERial:USB:PID:DATA](#)" on page 925
 - "[:SEARCh:SERial:USB:PID:HANDshake](#)" on page 926
 - "[:SEARCh:SERial:USB:PID:SPECial](#)" on page 927
 - "[:SEARCh:SERial:USB:PID:TOKen](#)" on page 928

:SEARCh:SERial:USB:ADDResS

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:ADDResS <string>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

The :SEARCh:SERial:USB:ADDResS command specifies the 7-bit Address portion of the search value, in hex.

Query Syntax :SEARCh:SERial:USB:ADDResS?

The :SEARCh:SERial:USB:ADDResS? query returns the specified Address portion of the search value.

Return Format <string><NL>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

See Also • "[:SEARCh:SERial:USB:PID:SPEcial](#)" on page 927

• "[:SEARCh:SERial:USB:PID:TOken](#)" on page 928

:SEARch:SERial:USB:CRC**N** (see [page 1232](#))**Command Syntax** `:SEARch:SERial:USB:CRC <string>``<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | $}`

The :SEARch:SERial:USB:CRC command specifies the CRC portion of the search value, in hex.

Query Syntax `:SEARch:SERial:USB:CRC?`

The :SEARch:SERial:USB:CRC? query returns the specified CRC portion of the search value.

Return Format `<string><NL>``<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | $}`**See Also**

- "[:SEARch:SERial:USB:PID:SPECial](#)" on page 927
- "[:SEARch:SERial:USB:PID:TOken](#)" on page 928

:SEARCh:SERial:USB:DATA

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:DATA <string>

<string> ::= "0xn..." where n ::= {0,...,9 | A,...,F | X | \$}

The :SEARCh:SERial:USB:DATA command specifies the Data portion of the search value, in hex.

See the :SEARCh:SERial:USB:DATA:LENGth command for setting the length of the data value.

Query Syntax :SEARCh:SERial:USB:DATA?

The :SEARCh:SERial:USB:DATA? query returns the specified Data portion of the search value.

Return Format <string><NL>

<string> ::= "0xn..." where n ::= {0,...,9 | A,...,F | X | \$}

See Also • "[:SEARCh:SERial:USB:DATA:LENGth](#)" on page 920

:SEARch:SERial:USB:DATA:LENGTH

N (see page 1232)

Command Syntax `:SEARch:SERial:USB:DATA:LENGTH <length>`
 `<length> ::= data length between 1-20`

The :SEARch:SERial:USB:DATA:LENGTH command specifies the data length in bytes.

Query Syntax `:SEARch:SERial:USB:DATA:LENGTH?`
The :SEARch:SERial:USB:DATA:LENGTH? query returns the specified data length.

Return Format `<length><NL>`
 `<length> ::= data length between 1-20`

See Also • "":SEARch:SERial:USB:DATA" on page 919

:SEARCh:SERial:USB:ENDPoint

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:ENDPoint <string>

<string> ::= "0xn" where n ::= {0,...,9 | A,...,F | x | \$}

The :SEARCh:SERial:USB:ENDPoint command specifies the 4-bit Endpoint portion of the search value, in hex.

Query Syntax :SEARCh:SERial:USB:ENDPoint?

The :SEARCh:SERial:USB:ENDPoint? query returns the specified Endpoint portion of the search value.

Return Format <string><NL>

<string> ::= "0xn" where n ::= {0,...,9 | A,...,F | x | \$}

See Also • "[:SEARCh:SERial:USB:PID:SPECial](#)" on page 927
 • "[:SEARCh:SERial:USB:PID:TOken](#)" on page 928

:SEARch:SERial:USB:ET

N (see [page 1232](#))

Command Syntax :SEARch:SERial:USB:ET <string>

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | X | \$}

The :SEARch:SERial:USB:ET command specifies the 2-bit ET portion of the search value, in hex.

Query Syntax :SEARch:SERial:USB:ET?

The :SEARch:SERial:USB:ET? query returns the specified ET portion of the search value.

Return Format <string><NL>

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | X | \$}

See Also • "[:SEARch:SERial:USB:PID:SPECial](#)" on page 927

:SEARCh:SERial:USB:FRAMe**N** (see page 1232)**Command Syntax** :SEARCh:SERial:USB:FRAMe <string>

<string> ::= "0xnnn" where n ::= {0,...,9 | A,...,F | X | \$}

The :SEARCh:SERial:USB:FRAMe command specifies the 11-bit Frame portion of the search value, in hex.

Query Syntax :SEARCh:SERial:USB:FRAMe?

The :SEARCh:SERial:USB:FRAMe? query returns the specified Frame portion of the search value.

Return Format <string><NL>

<string> ::= "0xnnn" where n ::= {0,...,9 | A,...,F | X | \$}

See Also • "[:SEARCh:SERial:USB:PID:TOKen](#)" on page 928

:SEARch:SERial:USB:HADDress**N** (see page 1232)**Command Syntax** :SEARch:SERial:USB:HADDress <string>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

The :SEARch:SERial:USB:HADDress command specifies the 7-bit Hub Address portion of the search value, in hex.

Query Syntax :SEARch:SERial:USB:HADDress?

The :SEARch:SERial:USB:HADDress? query returns the specified Hub Address portion of the search value.

Return Format <string><NL>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

See Also • "[:SEARch:SERial:USB:PID:SPECial](#)" on page 927

:SEARCh:SERial:USB:PID:DATA

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:PID:DATA <pid>

<pid> ::= {DATA0 | DATA1 | DATA2 | MDATA}

The :SEARCh:SERial:USB:PID:DATA command specifies the USB data PID to find.

The DATA USB search mode can be selected using the :SEARCh:SERial:USB:MODE command.

Query Syntax :SEARCh:SERial:USB:PID:DATA?

The :SEARCh:SERial:USB:PID:DATA? query returns the specified data PID.

Return Format <pid><NL>

<pid> ::= {DATA0 | DATA1 | DATA2 | MDAT}

See Also • "[:SEARCh:SERial:USB:MODE](#)" on page 916

:SEARch:SERial:USB:PID:HANDshake**N** (see page 1232)**Command Syntax** :SEARch:SERial:USB:PID:HANDshake <pid>
 <pid> ::= {ACK | NAK | STALL | NYET}

The :SEARch:SERial:USB:PID:HANDshake command specifies the USB handshake PID to find.

The HANDshake USB search mode can be selected using the :SEARch:SERial:USB:MODE command.

Query Syntax :SEARch:SERial:USB:PID:HANDshake?

The :SEARch:SERial:USB:PID:HANDshake? query returns the specified handshake PID.

Return Format <pid><NL>
 <pid> ::= {ACK | NAK | STAL | NYET}**See Also** • "[:SEARch:SERial:USB:MODE](#)" on page 916

:SEARCh:SERial:USB:PID:SPECial

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:PID:SPECial <pid>
<pid> ::= {PING | PRE | ERR | SPL}

The :SEARCh:SERial:USB:PID:SPECial command specifies the USB special PID to find.

The SPECial USB search mode can be selected using the :SEARCh:SERial:USB:MODE command.

Query Syntax :SEARCh:SERial:USB:PID:SPECial?

The :SEARCh:SERial:USB:PID:SPECial? query returns the specified special PID.

Return Format <pid><NL>
<pid> ::= {PING | PRE | ERR | SPL}

See Also • "[:SEARCh:SERial:USB:MODE](#)" on page 916

:SEARch:SERial:USB:PID:TOKen

N (see [page 1232](#))

Command Syntax `:SEARch:SERial:USB:PID:TOKen <pid>`
`<pid> ::= {OUT | IN | SETup | SOF}`

The :SEARch:SERial:USB:PID:TOKen command specifies the USB token PID to find.

The TOKen USB search mode can be selected using the :SEARch:SERial:USB:MODE command.

Query Syntax `:SEARch:SERial:USB:PID:TOKen?`

The :SEARch:SERial:USB:PID:TOKen? query returns the specified token PID.

Return Format `<pid><NL>`
`<pid> ::= {OUT | IN | SETup | SOF}`

See Also • "[:SEARch:SERial:USB:MODE](#)" on page 916

:SEARCh:SERial:USB:PORT

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:PORT <string>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

The :SEARCh:SERial:USB:PORT command specifies the 7-bit Port portion of the search value, in hex.

Query Syntax :SEARCh:SERial:USB:PORT?

The :SEARCh:SERial:USB:PORT? query returns the specified Port portion of the search value.

Return Format <string><NL>

<string> ::= "0xnn" where n ::= {0,...,9 | A,...,F | X | \$}

See Also • "[:SEARCh:SERial:USB:PID:SPECial](#)" on page 927

:SEARch:SERial:USB:SC

N (see page 1232)

Command :SEARch:SERial:USB:SC <string>

SyntaxS <string> ::= "0xn" where n ::= {0 | 1 | X | \$}

The :SEARch:SERial:USB:SC command specifies the 1-bit SC portion of the search value, in hex.

Query Syntax :SEARch:SERial:USB:SC?

The :SEARch:SERial:USB:SC? query returns the specified SC portion of the search value.

Return Format <string><NL>

<string> ::= "0xn" where n ::= {0 | 1 | X | \$}

See Also • "[:SEARch:SERial:USB:PID:SPECial](#)" on page 927

:SEARCh:SERial:USB:SEU

N (see page 1232)

Command Syntax :SEARCh:SERial:USB:SEU <string>

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | X | \$}

The :SEARCh:SERial:USB:SEU command specifies the 2-bit S and E or U portion of the search value, in hex.

Query Syntax :SEARCh:SERial:USB:SEU?

The :SEARCh:SERial:USB:SEU? query returns the specified S and E or U portion of the search value.

Return Format <string><NL>

<string> ::= "0xn" where n ::= {0 | 1 | 2 | 3 | X | \$}

See Also • "[:SEARCh:SERial:USB:PID:SPECial](#)" on page 927

30 :SYSTem Commands

Control basic system functions of the oscilloscope. See "[Introduction to :SYSTem Commands](#)" on page 934.

Table 129 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see page 935)	:SYSTem:DATE? (see page 935)	<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 936)	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 937)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 1191).
:SYSTem:LOCK <value> (see page 938)	:SYSTem:LOCK? (see page 938)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:PRESet (see page 939)	n/a	See :SYSTem:PRESet (see page 939)
:SYSTem:PROtection:LOCK <value> (see page 942)	:SYSTem:PROtection:LOCK? (see page 942)	<value> ::= {{1 ON} {0 OFF}}
:SYSTem:SETup <setup_data> (see page 943)	:SYSTem:SETup? (see page 943)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 945)	:SYSTem:TIME? (see page 945)	<time> ::= hours,minutes,seconds in NR1 format
:SYSTem:TOUCH {{1 ON} {0 OFF}} (see page 946)	:SYSTem:TOUCH? (see page 946)	{1 0}



Introduction to :SYSTem Commands SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem:DATE

N (see page 1232)

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>

- "Introduction to :SYSTem Commands" on page 934
- ":SYSTem:TIME" on page 945

:SYSTem:DSP

N (see [page 1232](#))

Command Syntax :SYSTem:DSP <string>

<string> ::= quoted ASCII string (up to 75 characters)

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box on-screen.

Use :SYSTem:DSP "" to remotely remove the message from the display.
(Two sets of quote marks without a space between them creates a NULL string.)

Press any menu key to manually remove the message from the display.

See Also • "Introduction to :SYSTem Commands" on page 934

:SYSTem:ERRor

(see page 1232)

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 37, “Error Messages,”](#) starting on page 1191.

See Also

- "[Introduction to :SYSTem Commands](#)" on page 934
- "[*ESR \(Standard Event Status Register\)](#)" on page 172
- "[*CLS \(Clear Status\)](#)" on page 169

:SYSTem:LOCK

N (see page 1232)

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 934

:SYSTem:PRESet

C (see page 1232)

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 (1.4 V)

Display Menu	
Persistence	Off
Grid	20%

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	40 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also**
- "[Introduction to Common \(*\) Commands](#)" on page 168
 - "["*RST \(Reset\)](#)" on page 180

:SYSTem:PROTection:LOCK

N (see page 1232)

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 934

:SYSTem:SEtup

C (see page 1232)

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 934
 - "[*LRN \(Learn Device Setup\)](#)" on page 175

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.
```

30 :SYSTem Commands

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"  
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

See complete example programs at: [Chapter 41, “Programming Examples,”](#) starting on page 1241

:SYSTem:TIME

N (see page 1232)

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 934
 - ":SYSTem:DATE" on page 935

:SYSTem:TOUCH

N (see [page 1232](#))

Command Syntax `:SYSTem:TOUCH <on_off>`

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

The `:SYSTem:TOUCH` command disables or enables the touchscreen.

Query Syntax `:SYSTem:TOUCH?`

The `:SYSTem:TOUCH?` query returns the touchscreen's on/off status.

Return Format `<on_off><NL>`

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

See Also • "Introduction to `:SYSTem Commands`" on [page 934](#)

31 :TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 948.

Table 130 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 949)	:TIMebase:MODE? (see page 949)	<value> ::= {MAIN WINDOW XY ROLL}
:TIMebase:POSITION <pos> (see page 950)	:TIMebase:POSITION? (see page 950)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGE <range_value> (see page 951)	:TIMebase:RANGE? (see page 951)	<range_value> ::= time for 10 div in seconds in NR3 format
:TIMebase:REFClock { {0 OFF} {1 ON} } (see page 952)	:TIMebase:REFClock? (see page 952)	{0 1}
:TIMebase:REFERENCE {LEFT CENTER RIGHT} (see page 953)	:TIMebase:REFERENCE? (see page 953)	<return_value> ::= {LEFT CENTER RIGHT}
:TIMebase:SCALE <scale_value> (see page 954)	:TIMebase:SCALE? (see page 954)	<scale_value> ::= time/div in seconds in NR3 format
:TIMebase:VERNier {{0 OFF} {1 ON}} (see page 955)	:TIMebase:VERNier? (see page 955)	{0 1}
:TIMebase:WINDOW:POSITION <pos> (see page 956)	:TIMebase:WINDOW:POSITION? (see page 956)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format



Table 130 :TIMEbase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMEbase:WINDOW:RANGE <range_value> (see page 957)	:TIMEbase:WINDOW:RANGE? (see page 957)	<range_value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALE <scale_value> (see page 958)	:TIMEbase:WINDOW:SCALE? (see page 958)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

Introduction to :TIMEbase Commands The TIMEbase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (zoomed) time bases.

Reporting the Setup

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

Return Format

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

```
:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

:TIMEbase:MODE

(see page 1232)

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:POSIon, 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 948
- "*RST (Reset)" on page 180
- ":TIMEbase:RANGE" on page 951
- ":TIMEbase:POSIon" on page 950
- ":TIMEbase:REFerence" on page 953

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 41, “Programming Examples,”](#) starting on page 1241

:TIMEbase:POSITION

C (see page 1232)

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 948
 - "`:TIMEbase:REFERENCE`" on page 953
 - "`:TIMEbase:RANGE`" on page 951
 - "`:TIMEbase:SCALE`" on page 954
 - "`:TIMEbase:WINDOW:POSITION`" on page 956
 - "`:TIMEbase:DELay`" on page 1188

:TIMEbase:RANGE

C (see page 1232)

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

- "[Introduction to :TIMEbase Commands](#)" on page 948
- "[:TIMEbase:MODE](#)" on page 949
- "[:TIMEbase:SCALe](#)" on page 954
- "[:TIMEbase:WINDow:RANGE](#)" on page 957

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 41, “Programming Examples,”](#) starting on page 1241

:TIMEbase:REFClock

N (see [page 1232](#))

Command Syntax `:TIMEbase:REFClock <value>`

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

The :TIMEbase:REFClock command enables or disables the 10 MHz REF BNC located on the rear panel of the oscilloscope.

The 10 MHz REF BNC can be used as an input for the oscilloscope's reference clock (instead of the internal 10 MHz reference), or it can be used to output the internal 10 MHz reference clock when synchronizing multiple instruments (see "[:ACQuire:RSIGnal](#)" on page 237).

The :TIMEbase:REFClock ON command enables the 10 MHz REF BNC and sets the reference signal mode to IN. The :TIMEbase:REFClock OFF command disables the 10 MHz REF BNC (the same as setting the reference signal mode to OFF).

Query Syntax `:TIMEbase:REFClock?`

The :TIMEbase:REFClock? query returns the current state of the 10 MHz reference signal mode. A "1" indicates that the 10 MHz REF input is enabled (on), and a "0" indicates that either the 10 MHz REF BNC is disabled (off) or that it is set as an output (by the :ACQuire:RSIGnal command).

Return Format `<value><NL>`

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

See Also • "[:ACQuire:RSIGnal](#)" on page 237

:TIMEbase:REFerence

C (see page 1232)

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 948
 - ":TIMEbase:MODE" on page 949

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 right.
myScope.WriteString ":TIMEbase:REFerence CENTER"      ' Set reference to center.
```

See complete example programs at: [Chapter 41, “Programming Examples,” starting on page 1241](#)

:TlMEbase:SCALe

N (see [page 1232](#))

Command Syntax `:TlMEbase:SCALe <scale_value>`

`<scale_value> ::= time/div in seconds in NR3 format`

The `:TlMEbase:SCALe` command sets the horizontal scale or units per division for the main window.

Query Syntax `:TlMEbase:SCALe?`

The `:TlMEbase: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 :TlMEbase Commands](#)" on page 948
 - "[":TlMEbase:RANGE"](#) on page 951
 - "[":TlMEbase:WINDOW:SCALe"](#) on page 958
 - "[":TlMEbase:WINDOW:RANGE"](#) on page 957

:TIMEbase:VERNier

N (see page 1232)

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 948

:TIMEbase:WINDOW:POSITION

(see page 1232)

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 948
- "[:TIMEbase:MODE](#)" on page 949
- "[:TIMEbase:POSITION](#)" on page 950
- "[:TIMEbase:RANGE](#)" on page 951
- "[:TIMEbase:SCALE](#)" on page 954
- "[:TIMEbase:WINDOW:RANGE](#)" on page 957
- "[:TIMEbase:WINDOW:SCALE](#)" on page 958

:TIMEbase:WINDOW:RANGE

C (see page 1232)

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 948
 - ":TIMEbase:RANGE" on page 951
 - ":TIMEbase:POSITION" on page 950
 - ":TIMEbase:SCALE" on page 954

:TIMEbase:WINDOW:SCALE

N (see page 1232)

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 948

- ":TIMEbase:RANGE" on page 951
- ":TIMEbase:POSITION" on page 950
- ":TIMEbase:SCALE" on page 954
- ":TIMEbase:WINDOW:RANGE" on page 957

32

:TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "[Introduction to :TRIGger Commands](#)" on page 959
- "[General :TRIGger Commands](#)" on page 961
- "[:TRIGger:DELay Commands](#)" on page 971
- "[:TRIGger:EBURst Commands](#)" on page 978
- "[:TRIGger\[:EDGE\] Commands](#)" on page 983
- "[:TRIGger:GLITch Commands](#)" on page 990 (Pulse Width trigger)
- "[:TRIGger:OR Commands](#)" on page 999
- "[:TRIGger:PATTern Commands](#)" on page 1001
- "[:TRIGger:RUNT Commands](#)" on page 1010
- "[:TRIGger:SHOLD Commands](#)" on page 1015
- "[:TRIGger:TRANSition Commands](#)" on page 1021
- "[:TRIGger:TV Commands](#)" on page 1026
- "[:TRIGger:ZONE Commands](#)" on page 1036

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 970) 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 968).

- **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 $\frac{1}{4}$ 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 +60.000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC;
:TRIG:ZONE:STAT 0
```

General :TRIGger Commands

Table 131 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FORCe (see page 962)	n/a	n/a
:TRIGger:HFReject {{0 OFF} {1 ON}} (see page 963)	:TRIGger:HFReject? (see page 963)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 964)	:TRIGger:HOLDoff? (see page 964)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:ASETup (see page 965)	n/a	n/a
:TRIGger:LEVel:HIGH <level>, <source> (see page 966)	:TRIGger:LEVel:HIGH? <source> (see page 966)	<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 967)	:TRIGger:LEVel:LOW? <source> (see page 967)	<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 968)	:TRIGger:MODE? (see page 968)	<mode> ::= {EDGE GLITCH PATTern TV DELay EBURst OR RUNT SHOLD TRANSition SBUS{1 2}} <return_value> ::= {<mode> <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREject {{0 OFF} {1 ON}} (see page 969)	:TRIGger:NREject? (see page 969)	{0 1}
:TRIGger:SWEep <sweep> (see page 970)	:TRIGger:SWEep? (see page 970)	<sweep> ::= {AUTO NORMal}

:TRIGger:FORCe

N (see [page 1232](#))

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 959

:TRIGger:HFReject

(see page 1232)

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 959
 - ":TRIGger[:EDGE]:REject" on page 987

:TRIGger:HOLDoff**C** (see page 1232)**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 959

:TRIGger:LEVel:ASETup

N (see page 1232)

Command Syntax :TRIGger:LEVel:ASETup

The :TRIGger:LEVel:ASETup command automatically sets the trigger levels of all displayed analog channels to their waveforms' 50% values.

If AC coupling is used, the trigger levels are set to 0 V.

When High and Low (dual) trigger levels are used (as with Rise/Fall Time and Runt triggers, for example), this command has no effect.

See Also • "[:TRIGger\[:EDGE\]:LEVel](#)" on page 986

:TRIGger:LEVel:HIGH

N (see page 1232)

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 959
 - ":TRIGger:LEVel:LOW" on page 967
 - ":TRIGger:RUNT Commands" on page 1010
 - ":TRIGger:TRANsition Commands" on page 1021
 - ":TRIGger[:EDGE]:SOURce" on page 989

:TRIGger:LEVel:LOW

N (see page 1232)

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 959
 - ":TRIGger:LEVel:HIGH" on page 966
 - ":TRIGger:RUNT Commands" on page 1010
 - ":TRIGger:TRANsition Commands" on page 1021
 - ":TRIGger[:EDGE]:SOURce" on page 989

:TRIGger:MODE

(see page 1232)

Command Syntax `:TRIGger:MODE <mode>`

```
<mode> ::= {EDGE | GLITch | PATTern | TV | DELay | EBURst | OR | RUNT
             | SHOLD | TRANsition | SBUS{1 | 2}}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax `:TRIGger:MODE?`

The :TRIGger:MODE? query returns the current trigger mode. 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}}
```

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 959
 - "[:TRIGger:SWEep](#)" on page 970
 - "[:TIMEbase:MODE](#)" on page 949

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: [Chapter 41, “Programming Examples,”](#) starting on page 1241

:TRIGger:NREject**C** (see page 1232)**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 959

:TRIGger:SWEep

(see page 1232)

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 959

:TRIGger:DELay Commands

Table 132 :TRIGger:DELay Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DELay:ARM:SL OPe <slope> (see page 972)	:TRIGger:DELay:ARM:SL OPe? (see page 972)	<slope> ::= {NEGative POSitive}
:TRIGger:DELay:ARM:SOURce <source> (see page 973)	:TRIGger:DELay:ARM:SOURce? (see page 973)	<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 974)	:TRIGger:DELay:TDELay :TIME? (see page 974)	<time_value> ::= time in seconds in NR3 format
:TRIGger:DELay:TRIGger:COUNT <count> (see page 975)	:TRIGger:DELay:TRIGger:COUNT? (see page 975)	<count> ::= integer in NR1 format
:TRIGger:DELay:TRIGger:SLOPe <slope> (see page 976)	:TRIGger:DELay:TRIGger:SLOPe? (see page 976)	<slope> ::= {NEGative POSitive}
:TRIGger:DELay:TRIGger:SOURce <source> (see page 977)	:TRIGger:DELay:TRIGger:SOURce? (see page 977)	<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 1232)

Command Syntax :TRIGger:DELay:ARM:SLOPe <slope>
 <slope> ::= {NEGative | POSitive}

The :TRIGger:DELay:ARM:SLOPe command specifies rising (POSitive) or falling (NEGative) for the arming edge in the Edge Then Edge trigger.

Query Syntax :TRIGger:DELay:ARM:SLOPe?

The :TRIGger:DELay:ARM:SLOPe? query returns the current arming edge slope setting.

Return Format <slope><NL>
 <slope> ::= {NEG | POS}

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":TRIGger:DELay:ARM:SOURce" on page 973
 • ":TRIGger:DELay:TDELay:TIME" on page 974

:TRIGger:DELay:ARM:SOURce

N (see page 1232)

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 959
 - ":TRIGger:DELay:ARM:SLOPe" on page 972
 - ":TRIGger:DELay:TDELay:TIME" on page 974
 - ":TRIGger:MODE" on page 968

:TRIGger:DELay:TDELay:TIME

N (see page 1232)

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 959
- "[:TRIGger:DELay:TRIGger:SLOPe](#)" on page 976
- "[:TRIGger:DELay:TRIGger:COUNt](#)" on page 975

:TRIGger:DELay:TRIGger:COUNt

N (see page 1232)

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 959
• "[:TRIGger:DELay:TRIGger:SLOPe](#)" on page 976
• "[:TRIGger:DELay:TRIGger:SOURce](#)" on page 977
• "[:TRIGger:DELay:TDELay:TIME](#)" on page 974

:TRIGger:DELay:TRIGger:SLOPe**N**

(see page 1232)

Command Syntax :TRIGger:DELay:TRIGger:SLOPe <slope>
 <slope> ::= {NEGative | POSitive}

The :TRIGger:DELay:TRIGger:SLOPe command specifies rising (POSitive) or falling (NEGative) for the trigger edge in the Edge Then Edge trigger.

Query Syntax :TRIGger:DELay:TRIGger:SLOPe?

The :TRIGger:DELay:TRIGger:SLOPe? query returns the current trigger edge slope setting.

Return Format <slope><NL>
 <slope> ::= {NEG | POS}

See Also • "Introduction to :TRIGger Commands" on page 959
 • ":TRIGger:DELay:TRIGger:SOURce" on page 977
 • ":TRIGger:DELay:TDELay:TIME" on page 974
 • ":TRIGger:DELay:TRIGger:COUNt" on page 975

:TRIGger:DELay:TRIGger:SOURce

N (see page 1232)

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 959

- ":TRIGger:DELay:TRIGger:SLOPe" on page 976

- ":TRIGger:DELay:TDELay:TIME" on page 974

- ":TRIGger:DELay:TRIGger:COUNt" on page 975

- ":TRIGger:MODE" on page 968

:TRIGger:EBURst Commands

Table 133 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNT <count> (see page 979)	:TRIGger:EBURst:COUNT? ? (see page 979)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 980)	:TRIGger:EBURst:IDLE? ? (see page 980)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 981)	:TRIGger:EBURst:SLOPe? ? (see page 981)	<slope> ::= {NEGative POSitive}
:TRIGger:EBURst:SOURce <source> (see page 982)	:TRIGger:EBURst:SOURce? ? (see page 982)	<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 1232)

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 959
 - "[:TRIGger:EBURst:SLOPe](#)" on page 981
 - "[:TRIGger:EBURst:IDLE](#)" on page 980

:TRIGger:EBURst:IDLE

N (see page 1232)

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 959
- "[:TRIGger:EBURst:SLOPe](#)" on page 981
- "[:TRIGger:EBURst:COUNT](#)" on page 979

:TRIGger:EBURst:SLOPe

N (see page 1232)

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 959

• ":TRIGger:EBURst:IDLE" on page 980

• ":TRIGger:EBURst:COUNT" on page 979

:TRIGger:EBURst:SOURce

(see page 1232)

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 959
- ":TRIGger:MODE" on page 968

:TRIGger[:EDGE] Commands

Table 134 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE] :COUPling {AC DC LFReject} (see page 985)	:TRIGger[:EDGE] :COUPling? (see page 985)	{AC DC LFReject}
:TRIGger[:EDGE] :LEVel <level> [,<source>] (see page 986)	:TRIGger[:EDGE] :LEVel? [<source>] (see page 986)	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 987)	:TRIGger[:EDGE] :REJect? (see page 987)	{OFF LFReject HFReject}

Table 134 :TRIGger[:EDGE] Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:SLOPe <polarity> (see page 988)	:TRIGger[:EDGE]:SLOPe ? (see page 988)	<polarity> ::= {POSitive NEGative EITHer ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 989)	:TRIGger[:EDGE]:SOURce? (see page 989)	<source> ::= {CHANnel<n> EXTernal LINE WGEN WGEN1 WGEN2 WMOD} for the DSO models <source> ::= {CHANnel<n> DIGital<d> EXTernal LINE WGEN WGEN1 WGEN2 WMOD} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format Note: WGEN and WGEN1 are equivalent. Note: WGEN2 only available on models with 2 WaveGen outputs.

:TRIGger[:EDGE]:COUPLing



(see page 1232)

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 959
- ":TRIGger:MODE" on page 968
- ":TRIGger[:EDGE]:REJect" on page 987

:TRIGger[:EDGE]:LEVel

(see page 1232)

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 959
- ":TRIGger[:EDGE]:SOURCE" on page 989
- ":EXTERNAL:RANGE" on page 340
- ":POD<n>:THRESHOLD" on page 555
- ":DIGITAL<d>:THRESHOLD" on page 313

:TRIGger[:EDGE]:REject

 (see page 1232)

Command Syntax :TRIGger[:EDGE]:REject <reject>

<reject> ::= {OFF | LFRReject | 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 959
- ":TRIGger:HFReject" on page 963
- ":TRIGger[:EDGE]:COUPLing" on page 985

:TRIGger[:EDGE]:SLOPe

(see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":TRIGger:TV:POLarity" on page 1029

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 41, “Programming Examples,”](#) starting on page 1241

:TRIGger[:EDGE]:SOURce

C (see page 1232)

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTERNAL | LINE | WGEN | WGEN1 | WGEN2
              | WMOD} for the DSO models

<source> ::= {CHANnel<n> | DIGITAL<d> | EXTERNAL | LINE | WGEN | WGEN1
              | WGEN2 | WMOD} for the MSO models

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

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

Note: WAVE and WGEN1 are equivalent.

Note: WGEN2 only available on models with 2 WaveGen outputs.
```

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTERNAL – triggers on the rear panel EXT TRIG IN signal.
- LINE – triggers at the 50% level of the rising or falling edge of the AC power source signal.
- WGEN, WGEN1, WGEN2 – triggers at the 50% level of the rising edge of the waveform generator output signal. This option is not available when the DC, NOISE, or CARDiac waveforms are selected.
- WMOD – when waveform generator FSK or FM modulation is used, triggers at the 50% level of the rising edge of the modulating signal.

Query Syntax

```
:TRIGger[:EDGE]:SOURce?
```

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | EXT | LINE | WGEN | WGEN1 | WGEN2 | WMOD
              | NONE} for the DSO models

<source> ::= {CHAN<n> | DIGITAL<d> | EXTERNAL | LINE | WGEN | WGEN1
              | WGEN2 | WMOD | NONE} for the MSO models
```

See Also

- "[Introduction to :TRIGger Commands](#)" on page 959
- "[:TRIGGER:MODE](#)" on page 968

Example Code

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces the
' edge trigger. Any channel can be selected.
myScope.WriteString ":TRIGger:EDGE:SOURce CHANnel1"
```

See complete example programs at: [Chapter 41, “Programming Examples,”](#) starting on page 1241

:TRIGger:GLITCH Commands

Table 135 :TRIGger:GLITCH Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITCH:GREAt erthan <greater_than_time>[s uffix] (see page 992)	:TRIGger:GLITCH:GREAt erthan? (see page 992)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITCH:LESSt han <less_than_time>[suff ix] (see page 993)	:TRIGger:GLITCH:LESSt han? (see page 993)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:GLITCH:LEVel <level> [<source>] (see page 994)	:TRIGger:GLITCH:LEVel ? (see page 994)	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:POLAr ity <polarity> (see page 995)	:TRIGger:GLITCH:POLAr ity? (see page 995)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITCH:QUALi fier <qualifier> (see page 996)	:TRIGger:GLITCH:QUALi fier? (see page 996)	<qualifier> ::= {GREaterthan LESSthan RANGE}

Table 135 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 997)	:TRIGger:GLITch:RANGE? ? (see page 997)	<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 998)	:TRIGger:GLITch:SOURce? ? (see page 998)	<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 1232)

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 959
 • ":TRIGger:GLITch:SOURce" on page 998
 • ":TRIGger:GLITch:QUALifier" on page 996
 • ":TRIGger:MODE" on page 968

:TRIGger:GLITch:LESSthan

N (see page 1232)

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 959
- ":TRIGger:GLITch:SOURce" on page 998
- ":TRIGger:GLITch:QUALifier" on page 996
- ":TRIGger:MODE" on page 968

:TRIGger:GLITch:LEVel

N (see page 1232)

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 959
- "[":TRIGger:MODE](#)" on page 968
- "[":TRIGger:GLITch:SOURce](#)" on page 998
- "[":EXTernal:RANGE](#)" on page 340

:TRIGger:GLITch:POLarity

N (see page 1232)

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 959
• ":TRIGger:MODE" on page 968
• ":TRIGger:GLITch:SOURce" on page 998

:TRIGger:GLITch:QUALifier

N (see page 1232)

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 959

• ":TRIGger:GLITch:SOURce" on page 998

• ":TRIGger:MODE" on page 968

:TRIGger:GLITch:RANGE

N (see page 1232)

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 959
 - ":TRIGger:GLITch:SOURce" on page 998
 - ":TRIGger:GLITch:QUALifier" on page 996
 - ":TRIGger:MODE" on page 968

:TRIGger:GLITch:SOURce

N (see page 1232)

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 959
 - "[:TRIGger:MODE](#)" on page 968
 - "[:TRIGger:GLITch:LEVel](#)" on page 994
 - "[:TRIGger:GLITch:POLarity](#)" on page 995
 - "[:TRIGger:GLITch:QUALifier](#)" on page 996
 - "[:TRIGger:GLITch:RANGE](#)" on page 997

Example Code

- "[Example Code](#)" on page 989

:TRIGger:OR Commands

Table 136 :TRIGger:OR Commands Summary

Command	Query	Options and Query Returns
:TRIGger:OR <string> (see page 1000)	:TRIGger:OR? (see page 1000)	<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 1232)

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 959
 - "[":TRIGger:MODE](#)" on page 968

:TRIGger:PATTERn Commands

Table 137 :TRIGger:PATTERn Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTERn<string>[,<edge_source>,<edge>] (see page 1002)	:TRIGger:PATTERn? (see page 1003)	<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:FORM at <base> (see page 1004)	:TRIGger:PATTERn:FORM at? (see page 1004)	<base> ::= {ASCII HEX}
:TRIGger:PATTERn:GREaterthan<greater_than_time>[suffix] (see page 1005)	:TRIGger:PATTERn:GREaterthan? (see page 1005)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:PATTERn:LESSthan<less_than_time>[suffix] (see page 1006)	:TRIGger:PATTERn:LESSthan? (see page 1006)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:PATTERn:QUALifier <qualifier> (see page 1007)	:TRIGger:PATTERn:QUALifier? (see page 1007)	<qualifier> ::= {ENTERed GREaterthan LESSthan INRange OUTRange TIMeout}
:TRIGger:PATTERn:RANGE<less_than_time>[suffix],<greater_than_time>[suffix] (see page 1009)	:TRIGger:PATTERn:RANGE? (see page 1009)	<less_than_time> ::= 15 ns to 10 seconds in NR3 format <greater_than_time> ::= 10 ns to 9.99 seconds in NR3 format [suffix] ::= {s ms us ns ps}

:TRIGger:PATTERn



(see page 1232)

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 959
- "[:TRIGger:PATTern:FORMAT](#)" on page 1004
- "[:TRIGger:PATTern:QUALifier](#)" on page 1007
- "[:TRIGger:MODE](#)" on page 968

:TRIGger:PATTERn:FORMAT

N (see page 1232)

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 959
 • ":TRIGger:PATTERn" on page 1002

:TRIGger:PATTERn:GREaterthan

N (see page 1232)

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 959
 - ":TRIGger:PATTERn" on page 1002
 - ":TRIGger:PATTERn:QUALifier" on page 1007
 - ":TRIGger:MODE" on page 968

:TRIGger:PATTERn:LESSthan

N (see page 1232)

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: LESS than?

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 959
- ":TRIGger:PATTERn" on page 1002
- ":TRIGger:PATTERn:QUALifier" on page 1007
- ":TRIGger:MODE" on page 968

:TRIGger:PATTERn:QUALifier

N (see page 1232)

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 959
- ":TRIGger:PATTERn:GREaterthan" on page 1005

- "[:TRIGGER:PATTERn:LESSthan](#)" on page 1006
- "[:TRIGGER:PATTERn:RANGE](#)" on page 1009

:TRIGger:PATTERn:RANGE

N (see page 1232)

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 959
- ":TRIGger:PATTERn" on page 1002
- ":TRIGger:PATTERn:QUALifier" on page 1007
- ":TRIGger:MODE" on page 968

:TRIGger:RUNT Commands

Table 138 :TRIGger:RUNT Commands Summary

Command	Query	Options and Query Returns
:TRIGger:RUNT:POLarit y <polarity> (see page 1011)	:TRIGger:RUNT:POLarit y? (see page 1011)	<polarity> ::= {POSitive NEGative EITHer}
:TRIGger:RUNT:QUALifi er <qualifier> (see page 1012)	:TRIGger:RUNT:QUALifi er? (see page 1012)	<qualifier> ::= {GREaterthan LESSthan NONE}
:TRIGger:RUNT:SOURce <source> (see page 1013)	:TRIGger:RUNT:SOURce? (see page 1013)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:RUNT:TIME <time>[suffix] (see page 1014)	:TRIGger:RUNT:TIME? (see page 1014)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

:TRIGger:RUNT:POLarity

N (see page 1232)

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 959

- ":TRIGger:MODE" on page 968
- ":TRIGger:LEVel:HIGH" on page 966
- ":TRIGger:LEVel:LOW" on page 967
- ":TRIGger:RUNT:SOURce" on page 1013

:TRIGger:RUNT:QUALifier

N (see [page 1232](#))

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 959
- "[:TRIGger:MODE](#)" on page 968
- "[:TRIGger:RUNT:TIME](#)" on page 1014

:TRIGger:RUNT:SOURce

N (see page 1232)

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 959

• ":TRIGger:RUNT:POLarity" on page 1011

:TRIGger:RUNT:TIME

N (see [page 1232](#))

Command Syntax :TRIGger:RUNT:TIME <time>[suffix]

<time> ::= floating-point number in NR3 format

[suffix] ::= {s | ms | us | ns | ps}

When triggering on runt pulses whose width is greater than or less than a certain value (see :TRIGger:RUNT:QUALifier), the :TRIGger:RUNT:TIME command specifies the time used with the qualifier.

Query Syntax :TRIGger:RUNT:TIME?

The :TRIGger:RUNT:TIME? query returns the current runt pulse qualifier time setting.

Return Format <time><NL>

<time> ::= floating-point number in NR3 format

See Also

- "Introduction to :TRIGger Commands" on [page 959](#)
- ":TRIGger:RUNT:QUALifier" on [page 1012](#)

:TRIGger:SHOLD Commands

Table 139 :TRIGger:SHOLD Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SHOLD:SLOPe <slope> (see page 1016)	:TRIGger:SHOLD:SLOPe? (see page 1016)	<slope> ::= {NEGative POSitive}
:TRIGger:SHOLD:SOURce :CLOCk <source> (see page 1017)	:TRIGger:SHOLD:SOURce :CLOCk? (see page 1017)	<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 1018)	:TRIGger:SHOLD:SOURce :DATA? (see page 1018)	<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:H OLD <time>[suffix] (see page 1019)	:TRIGger:SHOLD:TIME:H OLD? (see page 1019)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}
:TRIGger:SHOLD:TIME:S ETup <time>[suffix] (see page 1020)	:TRIGger:SHOLD:TIME:S ETup? (see page 1020)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

:TRIGger:SHOLD:SLOPe

N (see page 1232)

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 959
 • ":TRIGger:MODE" on page 968
 • ":TRIGger:SHOLD:SOURce:CLOCK" on page 1017
 • ":TRIGger:SHOLD:SOURce:DATA" on page 1018

:TRIGger:SHOLD:SOURce:CLOCk

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":TRIGger:SHOLD:SLOPe" on page 1016

:TRIGger:SHOLD:SOURce:DATA**N** (see page 1232)**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 959

• ":TRIGger:MODE" on page 968

• ":TRIGger:SHOLD:SLOPe" on page 1016

:TRIGger:SHOLD:TIME:HOLD

N (see page 1232)

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 959

:TRIGger:SHOLD:TIME:SETup

N (see page 1232)

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 959

:TRIGger:TRANSition Commands

The :TRIGger:TRANSition comamnds set the rise/fall time trigger options.

Table 140 :TRIGger:TRANSition Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TRANSition:QUALifier <qualifier> (see page 1022)	:TRIGger:TRANSition:QUALifier? (see page 1022)	<qualifier> ::= {GREaterthan LESSthan}
:TRIGger:TRANSition:SLOPe <slope> (see page 1023)	:TRIGger:TRANSition:SLOPe? (see page 1023)	<slope> ::= {NEGative POSitive}
:TRIGger:TRANSition:SOURce <source> (see page 1024)	:TRIGger:TRANSition:SOURce? (see page 1024)	<source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TRANSition:TIME <time>[suffix] (see page 1025)	:TRIGger:TRANSition:TIME? (see page 1025)	<time> ::= floating-point number in NR3 format [suffix] ::= {s ms us ns ps}

:TRIGger:TRANSition:QUALifier

N (see page 1232)

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 959
- ":TRIGger:TRANSition:TIME" on page 1025
- ":TRIGger:MODE" on page 968

:TRIGger:TRANSition:SLOPe

N (see page 1232)

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 959
• ":TRIGger:MODE" on page 968
• ":TRIGger:TRANSition:SOURce" on page 1024

:TRIGger:TRANSition:SOURce

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":TRIGger:TRANSition:SLOPe" on page 1023

:TRIGger:TRANSition:TIME

N (see page 1232)

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 959
 • ":TRIGger:TRANSition:QUALifier" on page 1022

:TRIGger:TV Commands

Table 141 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 1027)	:TRIGger:TV:LINE? (see page 1027)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 1028)	:TRIGger:TV:MODE? (see page 1028)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LINE LFIELD1 LFIELD2 LATERNATE}
:TRIGger:TV:POLarity <polarity> (see page 1029)	:TRIGger:TV:POLarity? (see page 1029)	<polarity> ::= {POSitive NEGative}
:TRIGger:TV:SOURce <source> (see page 1030)	:TRIGger:TV:SOURce? (see page 1030)	<source> ::= {CHANNEL<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANDARD <standard> (see page 1031)	:TRIGger:TV:STANDARD? (see page 1031)	<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:ENUM ber <count> (see page 1032)	:TRIGger:TV:UDTV:ENUM ber? (see page 1032)	<count> ::= edge number in NR1 format
:TRIGger:TV:UDTV:HSYN c {{0 OFF} {1 ON}} (see page 1033)	:TRIGger:TV:UDTV:HSYN c? (see page 1033)	{0 1}
:TRIGger:TV:UDTV:HTIM e <time> (see page 1034)	:TRIGger:TV:UDTV:HTIM e? (see page 1034)	<time> ::= seconds in NR3 format
:TRIGger:TV:UDTV:PGTH an <min_time> (see page 1035)	:TRIGger:TV:UDTV:PGTH an? (see page 1035)	<min_time> ::= seconds in NR3 format

:TRIGger:TV:LINE

N (see page 1232)

Command Syntax :TRIGger:TV:LINE <line_number>

<line_number> ::= integer in NR1 format

The :TRIGger:TV:LINE command allows triggering on a specific line of video. The line number limits vary with the standard and mode, as shown in the following table.

Table 142 TV Trigger Line Number Limits

TV Standard	Mode				
	LINE	LField1	LField2	LALternate	VERTical
NTSC		1 to 263	1 to 262	1 to 262	
PAL		1 to 313	314 to 625	1 to 312	
PAL-M		1 to 263	264 to 525	1 to 262	
SECAM		1 to 313	314 to 625	1 to 312	
GENERIC		1 to 1024	1 to 1024		1 to 1024
P480L60HZ	1 to 525				
P720L60HZ	1 to 750				
P1080L24HZ	1 to 1125				
P1080L25HZ	1 to 1125				
P1080L50HZ	1 to 1125				
P1080L60HZ	1 to 1125				
I1080L50HZ	1 to 1125				
I1080L60HZ	1 to 1125				

Query Syntax :TRIGger:TV:LINE?

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

Return Format <line_number><NL>

<line_number> ::= integer in NR1 format

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 959
 - "[":TRIGger:TV:STANDARD](#)" on page 1031
 - "[":TRIGger:TV:MODE](#)" on page 1028

:TRIGger:TV:MODE

N (see page 1232)

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 959
 - ":TRIGger:TV:STANDARD" on page 1031
 - ":TRIGger:MODE" on page 968

:TRIGger:TV:POLarity

N (see page 1232)

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 959
• ":TRIGger:MODE" on page 968
• ":TRIGger:TV:SOURce" on page 1030

:TRIGger:TV:SOURce

N (see page 1232)

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 959
 - ":TRIGger:MODE" on page 968
 - ":TRIGger:TV:POLarity" on page 1029

- Example Code**
- "Example Code" on page 989

:TRIGger:TV:STANDARD

N (see page 1232)

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 1232)

Command Syntax :TRIGger:TV:UDTV:ENUMber <count>

<count> ::= edge number in NR1 format

The :TRIGger:TV:UDTV:ENUMber command specifies the Generic video trigger's Nth edge to trigger on after synchronizing with the vertical sync.

Query Syntax :TRIGger:TV:UDTV:ENUMber?

The :TRIGger:TV:UDTV:ENUMber query returns the edge count setting.

Return Format <count><NL>

<count> ::= edge number in NR1 format

- See Also**
- "[:TRIGger:TV:STANDARD](#)" on page 1031
 - "[:TRIGger:TV:UDTV:PGTHAn](#)" on page 1035
 - "[:TRIGger:TV:UDTV:HSYNC](#)" on page 1033

:TRIGger:TV:UDTV:HSync

N (see page 1232)

Command Syntax :TRIGger:TV:UDTV:HSync {{0 | OFF} | {1 | ON}}

The :TRIGger:TV:UDTV:HSync command enables or disables the horizontal sync control in the Generic video trigger.

For interleaved video, enabling the HSync control and setting the HTIME adjustment to the sync time of the probed video signal allows the ENUMber function to count only lines and not double count during equalization. Additionally, the Field Holdoff can be adjusted so that the oscilloscope triggers once per frame.

Similarly, for progressive video with a tri-level sync, enabling the HSync control and setting the HTIME adjustment to the sync time of the probed video signal allows the ENUMber function to count only lines and not double count during vertical sync.

Query Syntax :TRIGger:TV:UDTV:HSync?

The :TRIGger:TV:UDTV:HSync query returns the horizontal sync control setting.

Return Format {0 | 1}

- See Also**
- "[:TRIGger:TV:STANDARD](#)" on page 1031
 - "[:TRIGger:TV:UDTV:HTIME](#)" on page 1034
 - "[:TRIGger:TV:UDTV:ENUMber](#)" on page 1032
 - "[:TRIGger:TV:UDTV:PGTHan](#)" on page 1035

:TRIGger:TV:UDTV:HTIMe

N (see page 1232)

Command Syntax :TRIGger:TV:UDTV:HTIMe <time>
<time> ::= seconds in NR3 format

When the Generic video trigger's horizontal sync control is enabled, the :TRIGger:TV:UDTV:HTIMe command sets the minimum time the horizontal sync pulse must be present to be considered valid.

Query Syntax :TRIGger:TV:UDTV:HTIMe?

The :TRIGger:TV:UDTV:HTIMe query returns the horizontal sync time setting.

Return Format <time><NL>
<time> ::= seconds in NR3 format

See Also • "[:TRIGger:TV:STANDARD](#)" on page 1031
• "[:TRIGger:TV:UDTV:HSYNC](#)" on page 1033

:TRIGger:TV:UDTV:PGTHan

N (see page 1232)

Command Syntax :TRIGger:TV:UDTV:PGTHan <min_time>
 <min_time> ::= seconds in NR3 format

The :TRIGger:TV:UDTV:PGTHan command specifies the "greater than the sync pulse width" time in the Generic video trigger. This setting allows oscilloscope synchronization to the vertical sync.

Query Syntax :TRIGger:TV:UDTV:PGTHan?

The :TRIGger:TV:UDTV:PGTHan query returns the "greater than the sync pulse width" time setting.

Return Format <min_time><NL>
 <min_time> ::= seconds in NR3 format

See Also

- "[:TRIGger:TV:STANDARD](#)" on page 1031
- "[:TRIGger:TV:UDTV:ENUMber](#)" on page 1032
- "[:TRIGger:TV:UDTV:HSync](#)" on page 1033

:TRIGger:ZONE Commands

Table 143 :TRIGger:ZONE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ZONE:SOURce <source> (see page 1037)	:TRIGger:ZONE:SOURce? (see page 1037)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:ZONE:STATE { {0 OFF} {1 ON} } (see page 1038)	:TRIGger:ZONE:STATE? (see page 1038)	{0 1}
:TRIGger:ZONE<n>:MODE <mode> (see page 1039)	:TRIGger:ZONE<n>:MODE? (see page 1039)	<mode> ::= {INTersect NOTintersect} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:PLACEMENT <width>, <height>, <x_center>, <y_center> (see page 1040)	:TRIGger:ZONE<n>:PLACEMENT? (see page 1040)	<width> ::= width of zone in seconds <height> ::= height of zone in volts <x_center> ::= center of zone in seconds <y_center> ::= center of zone in volts <n> ::= 1-2 in NR1 format
n/a	:TRIGger:ZONE<n>:VALIDity? (see page 1041)	<value> ::= {VALid INValid OSCreen} <n> ::= 1-2 in NR1 format
:TRIGger:ZONE<n>:STATE { {0 OFF} {1 ON} } (see page 1042)	:TRIGger:ZONE<n>:STATE? (see page 1042)	{0 1} <n> ::= 1-2 in NR1 format

:TRIGger:ZONE:SOURce

N (see page 1232)

Command Syntax :TRIGger:ZONE:SOURce <source>

<source> ::= {CHANnel<n>}
<n> ::= 1 to (# analog channels) in NR1 format

The :TRIGger:ZONE:SOURce command sets the analog source channel shared by all zones.

Query Syntax :TRIGger:ZONE:SOURce?

The :TRIGger:ZONE:SOURce? query returns the analog source channel specified for zone qualified triggers.

Return Format <source><NL>

<source> ::= {CHAN<n>}

See Also • "[:TRIGger:ZONE:STATe](#)" on page 1038

:TRIGger:ZONE:STATe

N (see page 1232)

Command Syntax :TRIGger:ZONE:STATe <on_off>

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

The :TRIGger:ZONE:STATe command enables or disables the zone qualified trigger feature.

When the zone qualified trigger is on, the zone(s) are actively being used to qualify the trigger.

Note that the :TRIGger:ZONE<n>:STATe setting must also be ON for a zone to be active.

Note that :TRIGger:ZONE:STATe mimics the behavior of the [Zone] key on the front panel, and :TRIGger:ZONE<n>:STATe mimics the behavior of the **Zone 1 On** and **Zone 2 On** softkeys. At least one zone's state must be on for the Zone Trigger feature (:TRIGger:ZONE:STATe) to be on. When the states of both individual zones are turned off, Zone Trigger is automatically turned off. In this case, when Zone Trigger is turned back on Zone 1 is forced to on. Otherwise, if at least one zone was on when Zone Trigger was turned off, the same configuration of individual zone on/off states will be restored when Zone Trigger is turned back on.

Query Syntax :TRIGger:ZONE:STATe?

The :TRIGger:ZONE:STATe? query returns whether the zone qualified trigger feature is enabled or disabled.

Return Format <on_off><NL>

<on_off> ::= {0 | 1}

See Also

- "[:TRIGger:ZONE<n>:STATe](#)" on page 1042
- "[:TRIGger:ZONE:SOURce](#)" on page 1037

:TRIGger:ZONE<n>:MODE

N (see page 1232)

Command Syntax :TRIGger:ZONE<n>:MODE <mode>
 <mode> ::= {INTersect | NOTintersect}
 <n> ::= 1-2 in NR1 format

The :TRIGger:ZONE<n>:MODE command sets the zone qualifying condition for Zone 1 or Zone 2 as either "Must Intersect" or "Must Not Intersect".

Query Syntax :TRIGger:ZONE<n>:MODE?

The :TRIGger:ZONE<n>:MODE? query returns the zone qualifying condition for Zone 1 or Zone 2.

Return Format <mode><NL>
 <mode> ::= {INT | NOT}

See Also • "[:TRIGger:ZONE<n>:STATE](#)" on page 1042
 • "[:TRIGger:ZONE<n>:PLACement](#)" on page 1040
 • "[:TRIGger:ZONE<n>:VALIDity](#)" on page 1041

:TRIGger:ZONE<n>:PLACement

N (see [page 1232](#))

Command Syntax :TRIGger:ZONE<n>:PLACement <width>, <height>, <x_center>, <y_center>
 <width> ::= width of zone in seconds
 <height> ::= height of zone in volts
 <x_center> ::= center of zone in seconds
 <y_center> ::= center of zone in volts
 <n> ::= 1-2 in NR1 format

The :TRIGger:ZONE<n>:PLACement command sets the size and location of Zone 1 or Zone 2.

No error is returned if the zone is placed off-screen, or if the zones overlap such that Zone 2 becomes invalid. The :TRIGger:ZONE<n>:VALidity? query is used to retrieve this information.

Query Syntax :TRIGger:ZONE<n>:PLACement?

The :TRIGger:ZONE<n>:PLACement? query returns the size and location of Zone 1 or Zone 2.

Return Format <opt><NL>
 <opt> ::= <width>, <height>, <x_center>, <y_center>

See Also • "[:TRIGger:ZONE<n>:STATE](#)" on page 1042
 • "[:TRIGger:ZONE<n>:MODE](#)" on page 1039
 • "[:TRIGger:ZONE<n>:VALidity](#)" on page 1041

:TRIGger:ZONE<n>:VALidity

N (see page 1232)

Query Syntax :TRIGger:ZONE<n>:VALidity?

<n> ::= 1-2 in NRI format

The :TRIGger:ZONE<n>:VALidity? query returns the validity of Zone 1 or Zone 2.

- INValid is returned (for Zone 2 only) when Zone 1 and Zone 2 overlap and have opposing qualifying conditions (modes). Zone 1 can never be invalid.
- OSCReen (off-screen) is returned when the associated zone is off-screen, and thus not being used to qualify the trigger.
- A zone is valid when it is neither invalid nor off-screen.

The validity of a zone is not affected by the zone's state. For example, a zone can be valid and off. You cannot directly set the validity of a zone.

Return Format <validity><NL>

<validity> ::= {VALid | INValid | OSCReen}

See Also • "[:TRIGger:ZONE<n>:STATE](#)" on page 1042

• "[:TRIGger:ZONE<n>:MODE](#)" on page 1039

• "[:TRIGger:ZONE<n>:PLACement](#)" on page 1040

:TRIGger:ZONE<n>:STATe

N (see page 1232)

Command Syntax `:TRIGger:ZONE<n>:STATe <on_off>`

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

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

`<n> ::= 1-2 in NR1 format`

The :TRIGger:ZONE<n>:STATe command sets the state for Zone 1 or Zone 2.

- When a zone's state is on, and the Zone Trigger feature is on (see "[:TRIGger:ZONE:STATe](#)" on page 1038), that zone is actively being used to qualify the trigger if it is not invalid or off-screen (see "[:TRIGger:ZONE<n>:VALidity](#)" on page 1041).
- When the Zone Trigger feature is off, no zones are being used to qualify the trigger, regardless of their individual states.

Note that :TRIGger:ZONE:STATe mimics the behavior of the **[Zone]** key on the front panel, and :TRIGger:ZONE<n>:STATe mimics the behavior of the **Zone 1 On** and **Zone 2 On** softkeys. At least one zone's state must be on for the Zone Trigger feature (:TRIGger:ZONE:STATe) to be on. When the states of both individual zones are turned off, Zone Trigger is automatically turned off. In this case, when Zone Trigger is turned back on Zone 1 is forced to on. Otherwise, if at least one zone was on when Zone Trigger was turned off, the same configuration of individual zone on/off states will be restored when Zone Trigger is turned back on.

Query Syntax `:TRIGger:ZONE<n>:STATe?`

The :TRIGger:ZONE<n>:STATe? query returns the state of Zone 1 or Zone 2.

Return Format `<on_off><NL>`

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

See Also

- "[:TRIGger:ZONE:STATe](#)" on page 1038
- "[:TRIGger:ZONE<n>:VALidity](#)" on page 1041

33

:WAVeform Commands

Provide access to waveform data. See "[Introduction to :WAVeform Commands](#)" on page 1045.

Table 144 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 1051)	:WAVeform:BYTeorder? (see page 1051)	<value> ::= {LSBFFirst MSBFFirst}
n/a	:WAVeform:COUNT? (see page 1052)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 1053)	<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 1055)	:WAVeform:FORMAT? (see page 1055)	<value> ::= {WORD BYTE ASCII}
:WAVeform:POINTS <# points> (see page 1056)	:WAVeform:POINTS? (see page 1056)	<# 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 144 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINTs:MODE <points_mode> (see page 1058)	:WAVeform:POINTs:MODE ? (see page 1058)	<points_mode> ::= {NORMal MAXimum RAW}
n/a	:WAVeform:PREamble? (see page 1060)	<p><preamble_block> ::= <format NR1>, <type NR1>, <points NR1>, <count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>, <yincrement NR3>, <yorigin NR3>, <yreference NR1></p> <p><format> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for BYTE format • 1 for WORD format • 2 for ASCII format <p><type> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for NORMAL type • 1 for PEAK detect type • 3 for AVERAGE type • 4 for HRESolution type <p><count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</p>
n/a	:WAVeform:SEGmented:COUNT? (see page 1063)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGmented:TAG? (see page 1064)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVeform:SOURce <source> (see page 1065)	:WAVeform:SOURce? (see page 1065)	<p><source> ::= {CHANnel<n> FUNCtion<m> MATH<m> SBUS} for DSO models</p> <p><source> ::= {CHANnel<n> POD{1 2} BUS{1 2} FUNCtion<m> MATH<m> SBUS} for MSO models</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p>
:WAVeform:SOURce:SUBS ource <subsource> (see page 1069)	:WAVeform:SOURce:SUBS ource? (see page 1069)	<subsource> ::= {{SUB0 RX MOSI} {SUB1 TX MISO}}
n/a	:WAVeform:TYPE? (see page 1070)	<return_mode> ::= {NORM PEAK AVER HRES}

Table 144 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:UNSIGNED { {0 OFF} {1 ON} } (see page 1071)	:WAVeform:UNSIGNED? (see page 1071)	{0 1}
:WAVeform:VIEW <view> (see page 1072)	:WAVeform:VIEW? (see page 1072)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 1073)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 1074)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 1075)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 1076)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 1077)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 1078)	<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 1053](#)) and :WAVeform:PREamble (see [page 1060](#)). 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 244](#)): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGItize command (see [page 203](#)) or :RUN command (see [page 224](#)), 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 1053](#)) 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 236](#)).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINTS command (see [page 1056](#)). 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 1056](#)). Only the magnitude values of each data point are transmitted. The first voltage value corresponds to the first time bucket on the left side of the screen and the last value corresponds to the next-to-last time bucket on the right side of the screen. Time buckets without data return 0. The time values for each data point correspond to the position of the data point in the data array. These time values are not transmitted.

AVERage Data

AVERage data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQuire:COUNT query (see [page 234](#)). Time buckets that have fewer than n hits return the average of the data they do have. If a time bucket does not have any data in it, it returns 0.

This data is transmitted over the interface linearly, starting with time bucket 0 and proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINTs? query (see [page 1056](#)). 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 1056](#)). 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 244](#)), the value returned by the :WAVeform:XINCREMENT query (see [page 1073](#)) 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:

$$\text{voltage} = [(\text{data value} - \text{yreference}) * \text{yincrement}] + \text{yorigin}$$

If the :WAVeform:FORMAT data format is ASCII (see [page 1055](#)), 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:

$$\text{time} = [(\text{data point number} - \text{xreference}) * \text{xincrement}] + \text{xorigin}$$

This would result in the following calculation for time bucket 3:

$$\text{time} = [(3 - 0) * 2 \text{ ns}] + 16 \text{ ns} = 22 \text{ ns}$$

In :ACQuire:TYPE PEAK mode (see [page 244](#)), because data is acquired in max-min pairs, modify the previous time formula to the following:

$$\text{time} = [(\text{data pair number} - \text{xreference}) * \text{xincrement} * 2] + \text{xorigin}$$

Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCII (see "[:WAVeform:FORMAT](#)" on page 1055). 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 1071](#)) 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 1055) 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 1051](#)) and :WAVeform:UNSIGNED (see [page 1071](#)) have no effect when the format is ASCII.

Data Format for Transfer - WORD format

WORD format (see "[:WAVeform:FORMAT](#)" on page 1055) 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 1056](#)). 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 1051](#)) 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 1055) 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 1051](#)) 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 1071](#)) 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 1055](#)) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see [page 1051](#)) 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 247](#)) 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 1232)

Command Syntax :WAVeform:BYTeorder <value>
 <value> ::= {LSBFFirst | MSBFFirst}

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFFirst 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 1045
 - "[:WAVeform:DATA](#)" on page 1053
 - "[:WAVeform:FORMAT](#)" on page 1055
 - "[:WAVeform:PREamble](#)" on page 1060

Example Code

- "[Example Code](#)" on page 1066
- "[Example Code](#)" on page 1061

:WAVeform:COUNt



(see [page 1232](#))

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 1045
- "[":ACQuire:COUNt](#)" on page 234
- "[":ACQuire:TYPE](#)" on page 244

:WAVeform:DATA



(see page 1232)

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 1045
- "[:WAVeform:UNSIGNED](#)" on page 1071
- "[:WAVeform:BYTeorder](#)" on page 1051
- "[:WAVeform:FORMat](#)" on page 1055
- "[:WAVeform:POINts](#)" on page 1056
- "[:WAVeform:PREamble](#)" on page 1060
- "[:WAVeform:SOURce](#)" on page 1065
- "[:WAVeform:TYPE](#)" on page 1070

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:
```

33 :WAVeform Commands

```
'           <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 41, “Programming Examples,” starting on page 1241](#)

:WAVeform:FORMAT

 (see page 1232)

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 1045
 - "[":WAVeform:BYTeorder](#)" on page 1051
 - "[":WAVeform:SOURce](#)" on page 1065
 - "[":WAVeform:DATA](#)" on page 1053
 - "[":WAVeform:PREamble](#)" on page 1060

- Example Code**
- "[Example Code](#)" on page 1066

:WAVeform:POINts

C (see page 1232)

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 1058) 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 1058) 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 1045
- "[:ACQuire:POINTs](#)" on page 236
- "[:WAVeform:DATA](#)" on page 1053
- "[:WAVeform:SOURce](#)" on page 1065
- "[:WAVeform:VIEW](#)" on page 1072
- "[:WAVeform:PREamble](#)" on page 1060
- "[:WAVeform:POINTs:MODE](#)" on page 1058

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 41](#), "Programming Examples," starting on page 1241

:WAVeform:POINts:MODE

N (see [page 1232](#))

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>
<points_mode> ::= {NORMAl | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be retrieved from the analog or digital sources.
- The second is referred to as the *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, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

**Considerations
for MAXimum or
RAW data
retrieval**

- The instrument must be stopped (see the :STOP command ([see page 228](#)) or the :DIGitize command ([see page 203](#)) 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.
- 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 1045
 - "[:WAVeform:DATA](#)" on page 1053
 - "[:ACQuire:POINts](#)" on page 236
 - "[:WAVeform:VIEW](#)" on page 1072
 - "[:WAVeform:PREamble](#)" on page 1060
 - "[:WAVeform:POINts](#)" on page 1056
 - "[:TIMEbase:MODE](#)" on page 949
 - "[:ACQuire:TYPE](#)" on page 244
 - "[:ACQuire:COUNt](#)" on page 234

:WAVEform:PREamble

(see page 1232)

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.

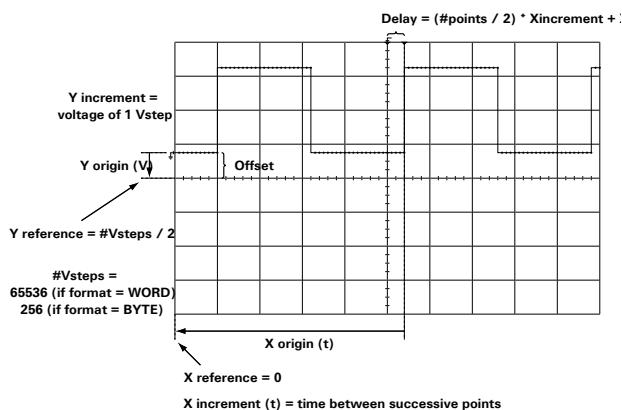
Return Format <preamble_block><NL>

```
<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 1045
- "":ACQuire:COUNt" on page 234
- "":ACQuire:POINTs" on page 236
- "":ACQuire:TYPE" on page 244

- "[:DIGITIZE](#)" on page 203
- "[:WAVeform:COUNt](#)" on page 1052
- "[:WAVeform:DATA](#)" on page 1053
- "[:WAVeform:FORMat](#)" on page 1055
- "[:WAVeform:POINts](#)" on page 1056
- "[:WAVeform:TYPE](#)" on page 1070
- "[:WAVeform:XINCrement](#)" on page 1073
- "[:WAVeform:XORigin](#)" on page 1074
- "[:WAVeform:XREFerence](#)" on page 1075
- "[:WAVeform:YINCrement](#)" on page 1076
- "[:WAVeform:YORigin](#)" on page 1077
- "[:WAVeform:YREFerence](#)" on page 1078

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)

```

33 :WAVeform Commands

```
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
```

See complete example programs at: [Chapter 41, “Programming Examples,”](#) starting on page 1241

:WAVeform:SEGmented:COUNt

N (see page 1232)

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 235
 - "[:ACQuire:SEGmented:COUNt](#)" on page 239
 - "[:DIGITIZE](#)" on page 203
 - "[:SINGle](#)" on page 226
 - "[:RUN](#)" on page 224
 - "[Introduction to :WAVeform Commands](#)" on page 1045

Example Code

- "[Example Code](#)" on page 240

:WAVeform:SEGmented:TTAG

N (see [page 1232](#))

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 240
 - "[Introduction to :WAVeform Commands](#)" on page 1045

Example Code

- "[Example Code](#)" on page 240

:WAVeform:SOURce

C (see page 1232)

Command Syntax

```
:WAVeform:SOURce <source>
<source> ::= {CHANnel<n> | FUNCTion<m> | MATH<m> | WMEMory<r>
              | SBUS{1 | 2}} for DSO models
<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCTion<m>
              | MATH<m> | WMEMory<r> | SBUS{1 | 2}}
              for MSO models
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, reference waveform, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply, integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS1 or SBUS2), ASCII is the only waveform format allowed, and the :WAVeform:DATA? query returns a string with timestamps and associated bus decode information.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCII formats (see "[:WAVeform:FORMAT](#)" on page 1055).

When the WORD format is chosen, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

When the ASCII format is chosen, the :WAVeform:DATA? query returns a string with unsigned decimal values separated by commas.

- When BUS1 or BUS2 is selected as the waveform source, you can choose the WORD or ASCII formats (but not BYTE because bus values are always returned as 16-bit values).

When the ASCII format is chosen, the :WAVeform:DATA? query returns a string with hexadecimal bus values, for example: 0x1938,0xff38,...

Query Syntax

:WAVeform:SOURce?

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE

MATH<m> is an alias for FUNCtion<m>. The :WAVeform:SOURce? query returns FUNC<m> if the source is FUNCtion<m> or MATH<m>.

Return Format

```
<source><NL>
<source> ::= {CHAN<n> | FUNC<m> | WMEM<r> | SBUS{1 | 2}} for DSO models
<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC<m>
| WMEM<r> | SBUS{1 | 2}} for MSO models
<n> ::= 1 to (# analog channels) in NR1 format
<m> ::= 1-4 in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
```

See Also

- "[Introduction to :WAVeform Commands](#)" on page 1045
- "[:DIGitize](#)" on page 203
- "[:WAVeform:FORMat](#)" on page 1055
- "[:WAVeform:BYTeorder](#)" on page 1051
- "[:WAVeform:DATA](#)" on page 1053
- "[:WAVeform:PREamble](#)" on page 1060

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 = " +
'
```

33 :WAVform Commands

```
        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 41](#), “Programming Examples,” starting on page 1241

:WAVeform:SOURce:SUBSource

(see page 1232)

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 1045
- "[":WAVeform:SOURce"](#) on page 1065

:WAVeform:TYPE

(see page 1232)

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 1045
- "[:ACQuire:TYPE](#)" on page 244
- "[:WAVeform:DATA](#)" on page 1053
- "[:WAVeform:PREamble](#)" on page 1060
- "[:WAVeform:SOURce](#)" on page 1065

:WAveform:UNSIGNED

 (see page 1232)

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 1045
 • ":WAveform:SOURce" on page 1065

:WAveform:VIEW

(see page 1232)

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 1045
• ":WAveform:POINts" on page 1056

:WAVeform:XINCrement



(see page 1232)

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 1045
 - ":WAVeform:PREamble" on page 1060

Example Code

- "Example Code" on page 1061

:WAVeform:XORigin



(see page 1232)

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 1045
 - "[:WAVeform:PREamble](#)" on page 1060
 - "[:WAVeform:XREFerence](#)" on page 1075

Example Code

- "[Example Code](#)" on page 1061

:WAVeform:XREFerence

 (see page 1232)

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 1045
- "`:WAVeform:PREamble`" on page 1060
- "`:WAVeform:XORigin`" on page 1074

Example Code

- "Example Code" on page 1061

:WAVeform:YINCrement



(see page 1232)

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 1045
 - "`:WAVeform:PREamble`" on page 1060

Example Code

- "Example Code" on page 1061

:WAVeform:YORigin



(see page 1232)

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 1045
 - ":WAVeform:PREamble" on page 1060
 - ":WAVeform:YREFerence" on page 1078

Example Code

- "Example Code" on page 1061

:WAVeform:YREFerence

(see page 1232)

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 1045
 - ":WAVeform:PREamble" on page 1060
 - ":WAVeform:YORigin" on page 1077

Example Code

- "Example Code" on page 1061

34

:WGEN<w> 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<w> commands are used to select the waveform function and parameters. See ["Introduction to :WGEN<w> Commands" on page 1082](#).

Table 145 :WGEN<w> Commands Summary

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:BYTeorder <order> (see page 1083)	:WGEN<w>:ARBitrary:BYTeorder? (see page 1083)	<order> ::= {MSBFFirst LSBFirst} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:DATA {<binary> <value>, <value> ...} (see page 1084)	n/a	<binary> ::= floating point values between -1.0 to +1.0 in IEEE 488.2 binary block format <value> ::= floating point values between -1.0 to +1.0 in comma-separated format <w> ::= 1 or 2 in NR1 format
n/a	:WGEN<w>:ARBitrary:DATTA:ATTRibute:POINTs? (see page 1085)	<points> ::= number of points in NR1 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:DATTA:CLEar (see page 1086)	n/a	<w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:DATTA:DAC {<binary> <value>, <value> ...} (see page 1087)	n/a	<binary> ::= decimal 16-bit integer values between -512 to +511 in IEEE 488.2 binary block format <value> ::= decimal integer values between -512 to +511 in comma-separated NR1 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:ARBitrary:INTerpolate {{0 OFF} {1 ON}} (see page 1088)	:WGEN<w>:ARBitrary:INTerpolate? (see page 1088)	{0 1} <w> ::= 1 or 2 in NR1 format



Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:ARBitrary:STORe <source> (see page 1089)	n/a	<p><source> ::= {CHANnel<n> WMEMory<r> FUNCtion<m> MATH<m>}</p> <p><n> ::= 1 to (# analog channels) in NR1 format</p> <p><r> ::= 1 to (# ref waveforms) in NR1 format</p> <p><m> ::= 1-4 in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FREQuency <frequency> (see page 1090)	:WGEN<w>:FREQuency? (see page 1090)	<p><frequency> ::= frequency in Hz in NR3 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion <signal> (see page 1091)	:WGEN<w>:FUNCtion? (see page 1093)	<p><signal> ::= {SINusoid SQuare RAMP PULSe NOISe DC SINC EXPrise EXPFall CARDiac GAUSSian ARBitrary}</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:PULSe:WIDTh <width> (see page 1094)	:WGEN<w>:FUNCtion:PULSe:WIDTh? (see page 1094)	<p><width> ::= pulse width in seconds in NR3 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:RAMP:SYMMetry <percent> (see page 1095)	:WGEN<w>:FUNCtion:RAMP:SYMMetry? (see page 1095)	<p><percent> ::= symmetry percentage from 0% to 100% in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:FUNCtion:SQUARE:DCYCle <percent> (see page 1096)	:WGEN<w>:FUNCtion:SQUARE:DCYCle? (see page 1096)	<p><percent> ::= duty cycle percentage from 20% to 80% in NR1 format</p> <p><w> ::= 1 or 2 in NR1 format</p>
:WGEN<w>:MODulation:AM:DEPTh <percent> (see page 1097)	:WGEN<w>:MODulation:AM:DEPTh? (see page 1097)	<p><percent> ::= AM depth percentage from 0% to 100% in NR1 format</p> <p><w> ::= 1 in NR1 format</p>
:WGEN<w>:MODulation:AM:FREQuency <frequency> (see page 1098)	:WGEN<w>:MODulation:AM:FREQuency? (see page 1098)	<p><frequency> ::= modulating waveform frequency in Hz in NR3 format</p> <p><w> ::= 1 in NR1 format</p>
:WGEN<w>:MODulation:FM:DEViation <frequency> (see page 1099)	:WGEN<w>:MODulation:FM:DEViation? (see page 1099)	<p><frequency> ::= frequency deviation in Hz in NR3 format</p> <p><w> ::= 1 in NR1 format</p>

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:MODulation:F M:FREQuency <frequency> (see page 1100)	:WGEN<w>:MODulation:F M:FREQuency? (see page 1100)	<frequency> ::= modulating waveform frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F SKey:FREQuency <percent> (see page 1101)	:WGEN<w>:MODulation:F SKey:FREQuency? (see page 1101)	<frequency> ::= hop frequency in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F SKey:RATE <rate> (see page 1102)	:WGEN<w>:MODulation:F SKey:RATE? (see page 1102)	<rate> ::= FSK modulation rate in Hz in NR3 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F UNCtion <shape> (see page 1103)	:WGEN<w>:MODulation:F UNCtion? (see page 1103)	<shape> ::= {SINusoid SQUare RAMP} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:F UNCtion:RAMP:SYMMetry <percent> (see page 1104)	:WGEN<w>:MODulation:F UNCtion:RAMP:SYMMetry? ? (see page 1104)	<percent> ::= symmetry percentage from 0% to 100% in NR1 format <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:N OISe <percent> (see page 1105)	:WGEN<w>:MODulation:N OISe? (see page 1105)	<percent> ::= 0 to 100 <w> ::= 1 or 2 in NR1 format
:WGEN<w>:MODulation:S TATE {{0 OFF} {1 ON}} (see page 1106)	:WGEN<w>:MODulation:S TATE? (see page 1106)	{0 1} <w> ::= 1 in NR1 format
:WGEN<w>:MODulation:T YPE <type> (see page 1107)	:WGEN<w>:MODulation:T YPE? (see page 1107)	<type> ::= {AM FM FSK} <w> ::= 1 in NR1 format
:WGEN<w>:OUTPut {{0 OFF} {1 ON}} (see page 1109)	:WGEN<w>:OUTPut? (see page 1109)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:OUTPut:LOAD <impedance> (see page 1110)	:WGEN<w>:OUTPut:LOAD? (see page 1110)	<impedance> ::= {ONEMeg FIFTy} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:OUTPut:POLArity <polarity> (see page 1111)	:WGEN<w>:OUTPut:POLArity? (see page 1111)	<polarity> ::= {NORMAL INVerted} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:PERiod <period> (see page 1112)	:WGEN<w>:PERiod? (see page 1112)	<period> ::= period in seconds in NR3 format <w> ::= 1 or 2 in NR1 format

Table 145 :WGEN<w> Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN<w>:RST (see page 1113)	n/a	<w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK {{0 OFF} {1 ON}} (see page 1114)	:WGEN<w>:TRACK? (see page 1114)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:AMPLItude {{0 OFF} {1 ON}} (see page 1115)	:WGEN<w>:TRACK:AMPLItude? (see page 1115)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:CSIGnal (see page 1116)	n/a	<w> ::= 1 or 2 in NR1 format
:WGEN<w>:TRACK:FREQuency {{0 OFF} {1 ON}} (see page 1117)	:WGEN<w>:TRACK:FREQuency? (see page 1117)	{0 1} <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage <amplitude> (see page 1118)	:WGEN<w>:VOLTage? (see page 1118)	<amplitude> ::= amplitude in volts in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:HIGH <high> (see page 1119)	:WGEN<w>:VOLTage:HIGH? (see page 1119)	<high> ::= high-level voltage in volts, in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:LOW <low> (see page 1120)	:WGEN<w>:VOLTage:LOW? (see page 1120)	<low> ::= low-level voltage in volts, in NR3 format <w> ::= 1 or 2 in NR1 format
:WGEN<w>:VOLTage:OFFSet <offset> (see page 1121)	:WGEN<w>:VOLTage:OFFS et? (see page 1121)	<offset> ::= offset in volts in NR3 format <w> ::= 1 or 2 in NR1 format

Introduction to :WGEN<w> Commands The :WGEN<w> subsystem provides commands to select the waveform generator function and parameters.

In the :WGEN<w> commands, the <w> can be 1 or 2, and :WGEN is equivalent to :WGEN1

Reporting the Setup

Use :WGEN<w>? to query setup information for the WGEN<w> subsystem.

Return Format

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

```
:WGEN1:FUNC SIN;OUTP 0;FREQ +1.0000E+03;VOLT +500.0E-03;VOLT:OFFS
+0.0E+00;:WGEN1:OUTP:LOAD ONEM
```

:WGEN<w>:ARBitrary:BYTeorder

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:BYTeorder <order>

<w> ::= 1 or 2 in NR1 format

<order> ::= {MSBFIRST | LSBFIRST}

The :WGEN<w>:ARBitrary:BYTeorder command selects the byte order for binary transfers.

Query Syntax :WGEN<w>:ARBitrary:BYTeorder?

The :WGEN<w>:ARBitrary:BYTeorder query returns the current byte order selection.

Return Format <order><NL>

<order> ::= {MSBFIRST | LSBFIRST}

See Also • "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084
• "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087

:WGEN<w>:ARBitrary:DATA

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:DATA {<binary> | <value>, <value> ...}

<w> ::= 1 or 2 in NR1 format

<binary> ::= floating point values between -1.0 to +1.0
 in IEEE 488.2 binary block format

<value> ::= floating point values between -1.0 to +1.0
 in comma-separated format

The :WGEN<w>:ARBitrary:DATA command downloads an arbitrary waveform in floating-point values format.

See Also • "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087

• "[:SAVE:ARBitrary\[:STARt\]](#)" on page 636

• "[:RECall:ARBitrary\[:STARt\]](#)" on page 625

:WGEN<w>:ARBitrary:DATA:ATTRibute:POINts

N (see page 1232)

Query Syntax :WGEN<w>:ARBitrary:DATA:ATTRibute:POINts?

<w> ::= 1 or 2 in NR1 format

The :WGEN<w>:ARBitrary:DATA:ATTRibute:POINts query returns the number of points used by the current arbitrary waveform.

Return Format <points> ::= number of points in NR1 format

- See Also**
- "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084
 - "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087
 - "[:SAVE:ARBitrary\[:STARt\]](#)" on page 636
 - "[:RECall:ARBitrary\[:STARt\]](#)" on page 625

:WGEN<w>:ARBitrary:DATA:CLEar

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:DATA:CLEar

<w> ::= 1 or 2 in NR1 format

The :WGEN<w>:ARBitrary:DATA:CLEar command clears the arbitrary waveform memory and loads it with the default waveform.

See Also

- "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084
- "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087
- "[:SAVE:ARBitrary\[:STARt\]](#)" on page 636
- "[:RECall:ARBitrary\[:STARt\]](#)" on page 625

:WGEN<w>:ARBitrary:DATA:DAC

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:DATA:DAC {<binary> | <value>, <value> ...}

<w> ::= 1 or 2 in NR1 format

<binary> ::= decimal 16-bit integer values between -512 to +511
in IEEE 488.2 binary block format

<value> ::= decimal integer values between -512 to +511
in comma-separated NR1 format

The :WGEN<w>:ARBitrary:DATA:DAC command downloads an arbitrary waveform using 16-bit integer (DAC) values.

See Also • "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084

• "[:SAVE:ARBitrary\[:STARt\]](#)" on page 636

• "[:RECall:ARBitrary\[:STARt\]](#)" on page 625

:WGEN<w>:ARBitrary:INTerpolate

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:INTerpolate {{0 | OFF} | {1 | ON}}

<w> ::= 1 or 2 in NR1 format

The :WGEN<w>:ARBitrary:INTerpolate command enables or disables the Interpolation control.

Interpolation specifies how lines are drawn between arbitrary waveform points:

- When ON, lines are drawn between points in the arbitrary waveform. Voltage levels change linearly between one point and the next.
- When OFF, all line segments in the arbitrary waveform are horizontal. The voltage level of one point remains until the next point.

Query Syntax :WGEN<w>:ARBitrary:INTerpolate?

The :WGEN<w>:ARBitrary:INTerpolate query returns the current interpolation setting.

Return Format {0 | 1}

See Also

- "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084
- "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087
- "[:SAVE:ARBitrary\[:STARt\]](#)" on page 636
- "[:RECall:ARBitrary\[:STARt\]](#)" on page 625

:WGEN<w>:ARBitrary:STORe

N (see page 1232)

Command Syntax :WGEN<w>:ARBitrary:STORe <source>

```

<w> ::= 1 or 2 in NR1 format
<source> ::= {CHANnel<n> | WMEMory<r> | FUNCtion<m> | MATH<m>}
<n> ::= 1 to (# analog channels) in NR1 format
<r> ::= 1 to (# ref waveforms) in NR1 format
<m> ::= 1-4 in NR1 format

```

The :WGEN<w>:ARBitrary:STORe command stores the source's waveform into the arbitrary waveform memory.

See Also

- "[:WGEN<w>:ARBitrary:DATA](#)" on page 1084
- "[:WGEN<w>:ARBitrary:DATA:DAC](#)" on page 1087
- "[:SAVE:ARBitrary\[:START\]](#)" on page 636
- "[:RECall:ARBitrary\[:START\]](#)" on page 625

:WGEN<w>:FREQuency

N (see [page 1232](#))

Command Syntax :WGEN<w>:FREQuency <frequency>

<w> ::= 1 or 2 in NR1 format

<frequency> ::= frequency in Hz in NR3 format

For all waveforms except Noise and DC, the :WGEN<w>:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN<w>:PERiod command.

Query Syntax :WGEN<w>:FREQuency?

The :WGEN<w>:FREQuency? query returns the currently set waveform generator frequency.

Return Format <frequency><NL>

<frequency> ::= frequency in Hz in NR3 format

See Also

- "[Introduction to :WGEN<w> Commands](#)" on page 1082
- "[:WGEN<w>:FUNCTION](#)" on page 1091
- "[:WGEN<w>:PERiod](#)" on page 1112

:WGEN<w>:FUNCtion

N (see page 1232)

Command Syntax :WGEN<w>:FUNCtion <signal>

<w> ::= 1 or 2 in NR1 format

<signal> ::= {SINusoid | SQUare | RAMP | PULSe | DC | NOISE | SINC
| EXPRIse | EXPFall | CARDiac | GAUSSian | ARBITrary}

The :WGEN<w>:FUNCtion command selects the type of waveform:

Waveform Type	Characteristics	Frequency Range	Max. Amplitude ² (High-Z) ¹	Offset ² (High-Z) ¹
SINusoid	<p>Use these commands to set the sine signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 	100 mHz to 20 MHz	20 mVpp to 10 Vpp	±4.00 V
SQUare	<p>Use these commands to set the square wave signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 • ":WGEN<w>:FUNCTION:SQUare:DCYCle" on page 1096 <p>The duty cycle can be adjusted from 20% to 80%.</p>	100 mHz to 10 MHz	20 mVpp to 10 Vpp	±5.00 V
RAMP	<p>Use these commands to set the ramp signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 • ":WGEN<w>:FUNCTION:RAMP:SYMMetry" on page 1095 <p>Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.</p>	100 mHz to 200 kHz	20 mVpp to 10 Vpp	±5.00 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude ² (High-Z) ¹	Offset ² (High-Z) ¹
PULSe	<p>Use these commands to set the pulse signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 • ":WGEN<w>:FUNCTION:PULSE:WIDTH" on page 1094 <p>The pulse width can be adjusted from 20 ns to the period minus 20 ns.</p>	100 mHz to 10 MHz.	20 mVpp to 10 Vpp	±5.00 V
DC	<p>Use this command to set the DC level:</p> <ul style="list-style-type: none"> • ":WGEN<w>:VOLTage:OFFSet" on page 1121 	n/a	n/a	±10.00 V
NOISe	<p>Use these commands to set the noise signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 	n/a	20 mVpp to 10 Vpp	±5.00 V
SINC	<p>Use these commands to set the sine cardinal signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 	100 mHz to 1 MHz	20 mVpp to 9 Vpp	±2.50 V
EXPRIse	<p>Use these commands to set the exponential rise signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 	100 mHz to 5 MHz	20 mVpp to 10 Vpp	±5.00 V
EXPFall	<p>Use these commands to set the exponential fall signal parameters:</p> <ul style="list-style-type: none"> • ":WGEN<w>:FREQuency" on page 1090 • ":WGEN<w>:PERiod" on page 1112 • ":WGEN<w>:VOLTage" on page 1118 • ":WGEN<w>:VOLTage:OFFSet" on page 1121 • ":WGEN<w>:VOLTage:HIGH" on page 1119 • ":WGEN<w>:VOLTage:LOW" on page 1120 	100 mHz to 5 MHz	20 mVpp to 10 Vpp	±5.00 V

Waveform Type	Characteristics	Frequency Range	Max. Amplitude ² (High-Z) ¹	Offset ² (High-Z) ¹
CARDiac	Use these commands to set the cardiac signal parameters: <ul style="list-style-type: none">• ":WGEN<w>:FREQuency" on page 1090• ":WGEN<w>:PERiod" on page 1112• ":WGEN<w>:VOLTage" on page 1118• ":WGEN<w>:VOLTage:OFFSet" on page 1121	100 mHz to 200 kHz	20 mVpp to 9 Vpp	±2.50 V
GAUSSian	Use these commands to set the gaussian pulse signal parameters: <ul style="list-style-type: none">• ":WGEN<w>:FREQuency" on page 1090• ":WGEN<w>:PERiod" on page 1112• ":WGEN<w>:VOLTage" on page 1118• ":WGEN<w>:VOLTage:OFFSet" on page 1121	100 mHz to 5 MHz	20 mVpp to 7.5 Vpp	±2.50 V
ARBitrary	Use these commands to set the arbitrary signal parameters: <ul style="list-style-type: none">• ":WGEN<w>:FREQuency" on page 1090• ":WGEN<w>:PERiod" on page 1112• ":WGEN<w>:VOLTage" on page 1118• ":WGEN<w>:VOLTage:OFFSet" on page 1121• ":WGEN<w>:VOLTage:HIGH" on page 1119• ":WGEN<w>:VOLTage:LOW" on page 1120	100 mHz to 12 MHz	20 mVpp to 10 Vpp	±5.00 V

¹When the output load is 50 Ω, these values are halved.

²The minimum amplitude is limited to 40 mVpp if the offset is greater than 500 mV or less than -500 mV. Likewise, the offset is limited to +/- 500 mV if the amplitude is less than 40 mVpp.

Query Syntax :WGEN<w>:FUNCTION?

The :WGEN<w>:FUNCTION? query returns the currently selected signal type.

Return Format <signal><NL>

```
<signal> ::= {SIN | SQU | RAMP | PULS | DC | NOIS | SINC | EXPR | EXPF
               | CARD | GAUS | ARB}
```

See Also

- "[Introduction to :WGEN<w> Commands](#)" on page 1082

- "[:WGEN<w>:MODulation:NOISE](#)" on page 1105

:WGEN<w>:FUNCtion:PULSe:WIDTH

N (see page 1232)

Command Syntax :WGEN<w>:FUNCtion:PULSe:WIDTH <width>

<w> ::= 1 or 2 in NR1 format

<width> ::= pulse width in seconds in NR3 format

For Pulse waveforms, the :WGEN<w>:FUNCtion:PULSe:WIDTH command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

Query Syntax :WGEN<w>:FUNCtion:PULSe:WIDTH?

The :WGEN<w>:FUNCtion:PULSe:WIDTH? query returns the currently set pulse width.

Return Format <width><NL>

<width> ::= pulse width in seconds in NR3 format

See Also • "Introduction to :WGEN<w> Commands" on page 1082
• ":WGEN<w>:FUNCtion" on page 1091

:WGEN<w>:FUNCTION:RAMP:SYMMetry

N (see page 1232)

Command Syntax :WGEN<w>:FUNCTION:RAMP:SYMMetry <percent>

<w> ::= 1 or 2 in NR1 format

<percent> ::= symmetry percentage from 0% to 100% in NR3 format

For Ramp waveforms, the :WGEN<w>:FUNCTION:RAMP:SYMMetry command specifies the symmetry of the waveform.

Symmetry represents the amount of time per cycle that the ramp waveform is rising.

Query Syntax :WGEN<w>:FUNCTION:RAMP:SYMMetry?

The :WGEN<w>:FUNCTION:RAMP:SYMMetry? query returns the currently set ramp symmetry.

Return Format <percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR3 format

See Also • "Introduction to :WGEN<w> Commands" on page 1082

• ":WGEN<w>:FUNCTION" on page 1091

:WGEN<w>:FUNCTION:SQUARE:DCYCle

N

(see page 1232)

Command Syntax

`:WGEN<w>:FUNCTION:SQUARE:DCYCle <percent>`

`<w> ::= 1 or 2 in NR1 format`

`<percent> ::= duty cycle percentage from 20% to 80% in NR3 format`

For Square waveforms, the :WGEN<w>:FUNCTION:SQUARE:DCYCle command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

Query Syntax

`:WGEN<w>:FUNCTION:SQUARE:DCYCle?`

The :WGEN<w>:FUNCTION:SQUARE:DCYCle? query returns the currently set square wave duty cycle.

Return Format

`<percent><NL>`

`<percent> ::= duty cycle percentage from 20% to 80% in NR3 format`

See Also

- "Introduction to :WGEN<w> Commands" on page 1082
- ":WGEN<w>:FUNCTION" on page 1091

:WGEN<w>:MODulation:AM:DEPTH

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:AM:DEPTH <percent>

<w> ::= 1 in NR1 format

<percent> ::= AM depth percentage from 0% to 100% in NR1 format

The :WGEN<w>:MODulation:AM:DEPTH command specifies the amount of amplitude modulation.

AM Depth refers to the portion of the amplitude range that will be used by the modulation. For example, a depth setting of 80% causes the output amplitude to vary from 10% to 90% (90% – 10% = 80%) of the original amplitude as the modulating signal goes from its minimum to maximum amplitude.

Query Syntax :WGEN<w>:MODulation:AM:DEPTH?

The :WGEN<w>:MODulation:AM:DEPTH? query returns the AM depth percentage setting.

Return Format <percent><NL>

<percent> ::= AM depth percentage from 0% to 100% in NR1 format

See Also • "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098

• "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099

• "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100

• "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101

• "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102

• "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103

• "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104

• "[:WGEN<w>:MODulation:STATE](#)" on page 1106

• "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:AM:FREQuency

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:AM:FREQuency <frequency>
 <w> ::= 1 in NR1 format
 <frequency> ::= modulating waveform frequency in Hz in NR3 format
 The :WGEN<w>:MODulation:AM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax :WGEN<w>:MODulation:AM:FREQuency?
 The :WGEN<w>:MODulation:AM:FREQuency? query returns the frequency of the modulating signal.

Return Format <frequency><NL>
 <frequency> ::= modulating waveform frequency in Hz in NR3 format

See Also

- "[:WGEN<w>:MODulation:AM:DEPTh](#)" on page 1097
- "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
- "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
- "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
- "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
- "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
- "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
- "[:WGEN<w>:MODulation:STATe](#)" on page 1106
- "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FM:DEViation

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:FM:DEViation <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= frequency deviation in Hz in NR3 format

The :WGEN<w>:MODulation:FM:DEViation command specifies the frequency deviation from the original carrier signal frequency.

When the modulating signal is at its maximum amplitude, the output frequency is the carrier signal frequency plus the deviation amount, and when the modulating signal is at its minimum amplitude, the output frequency is the carrier signal frequency minus the deviation amount.

The frequency deviation cannot be greater than the original carrier signal frequency.

Also, the sum of the original carrier signal frequency and the frequency deviation must be less than or equal to the maximum frequency for the selected waveform generator function plus 100 kHz.

Query Syntax :WGEN<w>:MODulation:FM:DEViation?

The :WGEN<w>:MODulation:FM:DEViation? query returns the frequency deviation setting.

Return Format <frequency><NL>

<frequency> ::= frequency deviation in Hz in NR3 format

- See Also**
- "[:WGEN<w>:MODulation:AM:DEPTH](#)" on page 1097
 - "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
 - "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
 - "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
 - "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
 - "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
 - "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
 - "[:WGEN<w>:MODulation:STATE](#)" on page 1106
 - "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FM:FREQuency

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:FM:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= modulating waveform frequency in Hz in NR3 format

The :WGEN<w>:MODulation:FM:FREQuency command specifies the frequency of the modulating signal.

Query Syntax :WGEN<w>:MODulation:FM:FREQuency?

The :WGEN<w>:MODulation:FM:FREQuency? query returns the frequency of the modulating signal.

Return Format <frequency><NL>

<frequency> ::= modulating waveform frequency in Hz in NR3 format

See Also • "[:WGEN<w>:MODulation:AM:DEPTh](#)" on page 1097

• "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098

• "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099

• "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101

• "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102

• "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103

• "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104

• "[:WGEN<w>:MODulation:STATe](#)" on page 1106

• "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FSKey:FREQuency

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:FSKey:FREQuency <frequency>

<w> ::= 1 in NR1 format

<frequency> ::= hop frequency in Hz in NR3 format

The :WGEN<w>:MODulation:FSKey:FREQuency command specifies the "hop frequency".

The output frequency "shifts" between the original carrier frequency and this "hop frequency".

Query Syntax :WGEN<w>:MODulation:FSKey:FREQuency?

The :WGEN<w>:MODulation:FSKey:FREQuency? query returns the "hop frequency" setting.

Return Format <frequency><NL>

<frequency> ::= hop frequency in Hz in NR3 format

- See Also**
- "[:WGEN<w>:MODulation:AM:DEPTH](#)" on page 1097
 - "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
 - "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
 - "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
 - "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
 - "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
 - "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
 - "[:WGEN<w>:MODulation:STATE](#)" on page 1106
 - "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FSKey:RATE

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:FSKey:RATE <rate>

<w> ::= 1 in NR1 format

<rate> ::= FSK modulation rate in Hz in NR3 format

The :WGEN<w>:MODulation:FSKey:RATE command specifies the rate at which the output frequency "shifts".

The FSK rate specifies a digital square wave modulating signal.

Query Syntax :WGEN<w>:MODulation:FSKey:RATE?

The :WGEN<w>:MODulation:FSKey:RATE? query returns the FSK rate setting.

Return Format <rate><NL>

<rate> ::= FSK modulation rate in Hz in NR3 format

- See Also**
- "[:WGEN<w>:MODulation:AM:DEPTH](#)" on page 1097
 - "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
 - "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
 - "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
 - "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
 - "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
 - "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
 - "[:WGEN<w>:MODulation:STATE](#)" on page 1106
 - "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FUNCTION

N (see page 1232)

Command Syntax

```
:WGEN<w>:MODulation:FUNCTION <shape>
<w> ::= 1 in NR1 format
<shape> ::= {SINusoid | SQUare| RAMP}
```

The :WGEN<w>:MODulation:FUNCTION command specifies the shape of the modulating signal.

When the RAMP shape is selected, you can specify the amount of time per cycle that the ramp waveform is rising with the :WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry command.

This command applies to AM and FM modulation. (The FSK modulation signal is a square wave shape.)

Query Syntax

```
:WGEN<w>:MODulation:FUNCTION?
```

The :WGEN<w>:MODulation:FUNCTION? query returns the specified modulating signal shape.

Return Format

```
<shape><NL>
<shape> ::= {SIN | SQU| RAMP}
```

See Also

- "[:WGEN<w>:MODulation:AM:DEPTH](#)" on page 1097
- "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
- "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
- "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
- "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
- "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
- "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
- "[:WGEN<w>:MODulation:STATE](#)" on page 1106
- "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry <percent>

<w> ::= 1 in NR1 format

<percent> ::= symmetry percentage from 0% to 100% in NR1 format

The :WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry command specifies the amount of time per cycle that the ramp waveform is rising. The ramp modulating waveform shape is specified with the :WGEN<w>:MODulation:FUNCTION command.

Query Syntax :WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry?

The :WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry? query returns ramp symmetry percentage setting.

Return Format <percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR1 format

See Also

- "[:WGEN<w>:MODulation:AM:DEPTH](#)" on page 1097
- "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
- "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
- "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
- "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
- "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
- "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
- "[:WGEN<w>:MODulation:STATE](#)" on page 1106
- "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:NOISe

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:NOISe <percent>
 <w> ::= 1 or 2 in NR1 format
 <percent> ::= 0 to 100

The :WGEN<w>:MODulation:NOISe command adds noise to the currently selected signal. The sum of the amplitude between the original signal and injected noise is limited to the regular amplitude limit (for example, 5 Vpp in 1 MOhm), so the range for <percent> varies according to current amplitude.

Note that adding noise affects edge triggering on the waveform generator source as well as the waveform generator sync pulse output signal (which can be sent to TRIG OUT). This is because the trigger comparator is located after the noise source.

Query Syntax :WGEN<w>:MODulation:NOISe?

The :WGEN<w>:MODulation:NOISe query returns the percent of added noise.

Return Format <percent><NL>
 <percent> ::= 0 to 100

See Also • "[:WGEN<w>:FUNCTION](#)" on page 1091

:WGEN<w>:MODulation:STATe

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:STATe <setting>

<w> ::= 1 in NR1 format

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

The :WGEN<w>:MODulation:STATe command enables or disables modulated waveform generator output.

You can enable modulation for all waveform generator function types except pulse, DC, and noise.

Query Syntax :WGEN<w>:MODulation:STATe?

The :WGEN<w>:MODulation:STATe? query returns whether the modulated waveform generator output is enabled or disabled.

Return Format <setting><NL>

<setting> ::= {0 | 1}

See Also

- "[:WGEN<w>:MODulation:AM:DEPTh](#)" on page 1097
- "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
- "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
- "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100
- "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
- "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
- "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
- "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
- "[:WGEN<w>:MODulation:TYPE](#)" on page 1107

:WGEN<w>:MODulation:TYPE

N (see page 1232)

Command Syntax :WGEN<w>:MODulation:TYPE <type>
 <w> ::= 1 in NR1 format
 <type> ::= {AM | FM | FSK}

The :WGEN<w>:MODulation:TYPE command selects the modulation type:

- AM (amplitude modulation) – the amplitude of the original carrier signal is modified according to the amplitude of the modulating signal.

Use the :WGEN<w>:MODulation:AM:FREQuency command to set the modulating signal frequency.

Use the :WGEN<w>:MODulation:AM:DEPTh command to specify the amount of amplitude modulation.

- FM (frequency modulation) – the frequency of the original carrier signal is modified according to the amplitude of the modulating signal.

Use the :WGEN<w>:MODulation:FM:FREQuency command to set the modulating signal frequency.

Use the :WGEN<w>:MODulation:FM:DEViation command to specify the frequency deviation from the original carrier signal frequency.

- FSK (frequency-shift keying modulation) – the output frequency "shifts" between the original carrier frequency and a "hop frequency" at the specified FSK rate.

The FSK rate specifies a digital square wave modulating signal.

Use the :WGEN<w>:MODulation:FSKey:FREQuency command to specify the "hop frequency".

Use the :WGEN<w>:MODulation:FSKey:RATE command to specify the rate at which the output frequency "shifts".

Query Syntax :WGEN<w>:MODulation:TYPE?

The :WGEN<w>:MODulation:TYPE? query returns the selected modulation type.

Return Format <type><NL>
 <type> ::= {AM | FM | FSK}

See Also

- "[:WGEN<w>:MODulation:AM:DEPTh](#)" on page 1097
- "[:WGEN<w>:MODulation:AM:FREQuency](#)" on page 1098
- "[:WGEN<w>:MODulation:FM:DEViation](#)" on page 1099
- "[:WGEN<w>:MODulation:FM:FREQuency](#)" on page 1100

- "[:WGEN<w>:MODulation:FSKey:FREQuency](#)" on page 1101
- "[:WGEN<w>:MODulation:FSKey:RATE](#)" on page 1102
- "[:WGEN<w>:MODulation:FUNCTION](#)" on page 1103
- "[:WGEN<w>:MODulation:FUNCTION:RAMP:SYMMetry](#)" on page 1104
- "[:WGEN<w>:MODulation:STATe](#)" on page 1106

:WGEN<w>:OUTPut

N (see page 1232)

Command Syntax :WGEN<w>:OUTPut <on_off>

<w> ::= 1 or 2 in NR1 format

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

The :WGEN<w>:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

Query Syntax :WGEN<w>:OUTPut?

The :WGEN<w>:OUTPut? query returns the current state of the waveform generator output setting.

Return Format <on_off><NL>

<on_off> ::= {1 | 0}

See Also • "Introduction to :WGEN<w> Commands" on page 1082

:WGEN<w>:OUTPut:LOAD

N (see page 1232)

Command Syntax `:WGEN<w>:OUTPut:LOAD <impedance>`
`<w> ::= 1 or 2 in NR1 format`
`<impedance> ::= {ONEMeg | FIFTy}`

The :WGEN<w>:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

Query Syntax `:WGEN<w>:OUTPut:LOAD?`

The :WGEN<w>:OUTPut:LOAD? query returns the current expected output load impedance.

Return Format `<impedance><NL>`
`<impedance> ::= {ONEM | FIFT}`

See Also • "Introduction to :WGEN<w> Commands" on page 1082

:WGEN<w>:OUTPut:POLarity

N (see [page 1232](#))

Command Syntax :WGEN<w>:OUTPut:POLarity <polarity>

<w> ::= 1 or 2 in NR1 format

<polarity> ::= {NORMAL | INVERTed}

The :WGEN<w>:OUTPut:POLarity command specifies whether the waveform generator output is inverted..

Query Syntax :WGEN<w>:OUTPut:POLarity?

The :WGEN<w>:OUTPut:POLarity? query returns the specified output polarity.

Return Format <polarity><NL>

<polarity> ::= {NORM | INV}

See Also • ["Introduction to :WGEN<w> Commands" on page 1082](#)

:WGEN<w>:PERiod

N (see page 1232)

Command Syntax :WGEN<w>:PERiod <period>

<w> ::= 1 or 2 in NR1 format

<period> ::= period in seconds in NR3 format

For all waveforms except Noise and DC, the :WGEN<w>:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN<w>:FREQuency command.

Query Syntax :WGEN<w>:PERiod?

The :WGEN<w>:PERiod? query returns the currently set waveform generator period.

Return Format <period><NL>

<period> ::= period in seconds in NR3 format

- See Also**
- "Introduction to :WGEN<w> Commands" on page 1082
 - ":WGEN<w>:FUNCTION" on page 1091
 - ":WGEN<w>:FREQuency" on page 1090

:WGEN<w>:RST

N (see page 1232)

Command Syntax :WGEN<w>:RST

<w> ::= 1 or 2 in NR1 format

The :WGEN<w>:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

See Also

- "Introduction to :WGEN<w> Commands" on page 1082
- ":WGEN<w>:FUNCTION" on page 1091
- ":WGEN<w>:FREQuency" on page 1090

:WGEN<w>:TRACK

N (see page 1232)

Command Syntax :WGEN<w>:TRACK <on_off>

```
<w> ::= 1 or 2 in NR1 format
<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :WGEN<w>:TRACK ON command causes frequency, amplitude, offset, and duty cycle adjustments to this waveform generator output signal to be tracked by the other waveform generator output.

Not all waveform shapes can be frequency tracked. When :WGEN<w>:TRACK or :WGEN<w>:TRACK:FREQuency is enabled, the other generator's waveform selections (:WGEN<w>:FUNCtion) must be in the same group as this generator's waveform:

Group 1	Group 2	Group 3	Group 4
SINusoid RAMP SINC CARDiac GAUSSian	SQUare PULSe	EXPRIse EXPFall	ARBitrary

When tracking is enabled, adjustments for tracked settings in the other waveform generator become unavailable.

Query Syntax :WGEN<w>:TRACK?

The :WGEN<w>:TRACK? query returns whether tracking is on or off.

Return Format <on_off><NL>

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

- See Also**
- ":WGEN<w>:FUNCTION" on page 1091
 - ":WGEN<w>:TRACK:AMPLitude" on page 1115
 - ":WGEN<w>:TRACK:FREQuency" on page 1117
 - ":WGEN<w>:TRACK:CSIGnal" on page 1116

:WGEN<w>:TRACk:AMPLitude

N (see page 1232)

Command Syntax :WGEN<w>:TRACk:AMPLitude <on_off>
 <w> ::= 1 or 2 in NR1 format
 <on_off> ::= {{1 | ON} | {0 | OFF}}

The :WGEN<w>:TRACk:AMPLitude ON command causes amplitude and offset adjustments to this waveform generator output signal to be tracked by the other waveform generator output.

Also, when tracking is enabled, adjustments for tracked settings in the other waveform generator become unavailable (ghosted).

Query Syntax :WGEN<w>:TRACk:AMPLitude?

The :WGEN<w>:TRACk:AMPLitude? query returns whether amplitude tracking is on or off.

Return Format <on_off><NL>
 <on_off> ::= {1 | 0}

See Also • "[:WGEN<w>:TRACk](#)" on page 1114
 • "[:WGEN<w>:TRACk:FREQuency](#)" on page 1117
 • "[:WGEN<w>:TRACk:CSIGnal](#)" on page 1116

:WGEN<w>:TRACk:CSIGnal

N (see page 1232)

Command Syntax

`:WGEN<w>:TRACK:CSIGNAL`

The `:WGEN<w>:TRACk:CSIGnal` command sets the other waveform generator output to be identical to this waveform generator output (except that the shape of either output may be inverted).

If the signal selected is ARB, the ARB waveform is copied as well.

See Also

- "`:WGEN<w>:TRACK`" on page 1114
- "`:WGEN<w>:TRACk:AMPLitude`" on page 1115
- "`:WGEN<w>:TRACk:FREQuency`" on page 1117

:WGEN<w>:TRACk:FREQuency

N (see page 1232)

Command Syntax :WGEN<w>:TRACk:FREQuency <on_off>

<w> ::= 1 or 2 in NR1 format

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

The :WGEN<w>:TRACk:FREQuency ON command causes frequency adjustments to this waveform generator output signal to be tracked by the other waveform generator output.

Not all waveform shapes can be frequency tracked. When :WGEN<w>:TRACk or :WGEN<w>:TRACk:FREQuency is enabled, the other generator's waveform selections (:WGEN<w>:FUNCtion) must be in the same group as this generator's waveform:

Group 1	Group 2	Group 3	Group 4
SINusoid RAMP SINC CARDiac GAUSSian	SQUare PULSe	EXPRIse EXPFall	ARBitrary

When tracking is enabled, adjustments for tracked settings in the other waveform generator become unavailable.

Query Syntax :WGEN<w>:TRACk:FREQuency?

The :WGEN<w>:TRACk:FREQuency? query returns whether frequency tracking is on or off.

Return Format <on_off><NL>

<on_off> ::= {1 | 0}

See Also • "[:WGEN<w>:FUNCtion](#)" on page 1091

• "[:WGEN<w>:TRACk](#)" on page 1114

• "[:WGEN<w>:TRACk:AMPLitude](#)" on page 1115

• "[:WGEN<w>:TRACk:CSIGnal](#)" on page 1116

:WGEN<w>:VOLTage

N (see page 1232)

Command Syntax :WGEN<w>:VOLTage <amplitude>

<w> ::= 1 or 2 in NR1 format

<amplitude> ::= amplitude in volts in NR3 format

For all waveforms except DC, the :WGEN<w>:VOLTage command specifies the waveform's amplitude. Use the :WGEN<w>:VOLTage:OFFSet command to specify the offset voltage or DC level.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

Query Syntax :WGEN<w>:VOLTage?

The :WGEN<w>:VOLTage? query returns the currently specified waveform amplitude.

Return Format <amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format

- See Also**
- "Introduction to :WGEN<w> Commands" on page 1082
 - ":WGEN<w>:FUNCTION" on page 1091
 - ":WGEN<w>:VOLTage:OFFSet" on page 1121
 - ":WGEN<w>:VOLTage:HIGH" on page 1119
 - ":WGEN<w>:VOLTage:LOW" on page 1120

:WGEN<w>:VOLTage:HIGH

N (see page 1232)

Command Syntax :WGEN<w>:VOLTage:HIGH <high>

<w> ::= 1 or 2 in NR1 format

<high> ::= high-level voltage in volts, in NR3 format

For all waveforms except DC, the :WGEN<w>:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN<w>:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

Query Syntax :WGEN<w>:VOLTage:HIGH?

The :WGEN<w>:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

Return Format <high><NL>

<high> ::= high-level voltage in volts, in NR3 format

- See Also**
- "[Introduction to :WGEN<w> Commands](#)" on page 1082
 - "[:WGEN<w>:FUNCTION](#)" on page 1091
 - "[:WGEN<w>:VOLTage:LOW](#)" on page 1120
 - "[:WGEN<w>:VOLTage](#)" on page 1118
 - "[:WGEN<w>:VOLTage:OFFSet](#)" on page 1121

:WGEN<w>:VOLTage:LOW

N (see page 1232)

Command Syntax :WGEN<w>:VOLTage:LOW <low>

<w> ::= 1 or 2 in NR1 format

<low> ::= low-level voltage in volts, in NR3 format

For all waveforms except DC, the :WGEN<w>:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN<w>:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN<w>:VOLTage and :WGEN<w>:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

Query Syntax :WGEN<w>:VOLTage:LOW?

The :WGEN<w>:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

Return Format <low><NL>

<low> ::= low-level voltage in volts, in NR3 format

- See Also**
- "Introduction to :WGEN<w> Commands" on page 1082
 - ":WGEN<w>:FUNCTION" on page 1091
 - ":WGEN<w>:VOLTage:LOW" on page 1120
 - ":WGEN<w>:VOLTage" on page 1118
 - ":WGEN<w>:VOLTage:OFFSet" on page 1121

:WGEN<w>:VOLTage:OFFSet

N (see page 1232)

Command Syntax :WGEN<w>:VOLTage:OFFSet <offset>

<w> ::= 1 or 2 in NR1 format

<offset> ::= offset in volts in NR3 format

The :WGEN<w>:VOLTage:OFFSet command specifies the waveform's offset voltage or the DC level. Use the :WGEN<w>:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN<w>:VOLTage:HIGH and :WGEN<w>:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

Query Syntax :WGEN<w>:VOLTage:OFFSet?

The :WGEN<w>:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

Return Format <offset><NL>

<offset> ::= offset in volts in NR3 format

- See Also**
- "Introduction to :WGEN<w> Commands" on page 1082
 - ":WGEN<w>:FUNCTION" on page 1091
 - ":WGEN<w>:VOLTage" on page 1118
 - ":WGEN<w>:VOLTage:HIGH" on page 1119
 - ":WGEN<w>:VOLTage:LOW" on page 1120

35

:WMEMory<r> Commands

Control reference waveforms.

Table 146 :WMEMory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMory<r>:CLEar (see page 1125)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format
:WMEMory<r>:DISPlay { {0 OFF} {1 ON} } (see page 1126)	:WMEMory<r>:DISPLAY? (see page 1126)	<r> ::= 1 to (# ref waveforms) in NR1 format {0 1}
:WMEMory<r>:LABEL <string> (see page 1127)	:WMEMory<r>:LABEL? (see page 1127)	<r> ::= 1 to (# ref waveforms) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMory<r>:SAVE <source> (see page 1128)	n/a	<r> ::= 1 to (# ref waveforms) in NR1 format <source> ::= {CHANnel<n> FUNCTion<m> MATH<m>} <n> ::= 1 to (# analog channels) in NR1 format <m> ::= 1-4 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMory<r>:SKEW <skew> (see page 1129)	:WMEMory<r>:SKEW? (see page 1129)	<r> ::= 1 to (# ref waveforms) in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMory<r>:YOFFSET <offset>[suffix] (see page 1130)	:WMEMory<r>:YOFFSET? (see page 1130)	<r> ::= 1 to (# ref waveforms) in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V mV}



35 :WMEMory<r> Commands

Table 146 :WMEMory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEMory<r>:YRAnge <range>[suffix] (see page 1131)	:WMEMory<r>:YRAnge? (see page 1131)	<r> ::= 1 to (# ref waveforms) in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V mV}
:WMEMory<r>:YScale <scale>[suffix] (see page 1132)	:WMEMory<r>:YScale? (see page 1132)	<r> ::= 1 to (# ref waveforms) in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V mV}

:WMEMory<r>:CLEar

N (see page 1232)

Command Syntax

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

The :WMEMory<r>:CLEar command clears the specified reference waveform location.

See Also

- Chapter 35, “:WMEMory<r> Commands,” starting on page 1123
- “:WMEMory<r>:SAVE” on page 1128
- “:WMEMory<r>:DISPlay” on page 1126

:WMEMory<r>:DISPlay

N (see page 1232)

Command Syntax :WMEMory<r>:DISPlay <on_off>

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

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

The :WMEMory<r>:DISPlay command turns the display of the specified reference waveform on or off.

There are two reference waveform locations, but only one reference waveform can be displayed at a time. That means, if :WMEMory1:DISPlay is ON, sending the :WMEMory2:DISPlay ON command will automatically set :WMEMory1:DISPlay OFF.

Query Syntax :WMEMory<r>:DISPlay?

The :WMEMory<r>:DISPlay? query returns the current display setting for the reference waveform.

Return Format <on_off><NL>

<on_off> ::= {1 | 0}

See Also

- Chapter 35, “:WMEMory<r> Commands,” starting on page 1123
- “:WMEMory<r>:CLEar” on page 1125
- “:WMEMory<r>:LABEL” on page 1127

:WMEMory<r>:LABel

N (see page 1232)

Command Syntax

```
:WMEMory<r>:LABel <string>
<r> ::= 1 to (# ref waveforms) in NR1 format
<string> ::= quoted ASCII string
```

NOTE

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

The :WMEMory<r>:LABel command sets the reference waveform label to the string that follows.

Setting a label for a reference waveform also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax

```
:WMEMory<r>:LABel?
```

The :WMEMory<r>:LABel? query returns the label associated with a particular reference waveform.

Return Format

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

See Also

- [Chapter 35, “:WMEMory<r> Commands,” starting on page 1123](#)
- [“:WMEMory<r>:DISPLAY” on page 1126](#)

:WMEMory<r>:SAVE**N** (see page 1232)**Command Syntax** `:WMEMory<r>:SAVE <source>`

```
<r> ::= 1 to (# ref waveforms) in NR1 format  
<source> ::= {CHANnel<n> | FUNCtion<m> | MATH<m>}  
<n> ::= 1 to (# analog channels) in NR1 format  
<m> ::= 1-4 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 35, “:WMEMory<r> Commands,” starting on page 1123
- “:WMEMory<r>:DISPLAY” on page 1126

:WMEMory<r>:SKEW

N (see page 1232)

Command Syntax :WMEMory<r>:SKEW <skew>

```
<r> ::= 1 to (# ref waveforms) in NR1 format
<skew> ::= time in seconds in NR3 format
```

The :WMEMory<r>:SKEW command sets the skew factor for the specified reference waveform.

Query Syntax :WMEMory<r>:SKEW?

The :WMEMory<r>:SKEW? query returns the current skew setting for the selected reference waveform.

Return Format <skew><NL>

```
<skew> ::= time in seconds in NR3 format
```

- See Also**
- Chapter 35, “:WMEMory<r> Commands,” starting on page 1123
 - “:WMEMory<r>:DISPlay” on page 1126
 - “:WMEMory<r>:YOFFset” on page 1130
 - “:WMEMory<r>:YRANGE” on page 1131
 - “:WMEMory<r>:YScale” on page 1132

:WMEMory<r>:YOFFset

N (see page 1232)

Command Syntax `:WMEMory<r>:YOFFset <offset> [<suffix>]`

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

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

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

The `:WMEMory<r>:YOFFset` command sets the value that is represented at center screen for the selected reference waveform.

The range of legal values varies with the value set by the `:WMEMory<r>:YRANge` or `:WMEMory<r>:YScale` commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are affected by the probe attenuation setting.

Query Syntax `:WMEMory<r>:YOFFset?`

The `:WMEMory<r>:YOFFset?` query returns the current offset value for the selected reference waveform.

Return Format `<offset><NL>`

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

See Also

- Chapter 35, “`:WMEMory<r>` Commands,” starting on page 1123
- “`:WMEMory<r>:DISPlay`” on page 1126
- “`:WMEMory<r>:YRANge`” on page 1131
- “`:WMEMory<r>:YScale`” on page 1132
- “`:WMEMory<r>:SKEW`” on page 1129

:WMEMory<r>:YRANge

N (see page 1232)

Command Syntax :WMEMory<r>:YRANge <range>[<suffix>]

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

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

<suffix> ::= {V | mV}

The :WMEMory<r>:YRANge command defines the full-scale vertical axis of the selected reference waveform.

Legal values for the range are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax :WMEMory<r>:YRANge?

The :WMEMory<r>:YRANge? query returns the current full-scale range setting for the specified reference waveform.

Return Format <range><NL>

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

See Also • [Chapter 35, “:WMEMory<r> Commands,” starting on page 1123](#)

- [":WMEMory<r>:DISPlay" on page 1126](#)
- [":WMEMory<r>:YOFFset" on page 1130](#)
- [":WMEMory<r>:SKEW" on page 1129](#)
- [":WMEMory<r>:YScale" on page 1132](#)

:WMEMory<r>:YSCale

N (see page 1232)

Command Syntax :WMEMory<r>:YSCale <scale>[<suffix>]

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

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

<suffix> ::= {V | mV}

The :WMEMory<r>:YSCale command sets the vertical scale, or units per division, of the selected reference waveform.

Legal values for the scale are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

Query Syntax :WMEMory<r>:YSCale?

The :WMEMory<r>:YSCale? query returns the current scale setting for the specified reference waveform.

Return Format <scale><NL>

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

- See Also**
- [Chapter 35, “:WMEMory<r> Commands,” starting on page 1123](#)
 - [":WMEMory<r>:DISPlay" on page 1126](#)
 - [":WMEMory<r>:YOFFset" on page 1130](#)
 - [":WMEMory<r>:YRANge" on page 1131](#)
 - [":WMEMory<r>:SKEW" on page 1129](#)

36

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 1232).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 270)	
ANALog<n>:COUpling	:CHANnel<n>:COUpling (see page 271)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 274)	
ANALog<n>:LABEL	:CHANnel<n>:LABEL (see page 275)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 276)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 277)	
ANALog<n>:PMODE	none	
ANALog<n>:RANGE	:CHANnel<n>:RANGE (see page 283)	
:CHANnel:ACTivity (see page 1140)	:ACTivity (see page 195)	
:CHANnel:LABEL (see page 1141)	:CHANnel<n>:LABEL (see page 275) or :DIGital<n>:LABEL (see page 310)	use CHANnel<n>:LABEL for analog channels and use DIGital<n>:LABEL for digital channels
:CHANnel:THreshold (see page 1142)	:POD<n>:THreshold (see page 555) or :DIGital<d>:THreshold (see page 313)	
:CHANnel2:SKEW (see page 1143)	:CHANnel<n>:PROBe:SKEW (see page 280)	



36 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel<n>:INPut (see page 1144)	:CHANnel<n>:IMPedance (see page 273)	
:CHANnel<n>:PMODe (see page 1145)	none	
:DISPlay:CONNect (see page 1146)	:DISPlay:VECTors (see page 328)	
:DISPlay:ORDer (see page 1147)	none	
:ERASe (see page 1148)	:DISPlay:CLEAR (see page 321)	
:EXTernal:PMODe (see page 1149)	none	
FUNCtion1, FUNCtion2	:FUNCtion Commands (see page 343)	ADD not included
:FUNCtion Commands	:FUNCtion2 Commands (see page 343)	:FUNCtion commands (with no <m> number) map to :FUNCtion2. This allows legacy programs to work without change.
:FUNCtion:GOFT:OPERation (see page 1150)	:FUNCtion1:OPERation (see page 364)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce1 (see page 1151)	:FUNCtion1:SOURce1 (see page 370)	GOFT maps to FUNCtion1.
:FUNCtion:GOFT:SOURce2 (see page 1152)	:FUNCtion1:SOURce2 (see page 372)	GOFT maps to FUNCtion1.
:FUNCtion:SOURce (see page 1153)	:FUNCtion:SOURce1 (see page 370)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNCtion:VIEW (see page 1154)	:FUNCtion:DISPlay (see page 353)	
:HARDcopy:DESTination (see page 1155)	:HARDcopy:FILEname (see page 1156)	
:HARDcopy:FILEname (see page 1156)	:RECall:FILEname (see page 627) :SAVE:FILEname (see page 627)	
:HARDcopy:GRAYscale (see page 1157)	:HARDcopy:PAlette (see page 389)	
:HARDcopy:IGColors (see page 1158)	:HARDcopy:INKSaver (see page 381)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:PDRiver (see page 1159)	:HARDcopy:APRinter (see page 378)	
:MEASure:LOWER (see page 1160)	:MEASure:DEFine:THResholds (see page 436)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see page 1161)	:MEASure:CLEar (see page 433)	
:MEASure:TDELta (see page 1162)	:MARKer:XDELta (see page 406)	
:MEASure:THResholds (see page 1163)	:MEASure:DEFine:THResholds (see page 436)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 1164)	:MEASure:XMAX (see page 492)	
:MEASure:TMIN (see page 1165)	:MEASure:XMIN (see page 493)	
:MEASure:TSTArt (see page 1166)	:MARKer:X1Position (see page 402)	
:MEASure:TSTOP (see page 1167)	:MARKer:X2Position (see page 404)	
:MEASure:TVOLt (see page 1168)	:MEASure:TVALue (see page 479)	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see page 1170)	:MEASure:DEFine:THResholds (see page 436)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 1171)	:MARKer:YDELta (see page 411)	
:MEASure:VSTArt (see page 1172)	:MARKer:Y1Position (see page 409)	
:MEASure:VSTOP (see page 1173)	:MARKer:Y2Position (see page 410)	
:MTEST:AMASK:{SAVE STORe} (see page 1174)	:SAVE:MASK[:STARt] (see page 645)	
:MTEST:AVERage (see page 1175)	:ACQuire:TYPE AVERage (see page 244)	
:MTEST:AVERage:COUNT (see page 1176)	:ACQuire:COUNt (see page 234)	
:MTEST:LOAD (see page 1177)	:RECall:MASK[:STARt] (see page 628)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:MTESt:RUMode (see page 1178)	:MTESt:RMODe (see page 536)	
:MTESt:RUMode:SOFailure (see page 1179)	:MTESt:RMODE:FACTion:STOP (see page 540)	
:MTESt:{STARt STOP} (see page 1180)	:RUN (see page 224) or :STOP (see page 228)	
:MTESt:TRIGger:SOURce (see page 1181)	:TRIGger Commands (see page 959)	There are various commands for setting the source with different types of triggers.
:PRINt? (see page 1182)	:DISPlay:DATA? (see page 322)	
:SAVE:IMAGE:AREA (see page 1184)	none	
:TIMEbase:DELay (see page 1188)	:TIMEbase:POSITION (see page 950) or :TIMEbase:WINDOW:POSITION (see page 956)	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 1185)	none	
:SBUS<n>:SPI:SOURce:DATA (see page 1186)	:SBUS<n>:SPI:SOURce:MOI (see page 780)	
:SYSTem:MENU (see page 1187)	:DISPlay:MENU (see page 325)	No change in behavior.
:TRIGger:THReShold (see page 1189)	:POD<n>:THReShold (see page 555) or :DIGItal<d>:THReShold (see page 313)	
:TRIGger:TV:TMoDe (see page 1190)	:TRIGger:TV:MODE (see page 1028)	

Discontinued Commands Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 4000 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 327)	
CHANnel:MATH	:FUNCTION:OPERation (see page 364)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see page 282)	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMal.
DISPlay:INVerse	none	
DISPlay:COLumn	none	
DISPlay:FREeze	none	
DISPlay:GRID	none	
DISPlay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSITION	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCTION:MOVE	none	
FUNCTION:PEAKs	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
:POWer:SIGNals:CYCLes	:POWer:SIGNals:CYCLes:HARMonics (see page 592) :POWer:SIGNals:CYCLes:QUALity (see page 593)	This command was separated into several other commands for specific types of power analysis.

36 Obsolete and Discontinued Commands

Discontinued Command	Current Command Equivalent	Comments
:POWer:SIGNals:DURation	:POWer:SIGNals:DURation:EF Ficiency (see page 594) :POWer:SIGNals:DURation:M ODulation (see page 595) :POWer:SIGNals:DURation:ON OFF:OFF (see page 596) :POWer:SIGNals:DURation:ON OFF:ON (see page 597) :POWer:SIGNals:DURation:RIP Ple (see page 598) :POWer:SIGNals:DURation:TR ANsient (see page 599)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:VMAXimum	:POWer:SIGNals:VMAXimum:I NRush (see page 602) :POWer:SIGNals:VMAXimum: ONOFF:OFF (see page 603) :POWer:SIGNals:VMAXimum: ONOFF:ON (see page 604)	This command was separated into several other commands for specific types of power analysis.
:POWer:SIGNals:VSteady	:POWer:SIGNals:VSteady:ON OFF:OFF (see page 605) :POWer:SIGNals:VSteady:ON OFF:ON (see page 606) :POWer:SIGNals:VSteady:TRA Nsient (see page 607)	This command was separated into several other commands for specific types of power analysis.
:POWer:SLEW:VALue	none	Slew rate values are now displayed using max and min measurements of a differentiate math function signal.
:PWRenable	none	The Power Event Enable Register does not exist in the 4000 X-Series oscilloscopes.
:PWRRegister[:EVENT]	none	The Power Event Event Register does not exist in the 4000 X-Series oscilloscopes.
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see page 189)	
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 1028)	

Discontinued Command	Current Command Equivalent	Comments
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
:TRIGger:USB:SOURce:DMINus	none	These USB trigger commands are superseded by commands that are available with the DSOX4USBFL (USB 2.0 Full/Low Speed) or DSOX4USBH (USB 2.0 High Speed) serial decode and triggering licenses. See " ":SBUS<n>:USB Commands " on page 809.
:TRIGger:USB:SOURce:DPLus	none	
:TRIGger:USB:SPEed	none	
:TRIGger:USB:TRIGger	none	
VAUToscale	none	

Discontinued Parameters Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 4000 X-Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:ACTivity

 (see [page 1232](#))

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 195](#)) 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

 (see [page 1232](#))

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 275](#)) or :DIGital<n>:LABEL command ([see page 310](#)).

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

 (see [page 1232](#))

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 555](#)) or :DIGItal<n>:THReShold command ([see page 313](#)).

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

 (see [page 1232](#))

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 280](#)) 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 268](#)

:CHANnel<n>:INPut

 (see [page 1232](#))

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 273](#)) 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 1232)

Command Syntax :CHANnel<n>:PMODe <pmod value>

<pmod 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

<pmod value><NL>

<pmod value> ::= {AUT | MAN}

:DISPlay:CONNect

 (see [page 1232](#))

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 328](#)) 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 328](#)

:DISPlay:ORDer

 (see page 1232)

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>:POSIon](#)" on page 311

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 41, “Programming Examples,”](#) starting on page 1241

:ERASe

 (see [page 1232](#))

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 321](#)) instead.

:EXTernal:PMODE

 (see [page 1232](#))

Command Syntax :EXTernal:PMODE <pmode value>

<pmode value> ::= {AUTo | MANual}

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE

The :EXTernal:PMODE command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax :EXTernal:PMODE?

The :EXTernal:PMODE? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format <pmode value><NL>

<pmode value> ::= {AUT | MAN}

:FUNCTION:GOFT:OPERation

 (see page 1232)

Command Syntax :FUNCTION:GOFT:OPERation <operation>

<operation> ::= {ADD | SUBTract | MULTiply}

The :FUNCTION:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to transform or filter functions (if available):

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTiply – Source1 * source2.

The :FUNCTION:GOFT:SOURce1 and :FUNCTION:GOFT:SOURce2 commands are used to select source1 and source2.

:FUNCTION:GOFT:SOURce1

 (see page 1232)

Command Syntax

```
:FUNCTION:GOFT:SOURce1 <value>
<value> ::= CHANnel<n>
<n> ::= {1 | 2 | 3 | 4} for 4ch models
<n> ::= {1 | 2} for 2ch models
```

The :FUNCTION:GOFT:SOURce1 command selects the first input channel for the g(t) source that can be used as the input to transform or filter functions (if available).

Query Syntax

```
:FUNCTION:GOFT:SOURce1?
```

The :FUNCTION:GOFT:SOURce1? query returns the current selection for the first input channel for the g(t) source.

Return Format

```
<value><NL>
<value> ::= CHAN<n>
<n> ::= {1 | 2 | 3 | 4} for the 4ch models
<n> ::= {1 | 2} for the 2ch models
```

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[:FUNCTION:GOFT:SOURce2](#)" on page 1152
- "[:FUNCTION:GOFT:OPERation](#)" on page 1150

:FUNCTION:GOFT:SOURce2

 (see page 1232)

Command Syntax :FUNCTION:GOFT:SOURce2 <value>

<value> ::= CHANnel<n>

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

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

The :FUNCTION:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to transform or filter functions (if available).

Query Syntax :FUNCTION:GOFT:SOURce2?

The :FUNCTION:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

Return Format <value><NL>

<value> ::= CHAN<n>

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

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

See Also • "[Introduction to :FUNCTION<m> Commands](#)" on page 346
• "[:FUNCTION:GOFT:SOURce1](#)" on page 1151
• "[:FUNCTION:GOFT:OPERation](#)" on page 1150

:FUNCTION:SOURce

 (see page 1232)

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 370) instead.

Query Syntax

```
:FUNCTION:SOURce?
```

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

Return Format

```
<value><NL>
<value> ::= {CHAN<n> | ADD | SUBT | MULT}
<n> ::= 1 to (# analog channels) in NR1 format
```

See Also

- "[Introduction to :FUNCTION<m> Commands](#)" on page 346
- "[:FUNCTION<m>:OPERation](#)" on page 364

:FUNCTION:VIEW

 (see [page 1232](#))

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 353](#)) 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 1232)

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 1156) 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 376

:HARDcopy:FILEname

 (see [page 1232](#))

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 638](#)) and :RECall:FILEname command (see [page 627](#)) 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 376

:HARDcopy:GRAYscale

 (see [page 1232](#))

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 389](#)) 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 376](#)

:HARDcopy:IGColors

 (see [page 1232](#))

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 381](#)) 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 376

:HARDcopy:PDRiver

 (see page 1232)

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 378](#)) 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 376

:MEASure:LOWER

 (see [page 1232](#))

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 436](#)) 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 428
 - "[":MEASure:THresholds](#)" on page 1163
 - "[":MEASure:UPPer](#)" on page 1170

:MEASure:SCRatch

 (see [page 1232](#))

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 433](#)) instead.

:MEASure:TDELta

 (see page 1232)

Query Syntax :MEASure:TDELta?

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

$$\text{Tdelta} = \text{Tstop} - \text{Tstart}$$

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 406) instead.

Return Format <value><NL>

<value> ::= time difference between start and stop markers in NR3 format

See Also

- "Introduction to :MARKer Commands" on page 398
- "Introduction to :MEASure Commands" on page 428
- ":MARKer:X1Position" on page 402
- ":MARKer:X2Position" on page 404
- ":MARKer:XDELta" on page 406
- ":MEASure:TSTArt" on page 1166
- ":MEASure:TSTOP" on page 1167

:MEASure:THResholds

 (see page 1232)

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 436) 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 428

- "[":MEASure:LOWER](#)" on page 1160

- "[":MEASure:UPPer](#)" on page 1170

:MEASure:TMAX

 (see page 1232)

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 492) 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 428
- ":MEASure:TMIN" on page 1165
- ":MEASure:XMAX" on page 492
- ":MEASure:XMIN" on page 493

:MEASure:TMIN

 (see page 1232)

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 493](#)) 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 428
- "[":MEASure:TMAX](#)" on page 1164
- "[":MEASure:XMAX](#)" on page 492
- "[":MEASure:XMIN](#)" on page 493

:MEASure:TSTArt

 (see [page 1232](#))

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 1234](#)). 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 402](#)) 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 398
- "[Introduction to :MEASure Commands](#)" on page 428
- "[":MARKer:X1Position](#)" on page 402
- "[":MARKer:X2Position](#)" on page 404
- "[":MARKer:XDELta](#)" on page 406
- "[":MEASure:TDELta](#)" on page 1162
- "[":MEASure:TSTOP](#)" on page 1167

:MEASure:TSTOp

 (see [page 1232](#))

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 1234](#)). 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 404](#)) 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 398
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MARKer:X1Position](#)" on page 402
- "[:MARKer:X2Position](#)" on page 404
- "[:MARKer:XDELta](#)" on page 406
- "[:MEASure:TDELta](#)" on page 1162
- "[:MEASure:TSTArt](#)" on page 1166

:MEASure:TVOLt

O (see [page 1232](#))

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 479](#)).

Return Format

<value><NL>

<value> ::= time in seconds of the specified voltage crossing
in NR3 format

:MEASure:UPPer

 (see [page 1232](#))

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 436](#)) 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 428
 - "[":MEASure:LOWER"](#) on page 1160
 - "[":MEASure:THresholds"](#) on page 1163

:MEASure:VDELta

 (see [page 1232](#))

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 411](#)) instead.

Return Format

`<value><NL>`

`<value>` ::= delta V value in NR1 format

See Also

- "[Introduction to :MARKer Commands](#)" on page 398
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MARKer:Y1Position](#)" on page 409
- "[:MARKer:Y2Position](#)" on page 410
- "[:MARKer:YDELta](#)" on page 411
- "[:MEASure:TDELta](#)" on page 1162
- "[:MEASure:TSTArt](#)" on page 1166

:MEASure:VSTArt

 (see [page 1232](#))

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 1234](#)). 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 409](#)) 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 398
- "[Introduction to :MEASure Commands](#)" on page 428
- "[":MARKer:Y1Position](#)" on page 409
- "[":MARKer:Y2Position](#)" on page 410
- "[":MARKer:YDELta](#)" on page 411
- "[":MARKer:X1Y1source](#)" on page 403
- "[":MEASure:SOURce](#)" on page 468
- "[":MEASure:TDELta](#)" on page 1162
- "[":MEASure:TSTARt](#)" on page 1166

:MEASure:VSTOp

O (see [page 1232](#))

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 1234](#)). 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 410](#)) 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 398
- "[Introduction to :MEASure Commands](#)" on page 428
- "[:MARKer:Y1Position](#)" on page 409
- "[:MARKer:Y2Position](#)" on page 410
- "[:MARKer:YDELta](#)" on page 411
- "[:MARKer:X2Y2source](#)" on page 405
- "[:MEASure:SOURce](#)" on page 468
- "[:MEASure:TDELta](#)" on page 1162
- "[:MEASure:TSTARt](#)" on page 1166

:MTEST:AMASK:{SAVE | STORe}



(see [page 1232](#))

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 645](#)) instead.

See Also

- "Introduction to :MTEST Commands" on [page 519](#)

:MTEST:AVERage

 (see page 1232)

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 244) 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 519
- ":MTEST:AVERage:COUNt" on page 1176

:MTEST:AVERage:COUNt

 (see [page 1232](#))

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 234](#)) 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 519](#)
- "[:MTEST:AVERage](#)" on [page 1175](#)

:MTEST:LOAD



(see [page 1232](#))

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 628](#)) instead.

See Also

- "[Introduction to :MTEST Commands](#)" on page 519
- "[":MTEST:AMASK:{SAVE | STORe}](#)" on page 1174

:MTEST:RUMode

 (see [page 1232](#))

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 536](#)) 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 519](#)
- "[:MTEST:RUMode:SOFailure](#)" on [page 1179](#)

:MTEST:RUMode:SOFailure

 (see page 1232)

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 540) 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 519
- ":MTEST:RUMode" on page 1178

:MTEST:{STARt | STOP}



(see [page 1232](#))

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 224](#)) and :STOP command (see [page 228](#)) instead.

See Also

- "Introduction to :MTEST Commands" on [page 519](#)

:MTEST:TRIGger:SOURce (see [page 1232](#))**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 959](#)) 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 519](#)

:PRINt?

 (see page 1232)

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 322](#)) 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 194
- "[Introduction to :HARDcopy Commands](#)" on page 376
- "[:HARDcopy:FACTors](#)" on page 379
- "[:HARDcopy:GRAYscale](#)" on page 1157
- "[:DISPlay:DATA](#)" on page 322

:SAVE:IMAGe:AREA

 (see page 1232)

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 635

- "[":SAVE:IMAGe\[:STARt\]](#)" on page 639
- "[":SAVE:IMAGe:FACTors](#)" on page 640
- "[":SAVE:IMAGe:FORMat](#)" on page 641
- "[":SAVE:IMAGe:INKSaver](#)" on page 642
- "[":SAVE:IMAGe:PALETTE](#)" on page 643

:SBUS<n>:LIN:SIGNAl:DEFinition

 (see page 1232)

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 959
- ":TRIGger:MODE" on page 968
- ":SBUS<n>:LIN:SIGNAl:BAUDrate" on page 754
- ":SBUS<n>:LIN:SOURce" on page 755

:SBUS<n>:SPI:SOURce:DATA

 (see page 1232)

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 959
- "[:SBUS<n>:SPI:SOURce:MOSI](#)" on page 780
- "[:SBUS<n>:SPI:SOURce:MISO](#)" on page 779
- "[:SBUS<n>:SPI:SOURce:CLOCK](#)" on page 777
- "[:SBUS<n>:SPI:SOURce:FRAME](#)" on page 778
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:DATA](#)" on page 781
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:DATA](#)" on page 783
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MISO:WIDTh](#)" on page 782
- "[:SBUS<n>:SPI:TRIGger:PATTERn:MOSI:WIDTh](#)" on page 784

:SYSTem:MENU (see page 1232)**Command Syntax** :SYSTem:MENU <menu>

<menu> ::= {MASK | MEASure | SEGmented | LISTer | POWer}

The :SYSTem:MENU command changes the front panel softkey menu.

:TIMEbase:DELay

 (see [page 1232](#))

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 953](#)).

NOTE

The :TIMEbase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POSItion command (see [page 950](#)) 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 41, “Programming Examples,”](#) starting on [page 1241](#)

:TRIGger:THreshold

O (see page 1232)

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 555), :DIGItal<d>:THreshold command (see page 313), or :TRIGger[:EDGE]:LEVel command (see page 986).

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:TMODE

 (see [page 1232](#))

Command Syntax :TRIGger:TV:TMODE <mode>

```
<mode> ::= {FIEld1 | FIEld2 | AFIEld | 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 1031](#)).

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

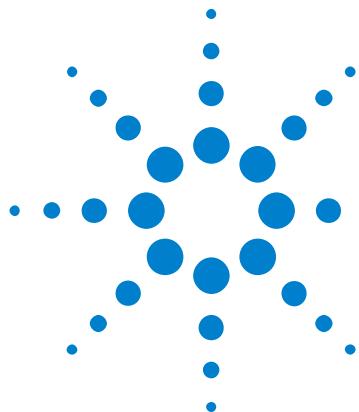
The :TRIGger:TV:TMODE command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see [page 1028](#)) instead.

Query Syntax :TRIGger:TV:TMODE?

The :TRIGger:TV:TMODE? query returns the TV trigger mode.

Return Format <value><NL>

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
             | LALT | LVER}
```



37 Error Messages

-440, Query UNTERMINATED after indefinite response

-430, Query DEADLOCKED

-420, Query UNTERMINATED

-410, Query INTERRUPTED

-400, Query error

-340, Calibration failed

-330, Self-test failed

-321, Out of memory

-320, Storage fault

-315, Configuration memory lost



-314, Save/recall memory lost

-313, Calibration memory lost

-311, Memory error

-310, System error

-300, Device specific error

-278, Macro header not found

-277, Macro redefinition not allowed

-276, Macro recursion error

-273, Illegal macro label

-272, Macro execution error

-258, Media protected

-257, File name error

-256, File name not found

-255, Directory full

-254, Media full

-253, Corrupt media

-252, Missing media

-251, Missing mass storage

-250, Mass storage error

-241, Hardware missing

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

-240, Hardware error

-231, Data questionable

-230, Data corrupt or stale

-224, Illegal parameter value

-223, Too much data

-222, Data out of range

-221, Settings conflict

-220, Parameter error

-200, Execution error

-183, Invalid inside macro definition

-181, Invalid outside macro definition

-178, Expression data not allowed

-171, Invalid expression

-170, Expression error

-168, Block data not allowed

-161, Invalid block data

-158, String data not allowed

-151, Invalid string data

-150, String data error

-148, Character data not allowed

-138, Suffix not allowed

-134, Suffix too long

-131, Invalid suffix

-128, Numeric data not allowed

-124, Too many digits

-123, Exponent too large

-121, Invalid character in number

-120, Numeric data error

-114, Header suffix out of range

-113, Undefined header

-112, Program mnemonic too long

-109, Missing parameter

-108, Parameter not allowed

-105, GET not allowed

-104, Data type error

-103, Invalid separator

-102, Syntax error

-101, Invalid character

-100, Command error

+10, Software Fault Occurred

+100, File Exists

+101, End-Of-File Found

+102, Read Error

+103, Write Error**+104, Illegal Operation****+105, Print Canceled****+106, Print Initialization Failed****+107, Invalid Trace File****+108, Compression Error****+109, No Data For Operation**

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPLAY:DATA? query, but there may be no image stored.

+112, Unknown File Type**+113, Directory Not Supported**

37 Error Messages

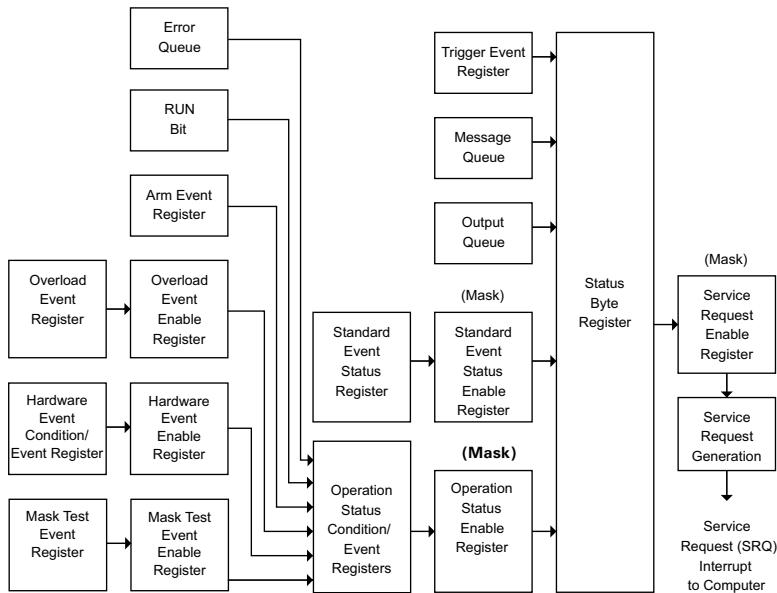
38 Status Reporting

- Status Reporting Data Structures [1201](#)
- Status Byte Register (STB) [1203](#)
- Service Request Enable Register (SRE) [1205](#)
- Trigger Event Register (TER) [1206](#)
- Output Queue [1207](#)
- Message Queue [1208](#)
- (Standard) Event Status Register (ESR) [1209](#)
- (Standard) Event Status Enable Register (ESE) [1210](#)
- Error Queue [1211](#)
- Operation Status Event Register (:OPERegister[:EVENT]) [1212](#)
- Operation Status Condition Register (:OPERegister:CONDITION) [1213](#)
- Arm Event Register (AER) [1214](#)
- Overload Event Register (:OVLRegister) [1215](#)
- Hardware Event Event Register (:HWERegister[:EVENT]) [1216](#)
- Hardware Event Condition Register (:HWERegister:CONDITION) [1217](#)
- Mask Test Event Event Register (:MTERegister[:EVENT]) [1218](#)
- Clearing Registers and Queues [1219](#)
- Status Reporting Decision Chart [1220](#)

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.





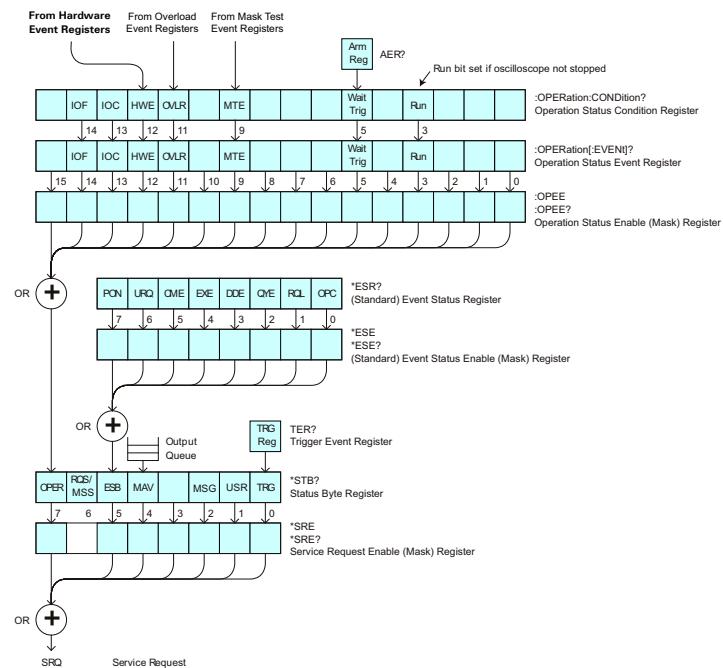
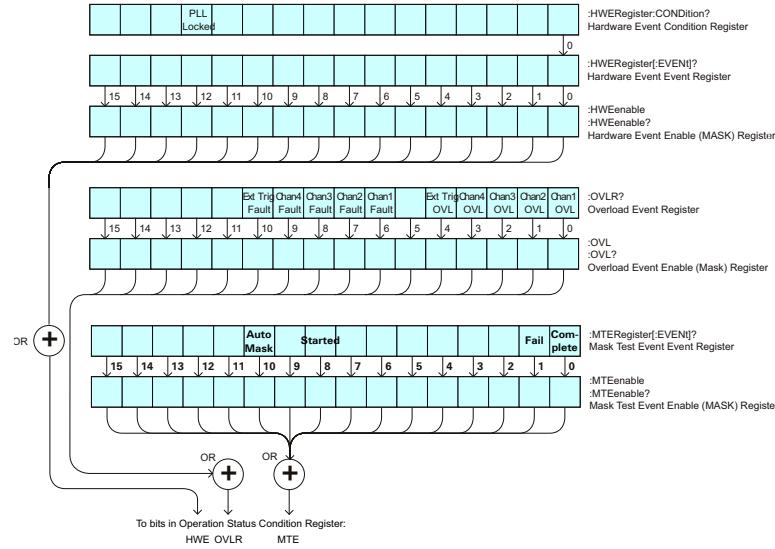
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the 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 70](#)
- [Table 68](#)
- [Table 78](#)
- [Table 79](#)
- [Table 81](#)
- [Table 73](#)
- [Table 74](#)
- [Table 76](#)

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.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.
HWE bit	bit 12	Comes from the Hardware Event Registers.
IOC bit	bit 13	Is set when the IO operation completes.
IOF bit	bit 14	Is set when the IO operation fails.

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.
MTE bit	bit 9	Comes from the Mask Test Event Registers.
OVLR bit	bit 11	Is set whenever a 50Ω input overload occurs.
HWE bit	bit 12	Comes from the Hardware Event Registers.
IOC bit	bit 13	Is set when the IO operation completes.
IOF bit	bit 14	Is set when the IO operation fails.

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.

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

This register hosts the PLL LOCKED bit (bit 12).

- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

Hardware Event Condition Register (:HWERegister:CONDITION)

This register hosts the PLL LOCKED bit (bit 12).

- The :HWERegister:CONDITION? query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

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

The Mask Test Event Event Register register hosts these bits:

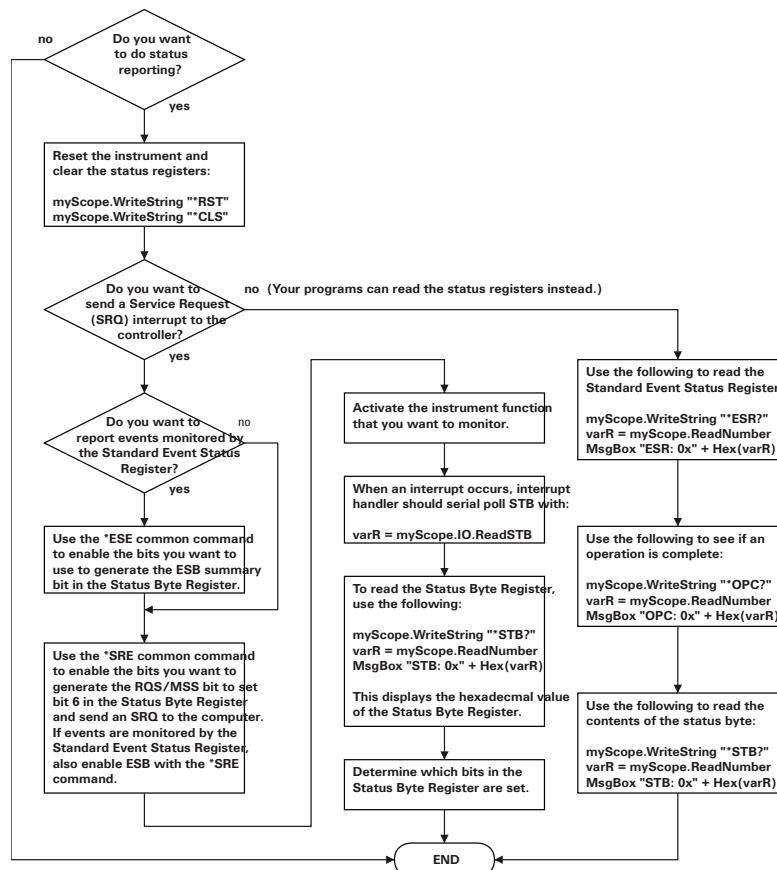
Name	Location	Description
Complete	bit 0	Is set when the mask test is complete.
Fail	bit 1	Is set when there is a mask test failure.
Started	bit 8	Is set when mask testing is started.
Auto Mask	bit 10	Is set when auto mask creation is completed.

The :MTERegister[:EVENT]? query returns the value of, and clears, the Mask Test Event Event Register.

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



39

Synchronizing Acquisitions

- [Synchronization in the Programming Flow](#) **1222**
- [Blocking Synchronization](#) **1223**
- [Polling Synchronization With Timeout](#) **1224**
- [Synchronizing with a Single-Shot Device Under Test \(DUT\)](#) **1226**
- [Synchronization with an Averaging Acquisition](#) **1228**

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 1222](#)).
- 2 Acquire a waveform (see [page 1222](#)).
- 3 Retrieve results (see [page 1222](#)).

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 <i>will</i> trigger based on the oscilloscope setup and device under test.	You know the oscilloscope <i>may or may not</i> 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 1223.	See " Polling Synchronization With Timeout " on page 1224.

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 1223 and "Polling Synchronization With Timeout" on page 1224 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 1224 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 ":OPERRegister: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

```

39 Synchronizing Acquisitions

40

More About Oscilloscope Commands

Command Classifications [1232](#)

Valid Command/Query Strings [1233](#)

Query Return Values [1239](#)

All Oscilloscope Commands Are Sequential [1240](#)



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 1232
- "Non-Core Commands" on page 1232
- "Obsolete Commands" on page 1232

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:

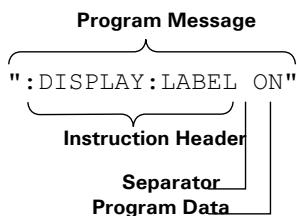
- [Chapter 36](#), "Obsolete and Discontinued Commands," starting on page 1133

Valid Command/Query Strings

- "Program Message Syntax" on page 1233
- "Duplicate Mnemonics" on page 1237
- "Tree Traversal Rules and Multiple Commands" on page 1237

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 1234), 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 1235](#)
- ["Compound Command Headers" on page 1235](#)
- ["Common Command Headers" on page 1235](#)

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 1236](#) 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 CHANel2.

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><terminator>
```

For example:

```
myScope.WriteString ":TIMEbase:RANGE 0.5;POSITION 0"
```

NOTE

The colon between TIMEbase and RANGE is necessary because TIMEbase:RANGE is a compound command. The semicolon between the RANGE command and the POSITION command is the required program message unit separator. The POSITION command does not need TIMEbase preceding it because the TIMEbase:RANGE command sets the parser to the TIMEbase node in the tree.

Example 2:

Program Message

Terminator Sets Parser Back to Root

NOTE

```
myScope.WriteString ":TIMEbase:REFERENCE CENTER;POSITION 0.00001"
```

or

```
myScope.WriteString ":TIMEbase:REFERENCE CENTER"  
myScope.WriteString ":TIMEbase:POSITION 0.00001"
```

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

41 Programming Examples

VISA COM Examples [1242](#)

VISA Examples [1275](#)

SICL Examples [1322](#)

SCPI.NET Examples [1342](#)

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.



Agilent Technologies

1241

VISA COM Examples

- "VISA COM Example in Visual Basic" on page 1242
- "VISA COM Example in C#" on page 1251
- "VISA COM Example in Visual Basic .NET" on page 1260
- "VISA COM Example in Python" on page 1268

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()

```

41 Programming Examples

```
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:VAMPplitude"
varQueryResult = DoQueryNumber(":MEASure:VAMPplitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
FormatNumber(varQueryResult, 4) + " V"

' Download the screen image.

```

```

' -----
' Get screen image.
DoCommand ":HARDcopy:INKSaver OFF"
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPLAY:DATA? PNG, COLOR")

' Save screen image to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If

Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , byteData      ' Write data.
Close hFile      ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
       " bytes) written to " + strPath

' Download waveform data.
' -----

' Set the waveform points mode.
DoCommand ":WAVEFORM:POINTS:MODE RAW"
Debug.Print "Waveform points mode: " + _
           DoQueryString(":WAVEFORM:POINTS:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
           DoQueryString(":WAVEFORM:POINTS?")

' Set the waveform source.
DoCommand ":WAVEFORM:SOURCe CHANnel1"
Debug.Print "Waveform source: " + _
           DoQueryString(":WAVEFORM:SOURCe?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVEFORM:FORMAT BYTE"
Debug.Print "Waveform format: " + _
           DoQueryString(":WAVEFORM:FORMAT?")

' Display the waveform settings:
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

```

```

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 4 Then
    Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
    Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
varQueryResult = DoQueryIEEEBlock_UI1(":WAVeform:DATA?")

```

41 Programming Examples

```
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.WriteLineEEBlock 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

```

41 Programming Examples

```
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
            {

```

41 Programming Examples

```
    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

```

41 Programming Examples

```
).
myScope.DoCommand(":ACQuire:TYPE NORMAl");
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQuire:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] DataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.stp";
DataArray = File.ReadAllBytes(strPath);
nBytesWritten = DataArray.Length;

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGItize.
myScope.DoCommand(":DIGItize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1");
    Console.WriteLine("Measure source: {0}",
        myScope.DoQueryString(":MEASure:SOURce?"));

    double fResult;
    myScope.DoCommand(":MEASure:FREQuency");
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    myScope.DoCommand(":MEASure:VAMplitude");
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?");
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

    // Download the screen image.
    // -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF");

    // Get the screen data.
    ResultsArray =
        myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR");
    nLength = ResultsArray.Length;

    // Store the screen data to a file.
```

```

strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// -----
// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"));

// Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINTS?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMAT BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMAT?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCII");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
    Console.WriteLine("Acquire type: NORMAL");
}
else if (fType == 1.0)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
    Console.WriteLine("Acquire type: RAMP");
}

```

41 Programming Examples

```
{  
    Console.WriteLine("Acquire type: AVERage");  
}  
else if (fType == 3.0)  
{  
    Console.WriteLine("Acquire type: HRESolution");  
}  
  
double fPoints = fResultsArray[2];  
Console.WriteLine("Waveform points: {0:e}", fPoints);  
  
double fCount = fResultsArray[3];  
Console.WriteLine("Waveform average count: {0:e}", fCount);  
  
double fXincrement = fResultsArray[4];  
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);  
  
double fXorigin = fResultsArray[5];  
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);  
  
double fXreference = fResultsArray[6];  
Console.WriteLine("Waveform X reference: {0:e}", fXreference);  
  
double fYincrement = fResultsArray[7];  
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);  
  
double fYorigin = fResultsArray[8];  
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);  
  
double fYreference = fResultsArray[9];  
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);  
  
// Read waveform data.  
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?");  
nLength = ResultsArray.Length;  
Console.WriteLine("Number of data values: {0}", nLength);  
  
// Set up output file:  
strPath = "c:\\scope\\data\\waveform_data.csv";  
if (File.Exists(strPath)) File.Delete(strPath);  
  
// Open file for output.  
StreamWriter writer = File.CreateText(strPath);  
  
// Output waveform data in CSV format.  
for (int i = 0; i < nLength - 1; i++)  
    writer.WriteLine("{0:f9}, {1:f6}",  
                    fXorigin + ((float)i * fXincrement),  
                    (((float)ResultsArray[i] - fYreference)  
                     * fYincrement) + fYorigin);  
  
// Close output file.  
writer.Close();  
Console.WriteLine("Waveform format BYTE data written to {0}",  
                 strPath);  
}  
}
```

```

class VisaComInstrument
{
    private ResourceManagerClass m_ResourceManager;
    private FormattedIO488Class m_IoObject;
    private string m_strVisaAddress;

    // Constructor.
    public VisaComInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA COM IO object.
        OpenIo();

        // Clear the interface.
        m_IoObject.IO.Clear();
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        m_IoObject.WriteString(strCommand, true);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public void DoCommandIEEEBlock(string strCommand,
                                  byte[] DataArray)
    {
        // Send the command to the device.
        m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public string DoQueryString(string strQuery)
    {
        // Send the query.
        m_IoObject.WriteString(strQuery, true);

        // Get the result string.
        string strResults;
        strResults = m_IoObject.ReadString();

        // Check for inst errors.
        CheckInstrumentErrors(strQuery);

        // Return results string.
        return strResults;
    }

    public double DoQueryNumber(string strQuery)
    {

```

41 Programming Examples

```
// 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")

```

41 Programming Examples

```
Console.WriteLine("Trigger edge source: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()      ' Results array.
Dim nLength As Integer         ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMEbase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMEbase:SCALe?"))

myScope.DoCommand(":TIMEbase:POSItion 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMEbase:POSItion?"))

' Set the acquisition type (NORMAL, PEAK, AVERage, or HRESolution)

myScope.DoCommand(":ACQuire:TYPE NORMAL")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim DataArray As Byte()
Dim nBytesWritten As Integer
```

```

' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
DataArray = File.ReadAllBytes(strPath)
nBytesWritten = DataArray.Length

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGITIZE.
myScope.DoCommand(":DIGITIZE CHANnel1")

End Sub

' Analyze the captured waveform.
' -----
Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte()      ' Results array.
    Dim nLength As Integer         ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
                      myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    myScope.DoCommand(":MEASure:VAMplitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?")
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

    ' Download the screen image.
    ' -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF")

    ' Get the screen data.
    ResultsArray = myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR")
    nLength = ResultsArray.Length

    ' Store the screen data to a file.
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Screen image ({0} bytes) written to {1}", _
                      nLength, strPath)

```

41 Programming Examples

```
' Download waveform data.  
' -----  
  
' Set the waveform points mode.  
myScope.DoCommand(":WAVEform:POINTS:MODE RAW")  
Console.WriteLine("Waveform points mode: {0}", _  
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"))  
  
' Get the number of waveform points available.  
Console.WriteLine("Waveform points available: {0}", _  
    myScope.DoQueryString(":WAVEform:POINTS?"))  
  
' Set the waveform source.  
myScope.DoCommand(":WAVEform:SOURce CHANnel1")  
Console.WriteLine("Waveform source: {0}", _  
    myScope.DoQueryString(":WAVEform:SOURce?"))  
  
' Choose the format of the data returned (WORD, BYTE, ASCII):  
myScope.DoCommand(":WAVEform:FORMAT BYTE")  
Console.WriteLine("Waveform format: {0}", _  
    myScope.DoQueryString(":WAVEform:FORMAT?"))  
  
' Display the waveform settings:  
Dim fResultsArray As Double()  
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")  
  
Dim fFormat As Double = fResultsArray(0)  
If fFormat = 0 Then  
    Console.WriteLine("Waveform format: BYTE")  
ElseIf fFormat = 1 Then  
    Console.WriteLine("Waveform format: WORD")  
ElseIf fFormat = 2 Then  
    Console.WriteLine("Waveform format: ASCII")  
End If  
  
Dim fType As Double = fResultsArray(1)  
If fType = 0 Then  
    Console.WriteLine("Acquire type: NORMAL")  
ElseIf fType = 1 Then  
    Console.WriteLine("Acquire type: PEAK")  
ElseIf fType = 2 Then  
    Console.WriteLine("Acquire type: AVERAGE")  
ElseIf fType = 3 Then  
    Console.WriteLine("Acquire type: HRESolution")  
End If  
  
Dim fPoints As Double = fResultsArray(2)  
Console.WriteLine("Waveform points: {0:e}", fPoints)  
  
Dim fCount As Double = fResultsArray(3)  
Console.WriteLine("Waveform average count: {0:e}", fCount)  
  
Dim fxIncrement As Double = fResultsArray(4)  
Console.WriteLine("Waveform X increment: {0:e}", fxIncrement)  
  
Dim fxOrigin As Double = fResultsArray(5)  
Console.WriteLine("Waveform X origin: {0:e}", fxOrigin)
```

```

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?")
nLength = ResultsArray.Length
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
                    fXorigin + (CSng(index) * fXincrement), _
                    ((CSng(ResultsArray(index)) - fYreference) _ 
                     * fYincrement) + fYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
                  strPath)

End Sub

End Class

Class VisaComInstrument
    Private m_ResourceManager As ResourceManagerClass
    Private m_IoObject As FormattedIO488Class
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)

        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA COM IO object.
    End Sub
End Class

```

41 Programming Examples

```
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
```

```

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}': {1}, _", _
                    strCommand)
                bFirstError = False
            End If
            Console.Write(strInstrumentError)
        End If
    Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

```

```
Private Sub OpenIo()
    m_ResourceManager = New ResourceManagerClass()
    m_IoObject = New FormattedIO488Class()

    ' Open the default VISA COM IO object.
    Try
        m_IoObject.IO =
            DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
                AccessMode.NO_LOCK, 0, ""), IMessage)
    Catch e As Exception
        Console.WriteLine("An error occurred: {0}", e.Message)
    End Try
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    m_IoObject.IO.Timeout = nSeconds * 1000
End Sub

Public Sub Close()
    Try
        m_IoObject.IO.Close()
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_IoObject)
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_ResourceManager)
    Catch
    End Try
    End Sub
End Class
End Namespace
```

VISA COM Example in Python

You can use the Python programming language with the "comtypes" package to control Agilent oscilloscopes.

The Python language and "comtypes" package can be downloaded from the web at "<http://www.python.org/>" and "<http://starship.python.net/crew/theller/comtypes/>", respectively.

To run this example with Python and "comtypes":

- 1 Cut-and-paste the code that follows into a file named "example.py".
- 2 Edit the program to use the VISA address of your oscilloscope.
- 3 If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```

python example.py

#
# Agilent VISA COM Example in Python using "comtypes"
# ****
# This program illustrates a few commonly used programming
# features of your Agilent oscilloscope.
# ****

# Import Python modules.
# -----
import string
import time
import sys
import array

from comtypes.client import GetModule
from comtypes.client import CreateObject

# Run GetModule once to generate comtypes.gen.VisaComLib.
if not hasattr(sys, "frozen"):
    GetModule("C:\Program Files (x86)\IVI Foundation\VISA\VisaCom\
GlobMgr.dll")

import comtypes.gen.VisaComLib as VisaComLib

# Global variables (booleans: 0 = False, 1 = True).
# ----

# =====
# Initialize:
# =====
def initialize():
    # Get and display the device's *IDN? string.
    idn_string = do_query_string("*IDN?")
    print "Identification string '%s'" % idn_string

    # Clear status and load the default setup.
    do_command("*CLS")
    do_command("*RST")

# =====
# Capture:
# =====
def capture():

    # Use auto-scale to automatically set up oscilloscope.
    print "Autoscale."
    do_command(":AUToscale")

    # Set trigger mode.
    do_command(":TRIGger:MODE EDGE")
    qresult = do_query_string(":TRIGger:MODE?")
    print "Trigger mode: %s" % qresult

```

41 Programming Examples

```
# Set EDGE trigger parameters.
do_command(":TRIGger:EDGE:SOURCe CHANnel1")
qresult = do_query_string(":TRIGger:EDGE:SOURce?")
print "Trigger edge source: %s" % qresult

do_command(":TRIGger:EDGE:LEVel 1.5")
qresult = do_query_string(":TRIGger:EDGE:LEVel?")
print "Trigger edge level: %s" % qresult

do_command(":TRIGger:EDGE:SLOPe POSitive")
qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
print "Trigger edge slope: %s" % qresult

# Save oscilloscope setup.
setup_bytes = do_query_ieee_block(":SYSTem:SETUp?")
nLength = len(setup_bytes)
f = open("c:\scope\config\setup.stp", "wb")
f.write(bytarray(setup_bytes))
f.close()
print "Setup bytes saved: %d" % nLength

# Change oscilloscope settings with individual commands:

# Set vertical scale and offset.
do_command(":CHANnel1:SCALe 0.05")
qresult = do_query_number(":CHANnel1:SCALe?")
print "Channel 1 vertical scale: %f" % qresult

do_command(":CHANnel1:OFFSet -1.5")
qresult = do_query_number(":CHANnel1:OFFSet?")
print "Channel 1 offset: %f" % qresult

# Set horizontal scale and offset.
do_command(":TIMEbase:SCALe 0.0002")
qresult = do_query_string(":TIMEbase:SCALe?")
print "Timebase scale: %s" % qresult

do_command(":TIMEbase:POSIon 0.0")
qresult = do_query_string(":TIMEbase:POSIon?")
print "Timebase position: %s" % qresult

# Set the acquisition type.
do_command(":ACQuire:TYPE NORMAL")
qresult = do_query_string(":ACQuire:TYPE?")
print "Acquire type: %s" % qresult

# Or, configure by loading a previously saved setup.
f = open("c:\scope\config\setup.stp", "rb")
setup_bytes = f.read()
f.close()
do_command_ieee_block(":SYSTem:SETUp", array.array('B', setup_bytes))
print "Setup bytes restored: %d" % len(setup_bytes)

# Capture an acquisition using :DIGitize.
do_command(":DIGitize CHANnel1")
```

```

# =====
# Analyze:
# =====
def analyze():

    # Make measurements.
    # -----
    do_command(":MEASure:SOURce CHANnel1")
    qresult = do_query_string(":MEASure:SOURce?")
    print "Measure source: %s" % qresult

    do_command(":MEASure:FREQuency")
    qresult = do_query_string(":MEASure:FREQuency?")
    print "Measured frequency on channel 1: %s" % qresult

    do_command(":MEASure:VAMPplitude")
    qresult = do_query_string(":MEASure:VAMPplitude?")
    print "Measured vertical amplitude on channel 1: %s" % qresult

    # Download the screen image.
    # -----
    do_command(":HARDcopy:INKSaver OFF")

    image_bytes = do_query_ieee_block(":DISPLAY:DATA? PNG, COLOR")
    nLength = len(image_bytes)
    f = open("c:\scope\data\screen.bmp", "wb")
    f.write(bytarray(image_bytes))
    f.close()
    print "Screen image written to c:\scope\data\screen.bmp."

    # 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",

```

41 Programming Examples

```
    4 : "ASCIi",
}
acq_type_dict = {
    0 : "NORMal",
    1 : "PEAK",
    2 : "AVERage",
    3 : "HRESolution",
}

(
    wav_form,
    acq_type,
    wfmpts,
    avgcnt,
    x_increment,
    x_origin,
    x_reference,
    y_increment,
    y_origin,
    y_reference
) = do_query_numbers(":WAVEform:PREamble?")

print "Waveform format: %s" % wav_form_dict[wav_form]
print "Acquire type: %s" % acq_type_dict[acq_type]
print "Waveform points desired: %d" % wfmpts
print "Waveform average count: %d" % avgcnt
print "Waveform X increment: %1.12f" % x_increment
print "Waveform X origin: %1.9f" % x_origin
print "Waveform X reference: %d" % x_reference # Always 0.
print "Waveform Y increment: %f" % y_increment
print "Waveform Y origin: %f" % y_origin
print "Waveform Y reference: %d" % y_reference # Always 125.

# Get numeric values for later calculations.
x_increment = do_query_number(":WAVEform:XINCrement?")
x_origin = do_query_number(":WAVEform:XORigin?")
y_increment = do_query_number(":WAVEform:YINCrement?")
y_origin = do_query_number(":WAVEform:YORigin?")
y_reference = do_query_number(":WAVEform:YREFerence?")

# Get the waveform data.
data_bytes = do_query_ieee_block(":WAVEform:DATA?")
nLength = len(data_bytes)
print "Number of data values: %d" % nLength

# Open file for output.
strPath = "c:\scope\data\waveform_data.csv"
f = open(strPath, "w")

# Output waveform data in CSV format.
for i in xrange(0, nLength - 1):
    time_val = x_origin + (i * x_increment)
    voltage = (data_bytes[i] - y_reference) * y_increment + y_origin
    f.write("%E, %f\n" % (time_val, voltage))

# Close output file.
f.close()
```

```
print "Waveform format BYTE data written to %s." % strPath

# =====
# Send a command and check for errors:
# =====
def do_command(command):
    myScope.WriteString("%s" % command, True)
    check_instrument_errors(command)

# =====
# Send a command and check for errors:
# =====
def do_command_ieee_block(command, data):
    myScope.WriteIEEEBlock(command, data, True)
    check_instrument_errors(command)

# =====
# Send a query, check for errors, return string:
# =====
def do_query_string(query):
    myScope.WriteString("%s" % query, True)
    result = myScope.ReadString()
    check_instrument_errors(query)
    return result

# =====
# Send a query, check for errors, return string:
# =====
def do_query_ieee_block(query):
    myScope.WriteString("%s" % query, True)
    result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_UI1, \
        False, True)
    check_instrument_errors(query)
    return result

# =====
# Send a query, check for errors, return values:
# =====
def do_query_number(query):
    myScope.WriteString("%s" % query, True)
    result = myScope.ReadNumber(VisaComLib.ASCIIType_R8, True)
    check_instrument_errors(query)
    return result

# =====
# Send a query, check for errors, return values:
# =====
def do_query_numbers(query):
    myScope.WriteString("%s" % query, True)
    result = myScope.ReadList(VisaComLib.ASCIIType_R8, ",;")
    check_instrument_errors(query)
```

41 Programming Examples

```
        return result

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

    while True:
        myScope.WriteString(":SYSTem:ERRor?", True)
        error_string = myScope.ReadString()
        if error_string: # If there is an error string value.

            if error_string.find("+0,", 0, 3) == -1: # Not "No error".
                print "ERROR: %s, command: '%s'" % (error_string, command)
                print "Exited because of error."
                sys.exit(1)

            else: # "No error"
                break

        else: # :SYSTem:ERRor? should always return string.
            print "ERROR: :SYSTem:ERRor? returned nothing, command: '%s' \
                  % command
            print "Exited because of error."
            sys.exit(1)

# =====
# Main program:
# =====
rm = CreateObject("VISA.GlobalRM", \
    interface=VisaComLib.IResourceManager)
myScope = CreateObject("VISA.BasicFormattedIO", \
    interface=VisaComLib.IFormattedIO488)
myScope.IO = \
    rm.Open("TCPIPO::a-mx3104a-90028.cos.agilent.com::inst0::INSTR")

# Clear the interface.
myScope.IO.Clear
print "Interface cleared."

# Set the Timeout to 15 seconds.
myScope.IO.Timeout = 15000 # 15 seconds.
print "Timeout set to 15000 milliseconds."

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program"
```

VISA Examples

- "VISA Example in C" on page 1275
- "VISA Example in Visual Basic" on page 1284
- "VISA Example in C#" on page 1294
- "VISA Example in Visual Basic .NET" on page 1305
- "VISA Example in Python" on page 1315

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
```

41 Programming Examples

```
* -----
* 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. */
}

```

41 Programming Examples

```
num_bytes = do_query_ieeeblock(":SYSTem:SETrun?");
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:POSIon 0.0");
do_query_string(":TIMEbase:POSIon?");
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:SETrun", 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:VAMPplitude");
    do_query_number(":MEASure:VAMPplitude?");
    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?");
}

```

41 Programming Examples

```
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;

```

41 Programming Examples

```
{  
    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)

```

41 Programming Examples

```
    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:POSIon 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSIon?")

' 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)

```

41 Programming Examples

```
' 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:VAMPplitude"  
dblQueryResult = DoQueryNumber(":MEASure:VAMPplitude?")  
MsgBox "Vertical amplitude:" + vbCrLf + _  
    FormatNumber(dblQueryResult, 4) + " V"  
  
' Download the screen image.  
' -----  
DoCommand ":HARDcopy:INKSaver OFF"  
  
' Get screen image.  
Dim lngBlockSize As Long  
lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPLAY:DATA? PNG, COLOR")  
Debug.Print "Screen image bytes: " + CStr(lngBlockSize)  
  
' Save screen image to a file:  
Dim strPath As String  
strPath = "c:\scope\data\screen.png"  
If Len(Dir(strPath)) Then  
    Kill strPath      ' Remove file if it exists.  
End If  
Dim hFile As Long  
hFile = FreeFile  
Open strPath For Binary Access Write Lock Write As hFile  
Dim lngI As Long  
For lngI = 0 To lngBlockSize - 1  
    Put hFile, , byteArray(lngI)      ' Write data.  
Next lngI  
Close hFile      ' Close file.  
MsgBox "Screen image written to " + strPath  
  
' Download waveform data.
```

```

' -----
' Set the waveform points mode.
DoCommand ":WAVeform:POINTs:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINTs:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINTs?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMAT BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMAT?")

' Display the waveform settings:
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim lngYOrigin As Long
Dim lngYReference As Long
Dim strOutput As String

Dim lngNumNumbers As Long
lngNumNumbers = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
lngYOrigin = dblArray(8)
lngYReference = dblArray(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
    Debug.Print "Waveform format: ASCII"
End If

If intType = 0 Then

```

41 Programming Examples

```
    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)

```

41 Programming Examples

```
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);
        }
    }
}

```

41 Programming Examples

```
// 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");
}

```

41 Programming Examples

```
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

// Download the screen image.
// -----
myScope.DoCommand(":HARDcopy:INKSaver OFF");

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// -----

// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"));

// Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINTS 10240");
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINTS?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII).
myScope.DoCommand(":WAVEform:FORMAT BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMAT?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCII");
```

```

}

double fType = fResultsArray[1];
if (fType == 0.0)
{
    Console.WriteLine("Acquire type: NORMAL");
}
else if (fType == 1.0)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
    Console.WriteLine("Acquire type: AVERage");
}
else if (fType == 3.0)
{
    Console.WriteLine("Acquire type: HRESolution");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
    out ResultsArray);
Console.WriteLine("Number of data values: {0}", nLength);

// Set up output file:
strPath = "c:\\scope\\data\\waveform_data.csv";
if (File.Exists(strPath)) File.Delete(strPath);

// Open file for output.
StreamWriter writer = File.CreateText(strPath);

// Output waveform data in CSV format.

```

41 Programming Examples

```
        for (int i = 0; i < nLength - 1; i++)
            writer.WriteLine("{0:f9}, {1:f6}",
                fxorigin + ((float)i * fXincrement),
                (((float)ResultsArray[i] - fYreference) *
                fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format BYTE data written to {0}",
            strPath);
    }

class VisaInstrument
{
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        VisaSendCommandOrQuery(strCommand);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
        byte[] dataArray)
    {
        // Send the command to the device.
        string strCommandAndLength;
        int nViStatus, nLength, nBytesWritten;

        nLength = dataArray.Length;
        strCommandAndLength = String.Format("{0} #8%08d",
            strCommand);

        // Write first part of command to formatted I/O write buffer.
```

```

nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
    nLength);
CheckVisaStatus(nViStatus);

// Write the data to the formatted I/O write buffer.
nViStatus = visa32.viBufWrite(m_nSession, dataArray, nLength,
    out nBytesWritten);
CheckVisaStatus(nViStatus);

// Check for inst errors.
CheckInstrumentErrors(strCommand);

return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}

public double DoQueryNumber(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultNumber();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double[] fResultsArray;
    fResultsArray = VisaGetResultNumbers();

    // Check for inst errors.
}

```

41 Programming Examples

```
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 = VisaGetString();
        if (!strInstrumentError.ToString().StartsWith("+0,"))
        {
            if (bFirstError)
            {
                Console.WriteLine("ERROR(s) for command '{0}': {1}",
                    strCommand),
                bFirstError = false;
            }
        }
    }
}

```

41 Programming Examples

```
        bFirstError = false;
    }
    Console.WriteLine(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 = _

```

41 Programming Examples

```
        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

```

41 Programming Examples

```
' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
DataArray = File.ReadAllBytes(strPath)

' Restore setup string.
nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
    DataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGitize.
myScope.DoCommand(":DIGitize CHANnel1")

End Sub

'
' Analyze the captured waveform.
' -----
Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte()      ' Results array.
    Dim nLength As Integer         ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
        myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    myScope.DoCommand(":MEASure:VAMplitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?")
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

    ' Download the screen image.
    ' -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF")

    ' Get the screen data.
    nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLOR", _
        ResultsArray)

    ' Store the screen data to a file.
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Screen image ({0} bytes) written to {1}", _
        nLength, strPath)

    ' Download waveform data.
```

```

' -----
' Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTS:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVEform:POINTS:MODE?"))

' Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINTS 10240")
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVEform:POINTS?"))

' Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVEform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMAT BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVEform:FORMAT?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
    Console.WriteLine("Waveform format: ASCII")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
    Console.WriteLine("Acquire type: NORMAL")
ElseIf fType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
    Console.WriteLine("Acquire type: AVERAGE")
ElseIf fType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fxIncrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fxIncrement)

Dim fxOrigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fxOrigin)

```

41 Programming Examples

```
Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVEFORM:DATA?", _
    ResultsArray)
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
        fXorigin + (CSng(index) * fXincrement), _
        ((CSng(ResultsArray(index)) - fYreference) _ 
         * fYincrement) + fYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
    strPath)

End Sub

End Class

Class VisaInstrument
    Private m_nResourceManager As Integer
    Private m_nSession As Integer
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)
        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA resource manager.
        OpenResourceManager()
    End Sub
End Class
```

```

' 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)

```

41 Programming Examples

```
' 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)

```

41 Programming Examples

```
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}': {1}, {2}",
                    strCommand, strInstrumentError)
                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

# ****

```

41 Programming Examples

```
# 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:POSIon 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:VAMPplitude")
    qresult = do_query_string(":MEASure:VAMPplitude?")
    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.

```

41 Programming Examples

```
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, wfmpnts, 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" % wfmpnts
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

```

41 Programming Examples

```
# =====
# Send a query, check for errors, return values:
# =====
def do_query_values(query):
    if debug:
        print "Qvv = '%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'" % command
            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'." % pound
        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" on page 1322
- "SICL Example in Visual Basic" on page 1331

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  300000

/* Function prototypes */
void initialize(void);           /* Initialize to known state. */
void capture(void);              /* Capture the waveform. */
void analyze(void);              /* Analyze the captured waveform. */

void do_command(char *command);   /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query); /* Query for string. */
void do_query_number(char *query); /* Query for number. */
void do_query_numbers(char *query); /* Query for numbers. */
int do_query_ieeeblock(char *query); /* Query for IEEE block. */
void check_instrument_errors();   /* Check for inst errors. */

/* Global variables */
INST id;                         /* Device session ID. */
char str_result[256] = {0};        /* Result from do_query_string(). */
double num_result;                /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /* Result from
                                                do_query_ieeeblock(). */
double dbl_results[10];           /* Result from do_query_numbers(). */

/* Main Program
 * -----
void main(void)
{
    /* Install a default SICL error handler that logs an error message
     * and exits. On Windows 98SE or Windows Me, view messages with
     * the SICL Message Viewer. For Windows 2000 or XP, use the Event
     * Viewer.
    */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the SICL_ADDRESS */
    id = iopen(SICL_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf ("Oscilloscope session opened!\n");
    }

    /* Initialize - start from a known state. */
    initialize();
}

```

41 Programming Examples

```
/* 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);

```

41 Programming Examples

```
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:VAMPplitude");
do_query_number(":MEASure:VAMPplitude?");
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");
}

```

41 Programming Examples

```
        }
    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");
    sprintf(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");
    sprintf(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");
    sprintf(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");
    sprintf(id, message);

    iscanf(id, "%lf", &num_result);
}

```

41 Programming Examples

```
        check_instrument_errors();
    }

/* Query for numbers result.
 * -----
void do_query_numbers(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    sprintf(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");
    sprintf(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, ", ");
        str_err_val[0] = '\0';
        ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
        strcat(str_out, str_err_val);
    }
}
```

```

        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/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

```

41 Programming Examples

```
' 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

```

41 Programming Examples

```
    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:VAMPplitude"
dblQueryResult = DoQueryNumber(":MEASure:VAMPplitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
FormatNumber(dblQueryResult, 4) + " V"

' Download the screen image.
' -----
DoCommand ":HARDcopy:INKSaver OFF"

' Get screen image.
Dim lngBlockSize As Long
lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPLAY:DATA? PNG, COLOR")
Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

' Save screen image to a file:
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath      ' Remove file if it exists.
End If
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)      ' Write data.
Next lngI
Close hFile      ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----
' Set the waveform points mode.
DoCommand ":WAVEform:POINTS:MODE RAW"
Debug.Print "Waveform points mode: " + _
DoQueryString(":WAVEform:POINTS:MODE?")

```

41 Programming Examples

```
' 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(db1XIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
Format(db1XOrigin, "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(db1XOrigin + (lngI * db1XIncrement), 9) + _
    ", " + _
    FormatNumber(((lngDataValue - lngYReference) * _
    sngYIncrement) + sngYOrigin)

Next lngI

' Close output file.
Close hFile   ' Close file.

```

41 Programming Examples

```
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 + vbCrLf)
    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:
```

41 Programming Examples

```
    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 1342
- "[SCPI.NET Example in Visual Basic .NET](#)" on page 1348
- "[SCPI.NET Example in IronPython](#)" on page 1354

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_02_00.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_02_00;

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-mx4054a-60154.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();
            }
        }
    }
}
```

41 Programming Examples

```
        }
        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, 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;

```

41 Programming Examples

```
// Restore setup string.  
myScope.SCPI.SYSTem.SETup.Command(strResultsArray);  
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);  
  
// Capture an acquisition using :DIGitize.  
myScope.SCPI.DIGitize.Command("CHANnel1", null, null, null, null);  
}  
  
/*  
 * Analyze the captured waveform.  
 * -----  
 */  
private static void Analyze()  
{  
    string strResults, source1, source2;  
    double fResult;  
  
    // Make a couple of measurements.  
    // -----  
    myScope.SCPI.MEASure.SOURce.Command("CHANnel1", null);  
    myScope.SCPI.MEASure.SOURce.Query(out source1, out source2);  
    Console.WriteLine("Measure source: {0}", source1);  
  
    myScope.SCPI.MEASure.FREQuency.Command("CHANnel1");  
    myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", out fResult);  
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);  
  
    // Use direct command/query when commands not in command set.  
    myScope.Transport.Command.Invoke(":MEASure:VAMplitude CHANnel1");  
    myScope.Transport.Query.Invoke(":MEASure:VAMplitude? CHANnel1",  
        out strResults);  
    Console.WriteLine("Vertical amplitude: {0} V", strResults);  
  
    // Download the screen image.  
    // -----  
    myScope.SCPI.HARDcopy.INKSaver.Command(false);  
  
    // Get the screen data.  
    byte[] byteResultsArray; // Results array.  
    myScope.SCPI.DISPlay.DATA.Query("PNG", "COLOR",  
        out byteResultsArray);  
    int nLength; // Number of bytes returned from instrument.  
    nLength = byteResultsArray.Length;  
  
    // Store the screen data to a file.  
    string strPath;  
    strPath = "c:\\scope\\data\\screen.png";  
    FileStream fStream = File.Open(strPath, FileMode.Create);  
    fStream.Write(byteResultsArray, 0, nLength);  
    fStream.Close();  
    Console.WriteLine("Screen image ({0} bytes) written to {1}",  
        nLength, strPath);  
  
    // Download waveform data.  
    // -----
```

```

// Set the waveform points mode.
myScope.SCPI.WAVEform.POINts.MODE.Command("RAW");
myScope.SCPI.WAVEform.POINts.MODE.Query(out strResults);
Console.WriteLine("Waveform points mode: {0}", strResults);

// Get the number of waveform points available.
myScope.SCPI.WAVEform.POINts.CommandPoints(10240);
int nPointsAvail;
myScope.SCPI.WAVEform.POINts.Query1(out nPointsAvail);
Console.WriteLine("Waveform points available: {0}", nPointsAvail);

// Set the waveform source.
myScope.SCPI.WAVEform.SOURce.Command("CHANnel1");
myScope.SCPI.WAVEform.SOURce.Query(out strResults);
Console.WriteLine("Waveform source: {0}", strResults);

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.SCPI.WAVEform.FORMat.Command("BYTE");
myScope.SCPI.WAVEform.FORMat.Query(out strResults);
Console.WriteLine("Waveform format: {0}", strResults);

// Display the waveform settings:
int nFormat, nType, nPoints, nCount, nXreference, nYreference;
double dblXincrement, dblXorigin, dblYincrement, dblYorigin;
myScope.SCPI.WAVEform.PREamble.Query(
    out nFormat,
    out nType,
    out nPoints,
    out nCount,
    out dblXincrement,
    out dblXorigin,
    out nXreference,
    out dblYincrement,
    out dblYorigin,
    out nYreference);

if (nFormat == 0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (nFormat == 1)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (nFormat == 2)
{
    Console.WriteLine("Waveform format: ASCII");
}

if (nType == 0)
{
    Console.WriteLine("Acquire type: NORMAL");
}
else if (nType == 1)
{
    Console.WriteLine("Acquire type: PEAK");
}

```

```
else if (nType == 2)
{
    Console.WriteLine("Acquire type: AVERage");
}
else if (nType == 3)
{
    Console.WriteLine("Acquire type: HRESolution");
}

Console.WriteLine("Waveform points: {0:e}", nPoints);
Console.WriteLine("Waveform average count: {0:e}", nCount);
Console.WriteLine("Waveform X increment: {0:e}", dblXincrement);
Console.WriteLine("Waveform X origin: {0:e}", dblXorigin);
Console.WriteLine("Waveform X reference: {0:e}", nXreference);
Console.WriteLine("Waveform Y increment: {0:e}", dblYincrement);
Console.WriteLine("Waveform Y origin: {0:e}", dblYorigin);
Console.WriteLine("Waveform Y reference: {0:e}", nYreference);

// Read waveform data.
myScope.SCPI.WAVEform.DATA.QueryBYTE(out byteResultsArray);
nLength = byteResultsArray.Length;
Console.WriteLine("Number of data values: {0}", nLength);

// Set up output file:
strPath = "c:\\scope\\data\\waveform_data.csv";
if (File.Exists(strPath)) File.Delete(strPath);

// Open file for output.
StreamWriter writer = File.CreateText(strPath);

// Output waveform data in CSV format.
for (int i = 0; i < nLength - 1; i++)
    writer.WriteLine("{0:f9}, {1:f6}",
                    dblXorigin + ((float)i * dblXincrement),
                    (((float)byteResultsArray[i] - nYreference)
                     * dblYincrement) + dblYorigin);

// Close output file.
writer.Close();
Console.WriteLine("Waveform format BYTE data written to {0}",
                  strPath);
    }
}
```

SCPI.NET Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Install the 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_02_00.dll**; then, click **OK**.
 - e** Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.ScpiNetInstrumentApp" as the **Startup object**.
- 7** Build and run the program.

For more information, see the SCPI.NET driver help that comes with 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_02_00

Namespace InfiniiVision
    Class ScpiNetInstrumentApp
        Private Shared myScope As AgInfiniiVision3000X

        Public Shared Sub Main(ByVal args As String())
            Try
                Dim strScopeAddress As String
                'strScopeAddress = "a-mx3054a-60028.cos.agilent.com";
                strScopeAddress =
                    "TCPIPO::a-mx4054a-60154.cos.agilent.com::inst0::INSTR"
                Console.WriteLine("Connecting to oscilloscope...")
            End Try
        End Sub
    End Class
End Namespace
```

41 Programming Examples

```
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)

```

41 Programming Examples

```
' Or, configure by loading a previously saved setup.
Dim nBytesWritten As Integer

strPath = "c:\scope\config\setup.stp"
strResultsArray = File.ReadAllLines(strPath)
nBytesWritten = strResultsArray.Length

' Restore setup string.
myScope.SCPI.SYSTem.SETup.Command(strResultsArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGITIZE.
myScope.SCPI.DIGITIZE.Command("CHANnel1", Nothing, Nothing, _
Nothing, Nothing)
End Sub

' Analyze the captured waveform.
' -----
Private Shared Sub Analyze()
    Dim strResults As String, source1 As String, source2 As String
    Dim fResult As Double

    ' Make a couple of measurements.
    ' -----
    myScope.SCPI.MEASure.SOURce.Command("CHANnel1", Nothing)
    myScope.SCPI.MEASure.SOURce.Query(source1, source2)
    Console.WriteLine("Measure source: {0}", source1)

    myScope.SCPI.MEASure.FREQuency.Command("CHANnel1")
    myScope.SCPI.MEASure.FREQuency.Query("CHANnel1", fResult)
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    ' Use direct command/query when commands not in command set.
    myScope.Transport.Command.Invoke(":MEASure:VAMplitude CHANnel1")
    myScope.Transport.Query.Invoke(":MEASure:VAMplitude? CHANnel1", _
strResults)
    Console.WriteLine("Vertical amplitude: {0} V", strResults)

    ' Download the screen image.
    ' -----
    myScope.SCPI.HARDcopy.INKSaver.Command(False)

    ' Get the screen data.
    Dim byteResultsArray As Byte()
    ' Results array.
    myScope.SCPI.DISPlay.DATA.Query("PNG", "COLOR", byteResultsArray)
    Dim nLength As Integer
    ' Number of bytes returned from instrument.
    nLength = byteResultsArray.Length

    ' Store the screen data to a file.
    Dim strPath As String
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream = File.Open(strPath, FileMode.Create)
    fStream.Write(byteResultsArray, 0, nLength)
    fStream.Close()
```

```

Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' -----
' Set the waveform points mode.
myScope.SCPI.WAVEform.POINts.MODE.Command("RAW")
myScope.SCPI.WAVEform.POINts.MODE.Query(strResults)
Console.WriteLine("Waveform points mode: {0}", strResults)

' Get the number of waveform points available.
myScope.SCPI.WAVEform.POINts.CommandPoints(10240)
Dim nPointsAvail As Integer
myScope.SCPI.WAVEform.POINts.Query1(nPointsAvail)
Console.WriteLine("Waveform points available: {0}", nPointsAvail)

' Set the waveform source.
myScope.SCPI.WAVEform.SOURce.Command("CHANnel1")
myScope.SCPI.WAVEform.SOURce.Query(strResults)
Console.WriteLine("Waveform source: {0}", strResults)

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.SCPI.WAVEform.FORMat.Command("BYTE")
myScope.SCPI.WAVEform.FORMat.Query(strResults)
Console.WriteLine("Waveform format: {0}", strResults)

' Display the waveform settings:
Dim nFormat As Integer, nType As Integer, nPoints As Integer, _
    nCount As Integer, nXreference As Integer, _
    nYreference As Integer
Dim dblXincrement As Double, dblXorigin As Double, _
    dblYincrement As Double, dblYorigin As Double
myScope.SCPI.WAVEform.PREAMble.Query(nFormat, nType, nPoints, _
    nCount, dblXincrement, dblXorigin, nXreference, _
    dblYincrement, dblYorigin, nYreference)

If nFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf nFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf nFormat = 2 Then
    Console.WriteLine("Waveform format: ASCII")
End If

If nType = 0 Then
    Console.WriteLine("Acquire type: NORMAL")
ElseIf nType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf nType = 2 Then
    Console.WriteLine("Acquire type: AVERAGE")
ElseIf nType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Console.WriteLine("Waveform points: {0:e}", nPoints)
Console.WriteLine("Waveform average count: {0:e}", nCount)

```

41 Programming Examples

```
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("AgInfiniiVision4000X_01_20")
from Agilent.CommandExpert.ScpiNet.AgInfiniiVision4000X_01_20 import *

# =====
# Initialize:
# =====
def initialize():

    # Get and display the device's *IDN? string.
    idn_string = scope.SCPI.IDN.Query()
    print "Identification string '%s'" % idn_string

    # Clear status and load the default setup.
    scope.SCPI.CLS.Command()
    scope.SCPI.RST.Command()

# =====
# Capture:
# =====
def capture():

    # Use auto-scale to automatically set up oscilloscope.
    print "Autoscale."
    scope.SCPI.AUToscale.Command(None, None, None, None, None)

    # Set trigger mode.
    scope.SCPI.TRIGger.MODE.Command("EDGE")
    qresult = scope.SCPI.TRIGger.MODE.Query()
```

41 Programming Examples

```
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",
    }

```

41 Programming Examples

```
        }
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 = AgInfiniiVision4000X(addr)
```

```
scope.Transport.DefaultTimeout.Set(10000)

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."

# Wait for a key press before exiting.
print "Press any key to exit..."
Console.ReadKey(True)
```

41 Programming Examples

Index

Symbols

+9.9E+37, infinity representation, 1239
+9.9E+37, measurement error, 429

Numerics

0 (zero) values in waveform data, 1053
1 (one) values in waveform data, 1053
10 MHz REF BNC, enabling/disabling, 952
10 MHz reference signal, 237
3000 X-Series oscilloscopes, command differences from, 39
82350B GPIB interface, 6

A

A429 SEARch commands, 861
A429 serial bus commands, 662
absolute value math function, 364
AC coupling, trigger edge, 985
AC input coupling for specified channel, 271
AC RMS measured on waveform, 446, 488
accumulate activity, 195
ACQuire commands, 231
acquire data, 203, 244
acquire mode on autoscale, 199
acquire reset conditions, 180, 939
acquire sample rate, 243
ACQuire subsystem, 61
acquired data points, 236
acquisition count, 234
acquisition mode, 231, 235, 1070
acquisition type, 231, 244
acquisition types, 1046
active edges, 195
active printer, 378
activity logic levels, 195
activity on digital channels, 195
add function, 1065
add math function, 364
add math function as g(t) source, 1150
address field size, IIC serial decode, 741
address of network printer, 383
address value, USB search, 917
address value, USB trigger, 818
address, IIC trigger pattern, 744
Addresses softkey, 48
adjacent signal source, USB 2.0 signal quality test, 291
AER (Arm Event Register), 196, 215, 217, 1214
Agilent Connection Expert, 49
Agilent Interactive IO application, 53

Agilent IO Control icon, 49
Agilent IO Libraries Suite, 6, 45, 58, 60
Agilent IO Libraries Suite, installing, 46
alignment, I2S trigger, 723
all (snapshot) measurement, 430
ALL segments waveform save option, 654
AM demo signal, 300
AM depth, waveform generator modulation, 1097
AM modulation type, waveform generator, 1107
amplitude tracking, dual channel waveform generator, 1115, 1117
amplitude, vertical, 442, 481
amplitude, waveform generator, 1118
analog channel coupling, 271
analog channel display, 272
analog channel impedance, 273
analog channel input, 1144
analog channel inversion, 274
analog channel labels, 275, 324
analog channel offset, 276
analog channel protection lock, 942
analog channel range, 283
analog channel scale, 284
analog channel source for glitch, 998
analog channel units, 285
analog channels only oscilloscopes, 6
analog probe attenuation, 277
analog probe head type, 278
analog probe sensing, 1145
analog probe skew, 280, 1143
analyzing captured data, 57
angle brackets, 162
annotate channels, 275
annotation background, display, 318
annotation color, display, 319
annotation text, display, 320
annotation, display, 317
apparent power, 500
apply network printer connection settings, 384
arbitrary waveform generator output, 1093
arbitrary waveform, byte order, 1083
arbitrary waveform, capturing from other sources, 1089
arbitrary waveform, clear, 1086
arbitrary waveform, download DAC values, 1087
arbitrary waveform, download floating-point values, 1084
arbitrary waveform, interpolation, 1088
arbitrary waveform, points, 1085
arbitrary waveform, recall, 625
arbitrary waveform, save, 636

area for hardcopy print, 377
area for saved image, 1184
area measurement, 431, 441
ARINC 429 auto setup, 664
ARINC 429 base, 665
ARINC 429 demo signal, 302
ARINC 429 signal speed, 672
ARINC 429 signal type, 670
ARINC 429 source, 671
ARINC 429 trigger data pattern, 674, 864
ARINC 429 trigger label, 673, 677, 862
ARINC 429 trigger SDI pattern, 675, 865
ARINC 429 trigger SSM pattern, 676, 866
ARINC 429 trigger type, 678, 863
ARINC 429 word and error counters, reset, 667
ARINC 429 word format, 669
Arm Event Register (AER), 196, 215, 217, 1214
arming edge slope, Edge Then Edge trigger, 972
arming edge source, Edge Then Edge trigger, 973
arrange waveforms, 1147
ASCII format, 1055
ASCII format for data transfer, 1049
ASCII string, quoted, 162
ASCIixy waveform data format, 651
assign channel names, 275
attenuation factor (external trigger) probe, 339
attenuation for oscilloscope probe, 277
audio channel, I2S trigger, 732
AUT option for probe sense, 1145, 1149
Auto Range capability for DVM, 330
auto set up, trigger level, 965
auto set up, USB 2.0 signal quality test, 288
auto setup (ARINC 429), 664
auto setup for M1553 trigger, 765
auto setup for power analysis signals, 591
auto trigger sweep mode, 959
automask create, 523
automask source, 524
automask units, 525
automatic measurements constants, 277
automatic probe type detection, 1145, 1149
autoscale, 197
autoscale acquire mode, 199
autoscale channels, 200
AUToscale command, 60
autoset for FLEXray event trigger, 714
autosetup for FLEXray decode, 704
average value measurement, 443, 482
Average, power modulation analysis, 580
averaged value math function, 365
averaging acquisition type, 232, 1047
averaging, synchronizing with, 1228

Ax + B math function, 364

B

bandwidth filter limits, 338

bandwidth filter limits to 20 MHz, 270

bar chart of current harmonics results, 567

base 10 exponential math function, 364

base value measurement, 444, 483

base, ARINC 429, 665

base, I2S serial decode, 724

base, MIL-STD-1553 serial decode, 766

base, UART trigger, 802

base, USB decode, 812

basic instrument functions, 168

baud rate, 689, 754, 791

begin acquisition, 203, 224, 226

BHARris window for minimal spectral leakage, 357

binary block data, 162, 322, 943, 1053

BINary waveform data format, 651

bind levels for masks, 544

bit order, 792

bit order, SPI decode, 773

bit selection command, bus, 249

bit weights, 172

bitmap display, 322

bits in Service Request Enable Register, 185

bits in Standard Event Status Enable Register, 170

bits in Status Byte Register, 187

bits selection command, bus, 250

blank, 202

block data, 162, 175, 943

block response data, 64

blocking synchronization, 1223

blocking wait, 1222

BMP format screen image data, 322

braces, 161

built-in measurements, 57

burst data demo signal, 300

burst width measurement, 432

burst, minimum time before next, 980

bus bit selection command, 249

bus bits selection commands, 250

bus clear command, 252

bus commands, 248

BUS data format, 1050

bus display, 253

bus label command, 254

bus mask command, 255

BUS<n> commands, 247

button disable, 938

button, calibration protect, 262

byte format for data transfer, 1049, 1055

BYTeorder, 1051

C

C, SICL library example, 1322

C, VISA library example, 1275

C#, SCPI.NET example, 1342

C#, VISA COM example, 1251

C#, VISA example, 1294

CAL PROTECT button, 262

CAL PROTECT switch, 258

calculating preshoot of waveform, 460

calculating the waveform overshoot, 454

calibrate, 259, 260, 262, 266

CALibrate commands, 257

calibrate date, 259

calibrate introduction, 258

calibrate label, 260

calibrate output, 261

calibrate start, 263

calibrate status, 264

calibrate switch, 262

calibrate temperature, 265

calibrate time, 266

CAN acknowledge, 688

CAN and LIN demo signal, 302

CAN baud rate, 689

CAN demo signal, 301

CAN frame counters, reset, 684

CAN SEARch commands, 867

CAN serial bus commands, 680

CAN serial search, data, 870

CAN serial search, data length, 871

CAN serial search, ID, 872

CAN serial search, ID mode, 873

CAN serial search, mode, 868

CAN signal definition, 690

CAN source, 691

CAN symbolic data display, 687

CAN symbolic data, recall, 626

CAN trigger, 692, 696

CAN trigger data pattern, 695

CAN trigger ID pattern, 697

CAN trigger pattern id mode, 698

CAN triggering, 657

capture data, 203

capturing data, 56

cardiac waveform generator output, 1093

center frequency set, 354, 364

center of screen, 1078

center reference, 953

center screen, vertical value at, 363, 368

channel, 230, 275, 1140, 1142

channel coupling, 271

channel display, 272

channel input impedance, 273

channel inversion, 274

channel label, 275, 1141

channel labels, 323, 324

channel numbers, 1147

channel overload, 282

channel protection, 282

channel reset conditions, 180, 939

channel selected to produce trigger, 998, 1030

channel signal type, 281

channel skew for oscilloscope probe, 280,

1143

channel status, 227, 1147

channel threshold, 1142

channel vernier, 286

channel, stop displaying, 202

CHANnel<n> commands, 267, 269

channels to autoscale, 200

channels, how autoscale affects, 197

characters to display, 936

chart logic bus state math function, 365

chart logic bus state, clock edge, 349

chart logic bus state, clock source, 348

chart logic bus timing math function, 365

chart logic bus, units, 352

chart logic bus, value for data = 0, 351

chart logic bus, value for data increment, 350

classes of input signals, 357

classifications, command, 1232

clear, 321

clear bus command, 252

clear cumulative edge variables, 1140

clear markers, 433, 1161

clear measurement, 433, 1161

clear message queue, 169

Clear method, 59

clear reference waveforms, 1125

clear screen, 1148

clear status, 169

clear waveform area, 316

clipped high waveform data value, 1053

clipped low waveform data value, 1053

clock, 742, 774, 777

clock slope, I2S, 725

CLOCK source, I2S, 727

clock source, setup and hold trigger, 1017

clock timeout, SPI, 775

clock with infrequent glitch demo signal, 300

CLS (Clear Status), 169

CME (Command Error) status bit, 170, 172

CMOS threshold voltage for digital channels, 313, 1142

CMOS trigger threshold voltage, 1189

code, :ACQuire:COMPlEte, 233

code, :ACQuire:SEGmented, 240

code, :ACQuire:TYPE, 245

code, :AUToscale, 198

code, :CHANnel<n>:LABel, 275

code, :CHANnel<n>:PROBe, 277

code, :CHANnel<n>:RANGE, 283

code, :DIGItize, 204

code, :DISPlay:DATA, 322

code, :DISPlay:LABel, 323

code, :DISPlay:ORDER, 1147

code, :MEASure:PERiod, 469

code, :MEASure:RESults, 462

code, :MEASure:TEDGe, 477

code, :MTESt, 519

code, :POD<n>:THRESHold, 555

code, :RUN:/STOP, 224

code, :SYSTEM:SETup, 943

code, :TIMEbase:DElay, 1188

code, :TIMEbase:MODE, 949

code, :TIMEbase:RANGE, 951

code, :TIMEbase:REFerence, 953

code, :TRIGger:MODE, 968
 code, :TRIGger:SLOPe, 988
 code, :TRIGger:SOURce, 989
 code, :VIEW and :BLANK, 230
 code, :WAveform, 1066
 code, :WAveform:DATA, 1053
 code, :WAveform:POInts, 1057
 code, :WAveform:PREamble, 1061
 code, :WAveform:SEGmented, 240
 code, *RST, 182
 code, SCPI.NET library example in C#, 1342
 code, SCPI.NET library example in IronPython, 1354
 code, SCPI.NET library example in Visual Basic .NET, 1348
 code, SICL library example in C, 1322
 code, SICL library example in Visual Basic Basic, 1331
 code, VISA COM library example in C#, 1251
 code, VISA COM library example in Python, 1268
 code, VISA COM library example in Visual Basic Basic, 1242
 code, VISA COM library example in Visual Basic .NET, 1260
 code, VISA library example in C, 1275
 code, VISA library example in C#, 1294
 code, VISA library example in Python, 1315
 code, VISA library example in Visual Basic Basic, 1284
 code, VISA library example in Visual Basic .NET, 1305
 colon, root commands prefixed by, 194
 color palette for hardcopy, 389
 color palette for image, 643
 Comma Separated Values (CSV) waveform data format, 651
 command classifications, 1232
 command differences from 3000 X-Series oscilloscopes, 39
 command errors detected in Standard Event Status, 172
 Command Expert, 1342
 command header, 1233
 command headers, common, 1235
 command headers, compound, 1235
 command headers, simple, 1235
 command strings, valid, 1233
 commands quick reference, 69
 commands sent over interface, 168
 commands, more about, 1231
 commands, obsolete and discontinued, 1133
 common (*) commands, 3, 165, 168
 common command headers, 1235
 common logarithm math function, 364
 completion criteria for an acquisition, 233, 234
 COMPliance commands, 287
 compound command headers, 1235
 compound header, 1237
 computer control examples, 1241
 conditions for external trigger, 337
 conditions, reset, 180, 939
 conduction calculation method for switching loss, 613
 Config softkey, 48
 configurations, oscilloscope, 175, 179, 183, 943
 Configure softkey, 48
 connect oscilloscope, 47
 connect sampled data points, 1146
 connection type, hi-speed, USB 2.0 signal quality test, 296
 constants for making automatic measurements, 277
 constants for scaling display factors, 277
 constants for setting trigger levels, 277
 controller initialization, 56
 copy display, 223
 copy signal, dual channel waveform generator, 1116
 core commands, 1232
 count, 1052
 count values, 234
 count, averaged value math function, 347
 count, Edge Then Edge trigger, 975
 count, Nth edge of burst, 979
 counter, 434
 coupling, 985
 coupling for channels, 271
 CRC value, USB search, 918
 CRC value, USB trigger, 819
 create automask, 523
 crest factor, 502
 CSV (Comma Separated Values) waveform data format, 651
 cumulative edge activity, 1140
 current harmonics analysis fail count, 568
 current harmonics analysis results, save, 647
 current harmonics analysis run count, 571
 current harmonics analysis, apply, 565
 current harmonics results data, 566
 current harmonics results display, 567
 current logic levels on digital channels, 195
 current oscilloscope configuration, 175, 179, 183, 943
 current probe, 285, 341
 CURRent segment waveform save option, 654
 current source, 608
 cursor mode, 401
 cursor position, 400, 402, 404, 406, 409, 411
 cursor readout, 1162, 1166, 1167
 cursor reset conditions, 180, 939
 cursor source, 403, 405
 cursor time, 1162, 1166, 1167
 cursor units, X, 407, 408
 cursor units, Y, 412, 413
 cursors track measurements, 467
 cursors, how autoscale affects, 197
 cursors, X1, X2, Y1, Y2, 398
 cycle count base, FLEXray frame trigger, 717
 cycle count repetition, FLEXray frame trigger, 718
 cycle measured, 447, 450
 cycle time, 457
 cycles analyzed, number of, 592, 593

D

D-source, USB 2.0 signal quality test, 293
 D-USB signal source, 813
 D+ source, USB 2.0 signal quality test, 294
 D+ USB signal source, 814
 data, 743, 745, 1053
 data (waveform) maximum length, 653
 data 2, 746
 data acquisition types, 1046
 data conversion, 1048
 data format for transfer, 1048
 data length, USB search, 920
 data length, USB trigger, 821
 data output order, 1051
 data pattern length, 696, 762
 data pattern, ARINC 429 trigger, 674, 864
 data pattern, CAN trigger, 695
 data PID, USB search, 925
 data PID, USB trigger, 827
 data point index, 1075
 data points, 236
 data record, measurement, 1058
 data record, raw acquisition, 1058
 data required to fill time buckets, 233
 DATA source, I2S, 728
 data source, setup and hold trigger, 1018
 data structures, status reporting, 1201
 data value, USB search, 919
 data value, USB trigger, 820
 data, saving and recalling, 316
 date, calibration, 259
 date, system, 935
 dB versus frequency, 364
 DC coupling for edge trigger, 985
 DC input coupling for specified channel, 271
 DC offset correction for integrate input, 360
 DC RMS measured on waveform, 446, 488
 DC waveform generator output, 1092
 DDE (Device Dependent Error) status bit, 170, 172
 decision chart, status reporting, 1220
 default conditions, 180, 939
 define channel labels, 275
 define glitch trigger, 996
 define logic thresholds, 1142
 define measurement, 437
 define measurement source, 468
 define trigger, 997, 1005, 1006, 1009
 defined as, 161
 definite-length block query response, 64
 definite-length block response data, 162
 delay measured to calculate phase, 458
 delay measurement, 437
 delay measurements, 476
 delay parameters for measurement, 439
 delay time, Edge Then Edge trigger, 974
 DElay trigger commands, 971
 delay, how autoscale affects, 197
 delayed time base, 949

- delayed window horizontal scale, 958
delete mask, 533
delta time, 1162
delta voltage measurement, 1171
delta X cursor, 398
delta Y cursor, 398
demo, 299
DEMO commands, 299
demo signal, 301
demo signal function, 300
demo signal phase angle, 304
demo signals output control, 305
deskew for power measurements, 562
detecting probe types, 1145, 1149
device-defined error queue clear, 169
DIFF source for function, 1153
differences from 3000 X-Series oscilloscope commands, 39
differential D+ and D- USB signal source, 815
differential probe heads, 278
differential probe source, USB 2.0 signal quality test, 292
differential signal type, 281
differentiate math function, 364, 1065
digital channel commands, 308, 309, 310, 311, 313
digital channel data, 1050
digital channel labels, 324
digital channel order, 1147
digital channel source for glitch trigger, 998
digital channels, 6
digital channels, activity and logic levels on, 195
digital channels, groups of, 551, 553, 555
digital pod, stop displaying, 202
digital reset conditions, 181, 940
DIGItal-<d> commands, 307
digitize channels, 203
DIGIte command, 56, 61, 1046
digits, 162
disable front panel, 938
disable function, 1154
disabling calibration, 262
disabling channel display, 272
disabling status register bits, 170, 184
discontinued and obsolete commands, 1133
display annotation, 317
display annotation background, 318
display annotation color, 319
display annotation text, 320
display channel labels, 323
display clear, 321
DISPLAY commands, 315
display commands introduction, 316
display connect, 1146
display date, 935
display factors scaling, 277
display for channels, 272
display frequency span, 355
display measurements, 428, 467
display order, 1147
display persistence, 327
display reference, 950, 953
display reference waveforms, 1126
display reset conditions, 181, 940
display serial number, 225
display vectors, 328
display wave position, 1147
display, lister, 395
display, measurement statistics on/off, 471
display, oscilloscope, 309, 327, 353, 553, 936
display, serial decode bus, 660
displaying a baseline, 970
displaying unsynchronized signal, 970
divide math function, 364
DNS IP, 48
domain, 48
domain, network printer, 385
driver, printer, 1159
DSO models, 6
duplicate mnemonics, 1237
duration, 1005, 1006, 1009
duration for glitch trigger, 992, 993, 997
duration of power analysis, 594, 595, 596, 597, 598, 599
duration qualifier, trigger, 1005, 1006
duration triggering, 960
duty cycle measurement, 57, 428, 447, 450
Duty Cycle, power modulation analysis, 580
DVM commands, 329
DVM displayed value, 331
DVM enable/disable, 332
DVM frequency value, 333
DVM input source, 335
DVM mode, 334
- ## E
- EBURst trigger commands, 978
ECL channel threshold, 1142
ECL threshold voltage for digital channels, 313
ECL trigger threshold voltage, 1189
edge activity, 1140
edge counter, Edge Then Edge trigger, 975
edge counter, Nth edge of burst, 979
edge coupling, 985
edge fall time, 448
edge parameter for delay measurement, 439
edge preshoot measured, 460
edge rise time, 465
EDGE SEARch commands, 841
edge search slope, 842
edge search source, 843
edge slope, 988
edge source, 989
edge string for OR'ed edge trigger, 1000
EDGE trigger commands, 983
edge triggering, 960
edges (activity) on digital channels, 195
edges in measurement, 437
efficiency, 503
efficiency power analysis, apply, 563
elapsed time in mask test, 530
ellipsis, 162
enable channel labels, 323
enabling calibration, 262
enabling channel display, 272
enabling status register bits, 170, 184
end of string (EOS) terminator, 1234
end of text (EOT) terminator, 1234
end or identify (EOI), 1234
endpoint value, USB search, 921
endpoint value, USB trigger, 822
energy loss, 504
EOI (end or identify), 1234
EOS (end of string) terminator, 1234
EOT (end of text) terminator, 1234
equivalent time acquisition mode, 235
erase data, 321
erase measurements, 1161
erase screen, 1148
error counter (ARINC 429), 666
error counter (ARINC 429), reset, 667
error frame count (CAN), 682
error frame count (UART), 793
error messages, 937, 1191
error number, 937
error queue, 937, 1211
error, measurement, 428
ESB (Event Status Bit), 185, 187
ESE (Standard Event Status Enable Register), 170, 1210
ESR (Standard Event Status Register), 172, 1209
ET value, USB search, 922
ET value, USB trigger, 823
ETE demo signal, 301
event status conditions occurred, 187
Event Status Enable Register (ESE), 170, 1210
Event Status Register (ESR), 172, 229, 1209
example code, :ACQuire:COMplete, 233
example code, :ACQuire:SEGmented, 240
example code, :ACQuire:TYPE, 245
example code, :AUToscale, 198
example code, :CHANnel<n>:LABel, 275
example code, :CHANnel<n>:PROBe, 277
example code, :CHANnel<n>:RANGe, 283
example code, :DIGIte, 204
example code, :DISPlay:DATA, 322
example code, :DISPlay:LABel, 323
example code, :DISPlay:ORDer, 1147
example code, :MEASure:PERiod, 469
example code, :MEASure:RESults, 462
example code, :MEASure:EDGE, 477
example code, :MTEST, 519
example code, :POD<n>:THReShold, 555
example code, :RUN:/STOP, 224
example code, :SYSTem:SETup, 943
example code, :TIMEbase:DELay, 1188
example code, :TIMEbase:MODE, 949
example code, :TIMEbase:RANGe, 951
example code, :TIMEbase:REference, 953
example code, :TRIGger:MODE, 968
example code, :TRIGger:SLOPe, 988
example code, :TRIGger:SOURce, 989
example code, :VIEW and :BLANK, 230

example code, :WAVEform, 1066
 example code, :WAVEform:DATA, 1053
 example code, :WAVEform:POINts, 1057
 example code, :WAVEform:PREamble, 1061
 example code, :WAVEform:SEGmented, 240
 example code, *RST, 182
 example programs, 6, 1241
 EXE (Execution Error) status bit, 170, 172
 execution error detected in Standard Event Status, 172
 exponential fall waveform generator output, 1092
 exponential math function, 364
 exponential notation, 161
 exponential rise waveform generator output, 1092
 extended video triggering license, 1031
 external glitch trigger source, 998
 external range, 340
 external trigger, 337, 339, 989
 EXternal trigger commands, 337
 EXternal trigger level, 986
 external trigger probe attenuation factor, 339
 external trigger probe sensing, 1149
 EXternal trigger source, 989
 external trigger units, 341

F

fail count, current harmonics analysis, 568
 fail/pass status (overall) for current harmonics analysis, 573
 failed waveforms in mask test, 528
 failure, self test, 189
 fall time measurement, 428, 448
 Fall Time, power modulation analysis, 580
 falling edge count measurement, 451
 falling pulse count measurement, 452
 Fast Fourier Transform (FFT) functions, 354, 355, 357, 364, 1153
 FF values in waveform data, 1053
 FFT (Fast Fourier Transform) functions, 354, 355, 357, 364, 1153
 FFT (Fast Fourier Transform) operation, 1065
 FFT vertical units, 356
 fifty ohm impedance, disable setting, 942
 filename for hardcopy, 1156
 filename for recall, 627, 1090
 filename for save, 638
 filter for frequency reject, 987
 filter for high frequency reject, 963
 filter for noise reject, 969
 filter used to limit bandwidth, 270, 338
 filters to Fast Fourier Transforms, 357
 filters, math, 364
 fine horizontal adjustment (vernier), 955
 fine vertical adjustment (vernier), 286
 finish pending device operations, 176
 first point displayed, 1075
 FLATtop window for amplitude measurements, 357
 FLEXray autoset for event trigger, 714

FLEXray autosetup, 704
 FlexRay demo signal, 302
 FlexRay frame counters, reset, 708
 FLEXray SEARch commands, 877
 FlexRay serial search, cycle, 878
 FlexRay serial search, data, 879
 FlexRay serial search, data length, 880
 FlexRay serial search, frame, 881
 FlexRay serial search, mode, 882
 FLEXray source, 711
 FLEXray trigger, 712
 FLEXray trigger commands, 702
 FM burst demo signal, 301
 FM modulation type, waveform generator, 1107
 force trigger, 962
 format, 1055, 1060
 format (word), ARINC 429, 669
 format for block data, 175
 format for generic video, 1027, 1031
 format for hardcopy, 1155
 format for image, 641
 format for waveform data, 651
 FormattedIO488 object, 59
 formfeed for hardcopy, 376, 380
 formulas for data conversion, 1048
 frame, 778
 frame counters (CAN), error, 682
 frame counters (CAN), overload, 683
 frame counters (CAN), reset, 684
 frame counters (CAN), total, 685
 frame counters (FlexRay), null, 707, 709
 frame counters (FlexRay), reset, 708
 frame counters (FlexRay), total, 710
 frame counters (UART), error, 793
 frame counters (UART), reset, 794
 frame counters (UART), Rx frames, 795
 frame counters (UART), Tx frames, 796
 frame ID, FLEXray BSS event trigger, 715
 frame ID, FLEXray frame trigger, 719
 frame type, FLEXray frame trigger, 720
 frame value, USB search, 923
 frame value, USB trigger, 824
 framing, 776
 frequency deviation, waveform generator FM modulation, 1099
 frequency measurement, 57, 428, 449
 frequency measurements with X cursors, 407
 frequency resolution, 357
 frequency span of display, 355
 frequency versus dB, 364
 Frequency, power modulation analysis, 580
 front panel mode, 970
 front panel Single key, 226
 front panel Stop key, 228
 front-panel lock, 938
 FSK modulation type, waveform generator, 1107
 FSK rate, waveform generator modulation, 1102
 full-scale horizontal time, 951, 957
 full-scale vertical axis defined, 367

function, 353, 354, 355, 357, 363, 364, 367, 368, 369, 1153, 1154
 FUNction commands, 343
 function memory, 227
 function turned on or off, 1154
 function, demo signal, 300
 function, first source input, 370
 function, second source input, 372
 function, waveform generator, 1091
 functions, 1065

G

g(t) source, first input channel, 1151
 g(t) source, math operation, 1150
 g(t) source, second input channel, 1152
 gain for Ax + B math operation, 361
 gated measurement window, 491
 gateway IP, 48
 gaussian pulse waveform generator output, 1093
 general SBUS<n> commands, 659
 general SEARch commands, 836
 general trigger commands, 961
 GENeric, 1027, 1031
 generic video format, 1027, 1031
 Generic video trigger, edge number, 1032
 Generic video trigger, greater than sync pulse width time, 1035
 Generic video trigger, horizontal sync control, 1033
 Generic video trigger, horizontal sync pulse time, 1034
 glitch demo signal, 300
 glitch duration, 997
 glitch qualifier, 996
 GLITch SEARch commands, 844
 glitch search, greater than value, 845
 glitch search, less than value, 846
 glitch search, polarity, 847
 glitch search, qualifier, 848
 glitch search, range, 849
 glitch search, source, 850
 glitch source, 998
 GLITch trigger commands, 990
 glitch trigger duration, 992
 glitch trigger polarity, 995
 glitch trigger source, 992
 graticule area for hardcopy print, 377
 graticule colors, invert for hardcopy, 381, 1158
 graticule colors, invert for image, 642
 grayscale palette for hardcopy, 389
 grayscale palette for image, 643
 grayscaling on hardcopy, 1157
 greater than qualifier, 996
 greater than time, 992, 997, 1005, 1009
 greater than value for glitch search, 845
 groups of digital channels, 551, 553, 555, 1142

H

handshake PID, USB search, 926
 handshake PID, USB trigger, 828
 HANNING window for frequency resolution, 357
 hardcopy, 223, 376
 HARDCopy commands, 375
 hardcopy factors, 379, 640
 hardcopy filename, 1156
 hardcopy format, 1155
 hardcopy formfeed, 380
 hardcopy grayscale, 1157
 hardcopy invert graticule colors, 381, 1158
 hardcopy layout, 382
 hardcopy palette, 389
 hardcopy print, area, 377
 hardcopy printer driver, 1159
 Hardware Event Condition Register (:HWERegister:CONDITION), 207
 Hardware Event Condition Register (:OPERegister:CONDITION), 1217
 Hardware Event Enable Register (HWEenable), 205
 Hardware Event Event Register (:HWERegister[:EVENT]), 208, 1216
 head type, probe, 278
 header, 1233
 high pass filter math function, 365
 high resolution acquisition type, 1047
 high trigger level, 966
 high-frequency reject filter, 963, 987
 high-level voltage, waveform generator, 1119
 high-pass filter cutoff frequency, 358
 high-resolution acquisition type, 232
 hold time, setup and hold trigger, 1019
 hold until operation complete, 176
 holdoff time, 964
 holes in waveform data, 1053
 hop frequency, waveform generator FSK modulation, 1101
 horizontal adjustment, fine (vernier), 955
 horizontal position, 956
 horizontal scale, 954, 958
 horizontal scaling, 1060
 horizontal time, 951, 957, 1162
 Host name softkey, 48
 hostname, 48
 hub address value, USB search, 924
 hub address value, USB trigger, 825
 hubs, USB 2.0 signal quality, 289
 HWEenable (Hardware Event Enable Register), 205
 HWERegister:CONDITION (Hardware Event Condition Register), 207, 1217
 HWERegister[:EVENT] (Hardware Event Event Register), 208, 1216

I

i1080L50HZ, 1027, 1031
 i1080L60HZ, 1027, 1031

I2C demo signal, 301
 I2S alignment, 723
 I2S audio channel, 732
 I2S clock slope, 725
 I2S CLOCK source, 727
 I2S DATA source, 728
 I2S demo signal, 302
 I2S pattern data, 733
 I2S pattern format, 735
 I2S range, 736
 I2S receiver width, 726
 I2S SEARCh commands, 883
 I2S serial bus commands, 721
 I2S serial decode base, 724
 I2S serial search, audio channel, 884
 I2S serial search, data, 886
 I2S serial search, format, 887
 I2S serial search, mode, 885
 I2S serial search, range, 888
 I2S transmit word size, 738
 I2S trigger operator, 730
 I2S triggering, 657
 I2S word select (WS) low, 739
 I2S word select (WS) source, 729
 id mode, 698
 ID pattern, CAN trigger, 697
 identification number, 174
 identification of options, 177
 identifier, LIN, 759
 idle, 980
 idle until operation complete, 176
 IDN (Identification Number), 174
 IEC 61000-3-2 standard for current harmonics analysis, 572
 IEEE 488.2 standard, 168
 IIC address, 744
 IIC clock, 742
 IIC data, 743, 745
 IIC data 2, 746
 IIC SEARCh commands, 889
 IIC serial decode address field size, 741
 IIC serial search, address, 892
 IIC serial search, data, 893
 IIC serial search, data2, 894
 IIC serial search, mode, 890
 IIC serial search, qualifier, 895
 IIC trigger commands, 740
 IIC trigger qualifer, 747
 IIC trigger type, 748
 IIC triggering, 658
 image format, 641
 image invert graticule colors, 642
 image memory, 227
 image palette, 643
 image, save, 639
 image, save with inksaver, 642
 impedance, 273
 infinity representation, 1239
 initial load current, transient response analysis, 620
 initialization, 56, 59
 initialize, 180, 939

initialize label list, 324
 initiate acquisition, 203
 inksaver, save image with, 642
 input coupling for channels, 271
 input for integrate, DC offset correction, 360
 input impedance for channels, 273, 1144
 input inversion for specified channel, 274
 input power, 506
 inrush current, 510
 inrush current analysis, 575, 576, 577
 inrush current expected, 600
 insert label, 275
 installed options identified, 177
 instruction header, 1233
 instrument number, 174
 instrument options identified, 177
 instrument requests service, 187
 instrument serial number, 225
 instrument settings, 376
 instrument status, 66
 instrument type, 174
 integrate DC offset correction, 360
 integrate math function, 364, 1065
 INTegrate source for function, 1153
 internal low-pass filter, 270, 338
 introduction to :ACQuire commands, 231
 introduction to :BUS<n> commands, 248
 introduction to :CALibrate commands, 258
 introduction to :CHANnel<n> commands, 269
 introduction to :DEMO commands, 299
 introduction to :DIGItal<d> commands, 308
 introduction to :DISPlay commands, 316
 introduction to :EXTernal commands, 337
 introduction to :FUNCTION commands, 346
 introduction to :HARDcopy commands, 376
 introduction to :LISTer commands, 393
 introduction to :MARKer commands, 398
 introduction to :MEASure commands, 428
 introduction to :POD<n> commands, 551
 introduction to :RECall commands, 624
 introduction to :SAVE commands, 635
 introduction to :SBUS commands, 657
 introduction to :SYSTem commands, 934
 introduction to :TIMEbase commands, 948
 introduction to :TRIGger commands, 959
 introduction to :WAVeform commands, 1045
 introduction to :WGEN<w> commands, 1082
 introduction to :WMEMory<r> commands, 1123
 introduction to common (*) commands, 168
 introduction to root () commands, 194
 invert graticule colors for hardcopy, 381, 1158
 invert graticule colors for image, 642
 inverted masks, bind levels, 544
 inverting input for channels, 274
 IO library, referencing, 58
 IP address, 48
 IronPython, SCPI.NET example, 1354

K

key disable, 938

key press detected in Standard Event Status Register, [172](#)
knob disable, [938](#)
known state, [180](#), [939](#)

L

label, [1141](#)
label command, bus, [254](#)
label list, [275](#), [324](#)
label reference waveforms, [1127](#)
label, ARINC 429 trigger, [673](#), [677](#), [862](#)
label, digital channel, [310](#)
labels, [275](#), [323](#), [324](#)
labels to store calibration information, [260](#)
labels, specifying, [316](#)
LAN interface, [47](#), [50](#)
LAN Settings softkey, [48](#)
landscape layout for hardcopy, [382](#)
language for program examples, [55](#)
layout for hardcopy, [382](#)
leakage into peak spectrum, [357](#)
learn string, [175](#), [943](#)
least significant byte first, [1051](#)
left reference, [953](#)
legal values for channel offset, [276](#)
legal values for frequency span, [355](#)
legal values for offset, [363](#), [368](#)
length for waveform data, [652](#)
less than qualifier, [996](#)
less than time, [993](#), [997](#), [1006](#), [1009](#)
less than value for glitch search, [846](#)
level for trigger voltage, [986](#), [994](#)
LF coupling, [985](#)
license information, [177](#)
limits for line number, [1027](#)
LIN acknowledge, [753](#)
LIN baud rate, [754](#)
LIN demo signal, [301](#)
LIN identifier, [759](#)
LIN pattern data, [760](#)
LIN pattern format, [763](#)
LIN SEARch commands, [896](#)
LIN serial decode bus parity bits, [752](#)
LIN serial search, data, [899](#)
LIN serial search, data format, [901](#)
LIN serial search, data length, [900](#)
LIN serial search, frame ID, [897](#)
LIN serial search, mode, [898](#)
LIN source, [755](#)
LIN standard, [756](#)
LIN sync break, [757](#)
LIN trigger, [758](#), [762](#)
LIN trigger commands, [750](#)
LIN trigger definition, [1185](#)
LIN triggering, [658](#)
line frequency setting for current harmonics analysis, [569](#)
line glitch trigger source, [998](#)
line number for TV trigger, [1027](#)
line terminator, [161](#)
LINE trigger level, [986](#)

LINE trigger source, [989](#)
list of channel labels, [324](#)
LISTer commands, [393](#)
lister display, [395](#)
lister time reference, [396](#)
ln math function, [364](#)
load utilization (CAN), [686](#)
local lockout, [938](#)
lock, [938](#)
lock mask to signal, [535](#)
lock, analog channel protection, [942](#)
lockout message, [938](#)
log math function, [364](#)
logic level activity, [1140](#)
long form, [1234](#)
low frequency sine with glitch demo signal, [301](#)
low pass filter math function, [364](#)
low trigger level, [967](#)
lower threshold, [457](#)
lower threshold voltage for measurement, [1160](#)
lowercase characters in commands, [1233](#)
low-frequency reject filter, [987](#)
low-level voltage, waveform generator, [1120](#)
low-pass filter cutoff frequency, [359](#)
low-pass filter used to limit bandwidth, [270](#), [338](#)
LRN (Learn Device Setup), [175](#)
lsbfirst, [1051](#)

M

M1553 SEARch commands, [902](#)
M1553 trigger commands, [764](#)
M1553 trigger type, [770](#)
magnify math function, [365](#)
magnitude of occurrence, [479](#)
main sweep range, [956](#)
main time base, [1188](#)
main time base mode, [949](#)
making measurements, [428](#)
MAN option for probe sense, [1145](#), [1149](#)
manual cursor mode, [401](#)
MARKer commands, [397](#)
marker mode, [409](#)
marker position, [410](#)
marker readout, [1166](#), [1167](#)
marker set for voltage measurement, [1172](#), [1173](#)
marker sets start time, [1163](#)
marker time, [1162](#)
markers for delta voltage measurement, [1171](#)
markers track measurements, [467](#)
markers, command overview, [398](#)
markers, mode, [401](#)
markers, time at start, [1167](#)
markers, time at stop, [1166](#)
markers, X delta, [400](#), [406](#)
markers, X1 position, [402](#)
markers, X1Y1 source, [403](#)
markers, X2 position, [404](#)
markers, X2Y2 source, [405](#)

markers, Y delta, [411](#)
markers, Y1 position, [409](#)
markers, Y2 position, [410](#)
mask, [170](#), [184](#)
mask command, bus, [255](#)
mask statistics, reset, [529](#)
mask test commands, [517](#)
Mask Test Event Enable Register (MTEenable), [209](#)
mask test event event register, [211](#)
Mask Test Event Event Register (:MTERegister[:EVENT]), [211](#), [1218](#)
mask test run mode, [536](#)
mask test termination conditions, [536](#)
mask test, all channels, [522](#)
mask test, enable/disable, [534](#)
mask, delete, [533](#)
mask, get as binary block data, [532](#)
mask, load from binary block data, [532](#)
mask, lock to signal, [535](#)
mask, recall, [628](#)
mask, save, [644](#), [645](#)
masks, bind levels, [544](#)
master summary status bit, [187](#)
math filters, [364](#)
math function, stop displaying, [202](#)
math operators, [364](#)
math transforms, [364](#)
math visualizations, [365](#)
MAV (Message Available), [169](#), [185](#), [187](#)
maximum duration, [993](#), [1005](#), [1006](#)
maximum position, [950](#)
maximum range for zoomed window, [957](#)
maximum scale for zoomed window, [958](#)
maximum vertical value measurement, [484](#)
maximum vertical value, time of, [492](#), [1164](#)
maximum waveform data length, [653](#)
MEASure commands, [415](#)
measure mask test failures, [537](#)
measure overshoot, [454](#)
measure period, [457](#)
measure phase between channels, [458](#)
MEASure power commands, [495](#)
measure preshoot, [460](#)
measure start voltage, [1172](#)
measure stop voltage, [1173](#)
measure value at a specified time, [489](#)
measure value at top of waveform, [490](#)
measurement error, [428](#)
measurement record, [1058](#)
measurement setup, [428](#), [468](#)
measurement source, [468](#)
measurement statistics results, [462](#)
measurement statistics, display on/off, [471](#)
measurement trend math function, [365](#)
measurement window, [491](#)
measurements, AC RMS, [446](#), [488](#)
measurements, area, [431](#), [441](#)
measurements, average value, [443](#), [482](#)
measurements, base value, [444](#), [483](#)
measurements, built-in, [57](#)
measurements, burst width, [432](#)

- measurements, clear, 433, 1161
measurements, command overview, 428
measurements, counter, 434
measurements, DC RMS, 446, 488
measurements, definition setup, 437
measurements, delay, 439
measurements, fall time, 448
measurements, falling edge count, 451
measurements, falling pulse count, 452
measurements, frequency, 449
measurements, how autoscale affects, 197
measurements, lower threshold level, 1160
measurements, maximum vertical value, 484
measurements, maximum vertical value, time of, 492, 1164
measurements, minimum vertical value, 485
measurements, minimum vertical value, time of, 493, 1165
measurements, negative duty cycle, 450
measurements, overshoot, 454
measurements, period, 457
measurements, phase, 458
measurements, positive duty cycle, 447
measurements, preshoot, 460
measurements, pulse width, negative, 453
measurements, pulse width, positive, 461
measurements, ratio of AC RMS values, 487
measurements, rise time, 465
measurements, rising edge count, 456
measurements, rising pulse count, 459
measurements, show, 467
measurements, snapshot all, 430
measurements, source channel, 468
measurements, standard deviation, 466
measurements, start marker time, 1166
measurements, stop marker time, 1167
measurements, thresholds, 1163
measurements, time between start and stop markers, 1162
measurements, time between trigger and edge, 476
measurements, time between trigger and vertical value, 479
measurements, time between trigger and voltage level, 1168
measurements, upper threshold value, 1170
measurements, vertical amplitude, 442, 481
measurements, vertical peak-to-peak, 445, 486
measurements, voltage difference, 1171
memory setup, 183, 943
menu, display, 325
menu, system, 1187
message available bit, 187
message available bit clear, 169
message displayed, 187
message error, 1191
message queue, 1208
message, CAN symbolic search, 874
message, CAN symbolic trigger, 699
messages ready, 187
midpoint of thresholds, 457
MIL-STD-1553 demo signal, 302
MIL-STD-1553 serial decode base, 766
MIL-STD-1553 serial search, data, 904
MIL-STD-1553 serial search, mode, 903
MIL-STD-1553 serial search, Remote Terminal Address, 905
MIL-STD-1553, dual demo signal, 302
minimum duration, 992, 1005, 1006, 1009
minimum vertical value measurement, 485
minimum vertical value, time of, 493, 1165
MISO data pattern width, 782
MISO data pattern, SPI trigger, 781
MISO data source, SPI trigger, 779
MISO data, SPI, 1069
mixed-signal demo signals, 301
mixed-signal oscilloscopes, 6
mnemonics, duplicate, 1237
mode, 401, 949, 1028
mode, serial decode, 661
model number, 174
models, oscilloscope, 3
modes for triggering, 968
Modify softkey, 48
modulating signal frequency, waveform generator, 1098, 1100
modulation (waveform generator), enabling/disabling, 1106
modulation analysis, 578
modulation analysis source (voltage or current), 579
modulation analysis, type of, 580
modulation type, waveform generator, 1107
MOSI data pattern width, 784
MOSI data pattern, SPI trigger, 783
MOSI data source, SPI trigger, 780, 1186
most significant byte first, 1051
move cursors, 1166, 1167
msbfirrst, 1051
MSG (Message), 185, 187
MSO models, 6
MSS (Master Summary Status), 187
MTEenable (Mask Test Event Enable Register), 209
MTERegister[:EVENTn] (Mask Test Event Event Register), 211, 1218
MTESet commands, 517
multi-channel waveform data, save, 646
multiple commands, 1237
multiple queries, 65
multiply math function, 364, 1065
multiply math function as g(t) source, 1150
- negative pulse width measurement, 57
negative pulse width, power modulation analysis, 580
negative slope, 774, 988
negative slope, Nth edge in burst, 981
negative TV trigger polarity, 1029
network domain password, 386
network domain user name, 388
network printer address, 383
network printer domain, 385
network printer slot, 387
network printer, apply connection settings, 384
new line (NL) terminator, 161, 1234
new load current, transient response analysis, 621
NL (new line) terminator, 161, 1234
NMONotonic, dual demo signal, 302
noise floor, 614, 617
noise reject filter, 969
noise waveform generator output, 1092
noise, adding to waveform generator output, 1105
noisy sine waveform demo signal, 300
non-core commands, 1232
non-interlaced GENeric mode, 1031
non-volatile memory, label list, 254, 310, 324
normal acquisition type, 232, 1046
normal trigger sweep mode, 959
notices, 2
NR1 number format, 161
NR3 number format, 161
Nth edge burst trigger source, 982
Nth edge burst triggering, 960
Nth edge in a burst idle, 980
Nth edge in burst slope, 981
Nth edge of burst counter, 979
Nth edge of Edge Then Edge trigger, 975
NTSC, 1027, 1031
null frame count (FlexRay), 707
null offset, 614
NULL string, 936
number format, 161
number of points, 236, 1056, 1058
number of time buckets, 1056, 1058
numeric variables, 64
numeric variables, reading query results into multiple, 66
nwidth, 453
- ## N
- N2820A high sensitivity current probe, 36, 441, 442, 443, 444, 445, 446
N8900A InfiniiView oscilloscope analysis software, 646
name channels, 275
name list, 324
natural logarithm math function, 364
negative glitch trigger polarity, 995
negative pulse width, 453
- ## O
- obsolete and discontinued commands, 1133
obsolete commands, 1232
occurrence reported by magnitude, 1168
offset, 346
offset for Ax + B math operation, 362
offset value for channel voltage, 276
offset value for selected function, 363, 368
offset, waveform generator, 1121
one values in waveform data, 1053
OPC (Operation Complete) command, 176
OPC (Operation Complete) status bit, 170, 172

OPEE (Operation Status Enable Register), 213
 Open method, 59
 operating configuration, 175, 943
 operating state, 183
 operation complete, 176
 operation status condition register, 215
 Operation Status Condition Register
 (:OPERegister:CONDITION), 215, 1213
 operation status conditions occurred, 187
 Operation Status Enable Register (OPEE), 213
 operation status event register, 217
 Operation Status Event Register
 (:OPERegister[:EVENT]), 217, 1212
 operations for function, 364
 operators, math, 364
 OPERegister:CONDITION (Operation Status Condition Register), 215, 1213
 OPERegister[:EVENT] (Operation Status Event Register), 217, 1212
 OPT (Option Identification), 177
 optional syntax terms, 161
 options, 177
 OR trigger commands, 999
 order of digital channels on display, 1147
 order of output, 1051
 oscilloscope connection, opening, 59
 oscilloscope connection, verifying, 49
 oscilloscope external trigger, 337
 oscilloscope models, 3
 oscilloscope rate, 243
 oscilloscope, connecting, 47
 oscilloscope, initialization, 56
 oscilloscope, operation, 6
 oscilloscope, program structure, 56
 oscilloscope, setting up, 47
 oscilloscope, setup, 60
 output control, demo signals, 305
 output control, waveform generator, 1109
 output load impedance, waveform generator, 1110
 output messages ready, 187
 output polarity, waveform generator, 1111
 output power, 509
 output queue, 176, 1207
 output queue clear, 169
 output ripple, 514
 output ripple analysis, 590
 output sequence, 1051
 overall pass/fail status for current harmonics analysis, 573
 overlapped commands, 1240
 overload, 282
 Overload Event Enable Register (OVL), 219
 Overload Event Register (:OVLRegister), 1215
 Overload Event Register (OVLR), 221
 overload frame count (CAN), 683
 overload protection, 219, 221
 overshoot of waveform, 454
 overshoot percent for transient response analysis, 601
 overvoltage, 282
 OVL (Overload Event Enable Register), 219

OVLR (Overload Event Register), 221
 OVLR bit, 215, 217
 OVLRegister (Overload Event Register), 1215

P

P1080L24HZ, 1027, 1031
 P1080L25HZ, 1027, 1031
 P1080L50HZ, 1027, 1031
 P1080L60HZ, 1027, 1031
 P480L60HZ, 1027, 1031
 P720L60HZ, 1027, 1031
 PAL, 1027, 1031
 palette for hardcopy, 389
 palette for image, 643
 PAL-M, 1027, 1031
 parameters for delay measurement, 439
 parametric measurements, 428
 parity, 798
 parity bits, LIN serial decode bus, 752
 parser, 194, 1237
 pass, self test, 189
 pass/fail status (overall) for current harmonics analysis, 573
 password, network domain, 386
 path information, recall, 629
 path information, save, 648
 pattern, 744, 745, 746
 pattern data, I2S, 733
 pattern data, LIN, 760
 pattern duration, 992, 993, 1005, 1006
 pattern for pattern trigger, 1002
 pattern format, I2S, 735
 pattern format, LIN, 763
 pattern length, 696, 762
 PATTern trigger commands, 1001
 pattern trigger format, 1004
 pattern trigger qualifier, 1007
 pattern triggering, 960
 pattern width, 782, 784
 peak current, 510
 peak data, 1047
 peak detect, 244
 peak detect acquisition type, 232, 1047
 peak-to-peak vertical value measurement, 445, 486
 pending operations, 176
 percent of waveform overshoot, 454
 percent thresholds, 437
 period measured to calculate phase, 458
 period measurement, 57, 428, 457
 Period, power modulation analysis, 580
 period, waveform generator, 1112
 persistence, waveform, 316, 327
 phase angle, demo signals, 304
 phase measured between channels, 458
 phase measurements, 476
 phase measurements with X cursors, 407
 phase shifted demo signals, 300
 PID check value, USB trigger, 826
 PLL Locked bit, 207
 PNG format screen image data, 322

pod, 551, 553, 554, 555, 1065, 1142
 POD commands, 551
 POD data format, 1050
 pod, stop displaying, 202
 points, 236, 1056, 1058
 points in waveform data, 1046
 polarity, 799, 1029
 polarity for glitch search, 847
 polarity for glitch trigger, 995
 polarity, runt search, 852
 polarity, runt trigger, 1011
 polling synchronization with timeout, 1224
 polling wait, 1222
 PON (Power On) status bit, 170, 172
 port value, USB search, 929
 port value, USB trigger, 831
 portrait layout for hardcopy, 382
 position, 311, 404, 950, 956
 position cursors, 1166, 1167
 position in zoomed view, 956
 position waveforms, 1147
 positive glitch trigger polarity, 995
 positive pulse width, 461
 positive pulse width measurement, 57
 positive pulse width, power modulation analysis, 580
 positive slope, 774, 988
 positive slope, Nth edge in burst, 981
 positive TV trigger polarity, 1029
 positive width, 461
 power analysis, enabling, 564
 POWer commands, 557
 power factor, 505
 power factor for IEC 61000-3-2 Standard Class C, 570
 power loss, 511
 power phase angle, 499
 power quality analysis, 589
 power supply rejection ratio (PSRR), 585, 586, 587, 588
 preamble data, 1060
 preamble metadata, 1045
 predefined logic threshold, 1142
 predefined threshold voltages, 1189
 present working directory, recall operations, 629
 present working directory, save operations, 648
 preset conditions, 939
 preshoot measured on waveform, 460
 previously stored configuration, 179
 print command, 223
 print job, start, 391
 print mask test failures, 538
 print query, 1182
 printer driver for hardcopy, 1159
 printer, active, 378
 printing, 376
 printing in grayscale, 1157
 probe, 986
 probe attenuation affects channel voltage range, 283
 probe attenuation factor (external trigger), 339

probe attenuation factor for selected channel, 277
probe head type, 278
probe ID, 279
probe sense for oscilloscope, 1145, 1149
probe skew value, 280, 1143
process sigma, mask test run, 541
program data, 1234
program data syntax rules, 1236
program initialization, 56
program message, 59, 168
program message syntax, 1233
program message terminator, 1234
program structure, 56
programming examples, 6, 1241
protecting against calibration, 262
protection, 219, 221, 282
protection lock, 942
pulse waveform generator output, 1092
pulse width, 453, 461
pulse width duration trigger, 992, 993, 997
pulse width measurement, 57, 428
pulse width trigger, 969
pulse width trigger level, 994
pulse width triggering, 960
pulse width, waveform generator, 1094
pwidth, 461
Python, VISA COM example, 1268
Python, VISA example, 1315

Q

qualifier, 997
qualifier for glitch search, 848
qualifier, runt search, 853
qualifier, runt trigger, 1012
qualifier, transition search, 857
qualifier, transition trigger, 1022
qualifier, trigger duration, 1005, 1006
qualifier, trigger pattern, 1007
queries, multiple, 65
query error detected in Standard Event Status, 172
query responses, block data, 64
query responses, reading, 63
query results, reading into numeric variables, 64
query results, reading into string variables, 64
query return values, 1239
query setup, 376, 398, 428, 943
query subsystem, 248, 308
querying setup, 269
querying the subsystem, 960
queues, clearing, 1219
quick reference, commands, 69
quoted ASCII string, 162
QYE (Query Error) status bit, 170, 172

R

ramp symmetry, waveform generator, 1095

ramp symmetry, waveform generator modulating signal, 1104
ramp waveform generator output, 1091
range, 346, 957
range for channels, 283
range for duration trigger, 1009
range for external trigger, 340
range for full-scale vertical axis, 367
range for glitch search, 849
range for glitch trigger, 997
range for time base, 951
range of offset values, 276
range qualifier, 996
range, I2S, 736
ranges, value, 162
rate, 243
ratio measurements with X cursors, 407
ratio measurements with Y cursors, 412
ratio of AC RMS values measured between channels, 487
Ratio, power modulation analysis, 580
raw acquisition record, 1058
RCL (Recall), 179
Rds (dynamic ON resistance) waveform, 613
Rds(on) value for conduction calculation, 615
reactive power, 512
read configuration, 175
ReadIEEEBlock method, 59, 63, 65
ReadList method, 59, 63
ReadNumber method, 59, 63
readout, 1162
ReadString method, 59, 63
real power, 513
real-time acquisition mode, 235
recall, 179, 624, 943
recall arbitrary waveform, 625
recall CAN symbolic data, 626
RECall commands, 623
recall filename, 627, 1090
recall mask, 628
recall path information, 629
recall reference waveform, 631
recall setup, 630
recalling and saving data, 316
receiver width, I2S, 726
RECTangular window for transient signals, 357
reference, 346, 953
reference clock, 952
reference for time base, 1188
reference signal (10 MHz), 237
reference signal mode, 952
reference waveform save source, 655
reference waveform, recall, 631
reference waveform, save, 656
reference waveforms, clear, 1125
reference waveforms, display, 1126
reference waveforms, label, 1127
reference waveforms, save to, 1128
reference waveforms, skew, 1129
reference waveforms, Y offset, 1130
reference waveforms, Y range, 1131
reference waveforms, Y scale, 1132
reference, lister, 396
registers, 172, 179, 183, 196, 209, 211, 213, 215, 217, 219, 221
registers, clearing, 1219
reject filter, 987
reject high frequency, 963
reject noise, 969
relative standard deviation, 475
remote control examples, 1241
Remote Terminal Address (RTA), M1553 trigger, 769
remove cursor information, 401
remove labels, 323
remove message from display, 936
reorder channels, 197
repetitive acquisitions, 224
report errors, 937
report transition, 476, 479
reporting status, 1199
reporting the setup, 960
request service, 187
Request-for-OPC flag clear, 169
reset, 180
reset conditions, 180
reset defaults, waveform generator, 1113
reset mask statistics, 529
reset measurements, 321
resolution of printed copy, 1157
resource session object, 59
ResourceManager object, 59
restore configurations, 175, 179, 183, 943
restore labels, 323
restore setup, 179
return values, query, 1239
returning acquisition type, 244
returning number of data points, 236
RF burst demo signal, 301
right reference, 953
ringing pulse demo signal, 300
ripple (output) analysis, 590
ripple, output, 514
rise time measurement, 428
rise time of positive edge, 465
Rise Time, power modulation analysis, 580
rising edge count measurement, 456
rising pulse count measurement, 459
RMS - AC, power modulation analysis, 580
RMS value measurement, 446, 488
roll time base mode, 949
root () commands, 191, 194
root level commands, 3
RQL (Request Control) status bit, 170, 172
RQS (Request Service), 187
RS-232/UART triggering, 658
RST (Reset), 180
rules, tree traversal, 1237
rules, truncation, 1234
run, 188, 224
Run bit, 215, 217
run count, current harmonics analysis, 571
run mode, mask test, 536
run, USB 2.0 signal quality test, 290

running configuration, 183, 943
 RUnT SEARch commands, 851
 runt search polarity, 852
 runt search qualifier, 853
 runt search source, 854
 runt search, pulse time, 855
 RUnT trigger commands, 1010
 runt trigger polarity, 1011
 runt trigger qualifier, 1012
 runt trigger source, 1013
 runt trigger time, 1014
 Rx frame count (UART), 795
 Rx source, 800

S

S and E or U value, USB search, 931
 S and E or U value, USB trigger, 833
 sample rate, 3, 243
 sampled data, 1146
 sampled data points, 1053
 SAV (Save), 183
 save, 183, 635
 save arbitrary waveform, 636
 SAVE commands, 633
 save current harmonics analysis results, 647
 save filename, 638
 save image, 639
 save image with inksaver, 642
 save mask, 644, 645
 save mask test failures, 539
 save path information, 648
 save reference waveform, 656
 save setup, 649
 save to reference waveform location, 1128
 save waveform data, 650
 saved image, area, 1184
 saving and recalling data, 316
 SBUS A429 commands, 662
 SBUS CAN commands, 680
 SBUS commands, 657
 SBUS I2S commands, 721
 SBUS-<n> commands, general, 659
 SC value, USB search, 930
 SC value, USB trigger, 832
 scale, 369, 954, 958
 scale factors output on hardcopy, 379, 640
 scale for channels, 284
 scale units for channels, 285
 scale units for external trigger, 341
 scaling display factors, 277
 SCPI commands, 67
 SCPI.NET example in C#, 1342
 SCPI.NET example in IronPython, 1354
 SCPI.NET example in Visual Basic .NET, 1348
 SCPI.NET examples, 1342
 scratch measurements, 1161
 screen area for hardcopy print, 377
 screen area for saved image, 1184
 screen image data, 322
 SDI pattern, ARINC 429 trigger, 675, 865
 SEARch commands, 835

SEARch commands, A429, 861
 SEARch commands, CAN, 867
 SEARch commands, EDGE, 841
 SEARch commands, FLEXray, 877
 SEARch commands, general, 836
 SEARch commands, GLItch, 844
 SEARch commands, I2S, 883
 SEARch commands, IIC, 889
 SEARch commands, LIN, 896
 SEARch commands, M1553, 902
 SEARch commands, RUnT, 851
 SEARch commands, SPI, 906
 SEARch commands, TRANSition, 856
 SEARch commands, UART, 910
 SEARch commands, USB, 914
 search for found event, 838
 search mode, 839
 search state, 840
 search, edge slope, 842
 search, edge source, 843
 search, USB, 916
 SECAM, 1027, 1031
 seconds per division, 954
 segmented waveform save option, 654
 segments, analyze, 238
 segments, count of waveform, 1063
 segments, setting number of memory, 239
 segments, setting the index, 240
 segments, time tag, 1064
 select measurement channel, 468
 self-test, 189
 sensing a channel probe, 1145
 sensing a external trigger probe, 1149
 sensitivity of oscilloscope input, 277
 sequential commands, 1240
 serial clock, 742, 777
 serial data, 743
 serial decode bus, 657
 serial decode bus display, 660
 serial decode mode, 661
 serial frame, 778
 serial number, 225
 service request, 187
 Service Request Enable Register (SRE), 185, 1205
 set center frequency, 354
 set cursors, 1166, 1167
 set date, 935
 set time, 945
 set up oscilloscope, 47
 setting digital display, 309
 setting digital label, 254, 310
 setting digital position, 311
 setting digital threshold, 313
 setting display, 353
 setting external trigger level, 337
 setting impedance for channels, 273
 setting inversion for channels, 274
 setting pod display, 553
 setting pod size, 554
 setting pod threshold, 555
 settings, 179, 183

settings, instrument, 376
 setup, 232, 248, 269, 308, 316, 376, 943
 setup and hold trigger clock source, 1017
 setup and hold trigger data source, 1018
 setup and hold trigger hold time, 1019
 setup and hold trigger setup time, 1020
 setup and hold trigger slope, 1016
 setup configuration, 179, 183, 943
 setup defaults, 180, 939
 setup memory, 179
 setup reported, 960
 setup time, setup and hold trigger, 1020
 setup, recall, 630
 setup, save, 649
 shape of modulation signal, waveform generator, 1103
 SHOЛd trigger commands, 1015
 short form, 5, 1234
 show channel labels, 323
 show measurements, 428, 467
 SICL example in C, 1322
 SICL example in Visual Basic, 1331
 SICL examples, 1322
 sidebar, display, 326
 sigma, mask test run, 541
 signal speed, ARINC 429, 672
 signal type, 281
 signal type, ARINC 429, 670
 signal value, CAN symbolic search, 876
 signal value, CAN symbolic trigger, 701
 signal, CAN symbolic search, 875
 signal, CAN symbolic trigger, 700
 signed data, 1049
 simple command headers, 1235
 sine cardinal waveform generator output, 1092
 sine waveform demo signal, 300
 sine waveform generator output, 1091
 single acquisition, 226
 single-ended probe heads, 278
 single-ended signal type, 281
 single-shot demo signal, 300
 single-shot DUT, synchronizing with, 1226
 size, 554
 size, digital channels, 312
 skew, 280, 1143
 skew reference waveform, 1129
 slew rate power analysis, 610
 slope, 774, 988
 slope (direction) of waveform, 1168
 slope not valid in TV trigger mode, 988
 slope parameter for delay measurement, 439
 slope, arming edge, Edge Then Edge trigger, 972
 slope, Nth edge in burst, 981
 slope, setup and hold trigger, 1016
 slope, transition search, 858
 slope, transition trigger, 1023
 slope, trigger edge, Edge Then Edge trigger, 976
 slot, network printer, 387
 smoothing acquisition type, 1047
 snapshot all measurement, 430

- software version, 174
 source, 468, 671, 691, 755
 source (voltage or current) for slew rate analysis, 611
 source channel, M1553, 767
 source for function, 1153
 source for glitch search, 850
 source for Nth edge burst trigger, 982
 source for trigger, 989
 source for TV trigger, 1030
 source input for function, first, 370
 source input for function, second, 372
 source, arming edge, Edge Then Edge trigger, 973
 source, automask, 524
 source, FLEXray, 711
 source, mask test, 549
 source, runt search, 854
 source, runt trigger, 1013
 source, save reference waveform, 655
 source, transition trigger, 859, 1024
 source, trigger edge, Edge Then Edge trigger, 977
 source, waveform, 1065
 span, 364
 span of frequency on display, 355
 special PID, USB search, 927
 special PID, USB trigger, 829
 specify measurement, 468
 speed of ARINC 429 signal, 672
 speed, USB decode, 816
 SPI, 774
 SPI clock timeout, 775
 SPI decode bit order, 773
 SPI decode word width, 786
 SPI demo signal, 302
 SPI MISO data, 1069
 SPI SEARch commands, 906
 SPI serial search, data, 908
 SPI serial search, data width, 909
 SPI serial search, mode, 907
 SPI trigger, 776, 782, 784
 SPI trigger clock, 777
 SPI trigger commands, 771
 SPI trigger frame, 778
 SPI trigger MISO data pattern, 781
 SPI trigger MOSI data pattern, 783
 SPI trigger type, 785
 SPI trigger, MISO data source, 779
 SPI trigger, MOSI data source, 780, 1186
 SPI triggering, 658
 square math function, 364
 square root math function, 364
 square wave duty cycle, waveform generator, 1096
 square waveform generator output, 1091
 SRE (Service Request Enable Register), 185, 1205
 SRQ (Service Request interrupt), 205, 209, 213
 SSM pattern, ARINC 429 trigger, 676, 866
 standard deviation measured on waveform, 466
 Standard Event Status Enable Register (ESE), 170, 1210
 Standard Event Status Register (ESR), 172, 1209
 standard for video, 1031
 standard, LIN, 756
 start acquisition, 188, 203, 224, 226
 start and stop edges, 437
 start cursor, 1166
 start measurement, 428
 start print job, 391
 start time, 997, 1166
 start time marker, 1163
 state memory, 183
 state of instrument, 175, 943
 statistics increment, 472
 statistics reset, 474
 statistics results, 462
 statistics, max count, 473
 statistics, relative standard deviation, 475
 statistics, type of, 470
 status, 186, 227, 229
 Status Byte Register (STB), 184, 186, 187, 1203
 status data structure clear, 169
 status registers, 66
 status reporting, 1199
 STB (Status Byte Register), 184, 186, 187, 1203
 steady state output voltage expected, 605, 606, 607
 step size for frequency span, 355
 stop, 203, 228
 stop acquisition, 228
 stop cursor, 1167
 stop displaying channel, 202
 stop displaying math function, 202
 stop displaying pod, 202
 stop on mask test failure, 540
 stop time, 997, 1167
 storage, 183
 store instrument setup, 175, 183
 store setup, 183
 storing calibration information, 260
 string variables, 64
 string variables, reading multiple query results into, 65
 string variables, reading query results into multiple, 65
 string, quoted ASCII, 162
 subnet mask, 48
 subsource, waveform source, 1069
 subsystem commands, 3, 1237
 subtract math function, 364, 1065
 subtract math function as g(t) source, 1150
 sweep mode, trigger, 959, 970
 sweep speed set to fast to measure fall time, 448
 sweep speed set to fast to measure rise time, 465
 switch disable, 938
 switching level, current, 614
 switching level, voltage, 617
 switching loss per cycle, 501
 switching loss power analysis, 612
 sync break, LIN, 757
 sync frame count (FlexRay), 709
 syntax elements, 161
 syntax rules, program data, 1236
 syntax, optional terms, 161
 syntax, program message, 1233
 SYStem commands, 933
 system commands, 935, 936, 937, 938, 943, 945
 system commands introduction, 934

T

- table of current harmonics results, 567
 tdelta, 1162
 tedge, 476
 telnet ports 5024 and 5025, 1053
 Telnet sockets, 67
 temporary message, 936
 TER (Trigger Event Register), 229, 1206
 termination conditions, mask test, 536
 test sigma, mask test run, 541
 test, self, 189
 test, USB 2.0 signal quality, 295
 text, writing to display, 936
 THD (total harmonics distortion), 574
 threshold, 313, 555, 1142, 1189
 threshold voltage (lower) for measurement, 1160
 threshold voltage (upper) for measurement, 1170
 thresholds, 437, 1163
 thresholds used to measure period, 457
 thresholds, how autoscale affects, 197
 time base, 949, 950, 951, 953, 954, 1188
 time base commands introduction, 948
 time base reset conditions, 181, 940
 time base window, 956, 957, 958
 time between points, 1162
 time buckets, 233, 234
 time delay, 1188
 time delay, Edge Then Edge trigger, 974
 time delta, 1162
 time difference between data points, 1073
 time duration, 997, 1005, 1006, 1009
 time holdoff for trigger, 964
 time interval, 476, 479, 1162
 time interval between trigger and occurrence, 1168
 time marker sets start time, 1163
 time measurements with X cursors, 407
 time per division, 951
 time record, 357
 time reference, lister, 396
 time specified, 489
 time, calibration, 266
 time, mask test run, 542
 time, runt pulse search, 855
 time, runt trigger, 1014

- time, start marker, 1166
 time, stop marker, 1167
 time, system, 945
 time, transition search, 860
 time, transition trigger, 1025
 time/div, how autoscale affects, 197
 time-at-max measurement, 1164
 time-at-min measurement, 1165
 TIMEbase commands, 947
 timebase vernier, 955
 TIMEbase:MODE, 62
 time-ordered label list, 324
 timeout, SPI clock, 775
 timing measurement, 428
 title channels, 275
 title, mask test, 550
 token PID, USB search, 928
 token PID, USB trigger, 830
 tolerance, automask, 526, 527
 top of waveform value measured, 490
 total frame count (CAN), 685
 total frame count (FlexRay), 710
 total harmonics distortion (THD), 574
 total waveforms in mask test, 531
 touchscreen on/off, 946
 trace memory, 227
 track measurements, 467
 tracking, dual channel waveform generator, 1114
 trademarks, 2
 transfer instrument state, 175, 943
 transforms, math, 364
 transient response, 515
 transient response analysis, 618, 619, 622
 TRANSition SEARch commands, 856
 transition search qualifier, 857
 transition search slope, 858
 transition search time, 860
 transition trigger qualifier, 1022
 transition trigger slope, 1023
 transition trigger source, 859, 1024
 transition trigger time, 1025
 transmit word size, I2S, 738
 tree traversal rules, 1237
 trend measurement, 373
 TRG (Trigger), 185, 187, 188
 TRIG OUT BNC, 261
 trigger armed event register, 215, 217
 trigger burst, UART, 803
 trigger channel source, 998, 1030
 TRIGger commands, 959
 TRIGger commands, general, 961
 trigger data, UART, 804
 TRIGger DELay commands, 971
 trigger duration, 1005, 1006
 TRIGger EBURst commands, 978
 TRIGger EDGE commands, 983
 trigger edge coupling, 985
 trigger edge slope, 988
 trigger edge slope, Edge Then Edge trigger, 976
 trigger edge source, Edge Then Edge trigger, 977
 trigger event bit, 229
 Trigger Event Register (TER), 1206
 TRIGger FLEXray commands, 702
 TRIGger GLITch commands, 990
 trigger holdoff, 964
 trigger idle, UART, 805
 TRIGger IIC commands, 740
 trigger level auto set up, 965
 trigger level constants, 277
 trigger level voltage, 986
 trigger level, high, 966
 trigger level, low, 967
 TRIGger LIN commands, 750
 TRIGger M1553 commands, 764
 trigger occurred, 187
 TRIGger OR commands, 999
 TRIGger PATTern commands, 1001
 trigger pattern qualifier, 1007
 trigger qualifier, UART, 806
 trigger reset conditions, 181, 940
 TRIGger RUNT commands, 1010
 TRIGger SHOLD commands, 1015
 trigger SPI clock slope, 774
 TRIGger SPI commands, 771
 trigger status bit, 229
 trigger sweep mode, 959
 TRIGger TV commands, 1021, 1026
 trigger type, ARINC 429, 678, 863
 trigger type, SPI, 785
 trigger type, UART, 807
 TRIGger UART commands, 787
 TRIGger USB commands, 809
 TRIGger ZONE commands, 1036
 trigger, ARINC 429 source, 671
 trigger, CAN, 692
 trigger, CAN pattern data length, 696
 trigger, CAN pattern ID mode, 698
 trigger, CAN sample point, 688
 trigger, CAN signal baudrate, 689
 trigger, CAN signal definition, 690
 trigger, CAN source, 691
 trigger, duration greater than, 1005
 trigger, duration less than, 1006
 trigger, duration range, 1009
 trigger, edge coupling, 985
 trigger, edge level, 986
 trigger, edge reject, 987
 trigger, edge slope, 988
 trigger, edge source, 989
 trigger, FLEXray, 712
 trigger, FLEXray error, 713
 trigger, FLEXray event, 716
 trigger, force a, 962
 trigger, glitch greater than, 992
 trigger, glitch less than, 993
 trigger, glitch level, 994
 trigger, glitch polarity, 995
 trigger, glitch qualifier, 996
 trigger, glitch range, 997
 trigger, glitch source, 998
 trigger, high frequency reject filter, 963
 trigger, holdoff, 964
 trigger, I2S, 730
 trigger, I2S alignment, 723
 trigger, I2S audio channel, 732
 trigger, I2S clock slope, 725
 trigger, I2S CLOCKsource, 727
 trigger, I2S DATA source, 728
 trigger, I2S pattern data, 733
 trigger, I2S pattern format, 735
 trigger, I2S range, 736
 trigger, I2S receiver width, 726
 trigger, I2S transmit word size, 738
 trigger, I2S word select (WS) low, 739
 trigger, I2S word select (WS) source, 729
 trigger, IIC clock source, 742
 trigger, IIC data source, 743
 trigger, IIC pattern address, 744
 trigger, IIC pattern data, 745
 trigger, IIC pattern data 2, 746
 trigger, IIC qualifier, 747
 trigger, IIC signal baudrate, 754
 trigger, IIC type, 748
 trigger, LIN, 758
 trigger, LIN pattern data, 760
 trigger, LIN pattern data length, 762
 trigger, LIN pattern format, 763
 trigger, LIN sample point, 753
 trigger, LIN signal definition, 1185
 trigger, LIN source, 755
 trigger, mode, 968
 trigger, noise reject filter, 969
 trigger, Nth edge burst source, 982
 trigger, Nth edge in burst slope, 981
 trigger, Nth edge of burst count, 979
 trigger, Nth edge of Edge Then Edge trigger, 975
 trigger, SPI clock slope, 774
 trigger, SPI clock source, 777
 trigger, SPI clock timeout, 775
 trigger, SPI frame source, 778
 trigger, SPI framing, 776
 trigger, SPI pattern MISO width, 782
 trigger, SPI pattern MOSI width, 784
 trigger, sweep, 970
 trigger, threshold, 1189
 trigger, TV line, 1027
 trigger, TV mode, 1028, 1190
 trigger, TV polarity, 1029
 trigger, TV source, 1030
 trigger, TV standard, 1031
 trigger, UART base, 802
 trigger, UART baudrate, 791
 trigger, UART bit order, 792
 trigger, UART parity, 798
 trigger, UART polarity, 799
 trigger, UART Rx source, 800
 trigger, UART Tx source, 801
 trigger, UART width, 808
 trigger, USB, 817
 truncation rules, 1234
 TST (Self Test), 189

tstart, 1166
tstop, 1167
TTL threshold voltage for digital channels, 313, 1142
TTL trigger threshold voltage, 1189
turn function on or off, 1154
turn off channel, 202
turn off channel labels, 323
turn off digital pod, 202
turn off math function, 202
turn off time, 507, 584
turn on channel labels, 323
turn on channel number display, 1147
turn on time, 508, 584
turn on/turn off time analysis, 581, 582, 583, 584
turning channel display on and off, 272
turning off/on function calculation, 353
turning vectors on or off, 1146
TV mode, 1028, 1190
TV trigger commands, 1021, 1026
TV trigger line number setting, 1027
TV trigger mode, 1030
TV trigger polarity, 1029
TV trigger standard setting, 1031
TV triggering, 960
tvmode, 1190
Tx data, UART, 1069
Tx frame count (UART), 796
Tx source, 801
type, USB 2.0 signal quality test, 297

U

UART base, 802
UART baud rate, 791
UART bit order, 792
UART frame counters, reset, 794
UART parity, 798
UART polarity, 799
UART Rx source, 800
UART SEARch commands, 910
UART serial search, data, 911
UART serial search, data qualifier, 913
UART serial search, mode, 912
UART trigger burst, 803
UART trigger commands, 787
UART trigger data, 804
UART trigger idle, 805
UART trigger qualifier, 806
UART trigger type, 807
UART Tx data, 1069
UART Tx source, 801
UART width, 808
UART/RS232 demo signal, 301
UART/RS-232 triggering, 658
units (vertical) for FFT, 356
units per division, 284, 285, 341, 954
units per division (vertical) for function, 284, 369
units, automask, 525
units, X cursor, 407, 408

units, Y cursor, 412, 413
unsigned data, 1049
unsigned mode, 1071
upper threshold, 457
upper threshold voltage for measurement, 1170
uppercase characters in commands, 1233
URQ (User Request) status bit, 170, 172
USB (Device) interface, 47
USB 2.0 signal quality test results, save, 637
USB 2.0 triggering, 658
USB demo signal, 302
USB SEARch commands, 914
USB trigger commands, 809
USB triggering, 960
user defined channel labels, 275
user defined threshold, 1142
user event conditions occurred, 187
user name, network domain, 388
User's Guide, 6
user-defined threshold voltage for digital channels, 313
user-defined trigger threshold, 1189
USR (User Event bit), 185, 187
utilization, CAN bus, 686

V

valid command strings, 1233
valid pattern time, 1005, 1006
value, 479
value measured at base of waveform, 444, 483
value measured at specified time, 489
value measured at top of waveform, 490
value ranges, 162
values required to fill time buckets, 234
VBA, 58, 1242
Vce(sat) value for conduction calculation, 616
vectors turned on or off, 1146
vectors, display, 328
vectors, turning on or off, 316
vernier, channel, 286
vernier, horizontal, 955
vertical adjustment, fine (vernier), 286
vertical amplitude measurement, 442, 481
vertical axis defined by RANGE, 367
vertical axis range for channels, 283
vertical offset for channels, 276
vertical peak-to-peak measured on waveform, 445, 486
vertical scale, 284, 369
vertical scaling, 1060
vertical threshold, 1142
vertical units for FFT, 356
vertical value at center screen, 363, 368
vertical value maximum measured on waveform, 484
vertical value measurements to calculate overshoot, 454
vertical value minimum measured on waveform, 485
video line to trigger on, 1027

video standard selection, 1031
view, 230, 1072, 1147
view turns function on or off, 1154
VISA COM example in C#, 1251
VISA COM example in Python, 1268
VISA COM example in Visual Basic, 1242
VISA COM example in Visual Basic .NET, 1260
VISA example in C, 1275
VISA example in C#, 1294
VISA example in Python, 1315
VISA example in Visual Basic, 1284
VISA example in Visual Basic .NET, 1305
VISA examples, 1242, 1275
Visual Basic .NET, SCPI.NET example, 1348
Visual Basic .NET, VISA COM example, 1260
Visual Basic .NET, VISA example, 1305
Visual Basic 6.0, 59
Visual Basic for Applications, 58, 1242
Visual Basic, SICL library example, 1331
Visual Basic, VISA COM example, 1242
Visual Basic, VISA example, 1284
visualizations, math, 365
voltage crossing reported or not found, 1168
voltage difference between data points, 1076
voltage difference measured, 1171
voltage level for active trigger, 986
voltage marker used to measure waveform, 1172, 1173
voltage offset value for channels, 276
voltage probe, 285, 341
voltage ranges for channels, 283
voltage ranges for external trigger, 340
voltage source, 609
voltage threshold, 437
voltage, maximum expected input, 602, 603, 604

W

WAI (Wait To Continue), 190
wait, 190
wait for operation complete, 176
Wait Trig bit, 215, 217
waveform base value measured, 444, 483
WAveform command, 57
WAveform commands, 1043
waveform data, 1045
waveform data format, 651
waveform data length, 652
waveform data length, maximum, 653
waveform data, save, 650
waveform generator, 1082
waveform generator amplitude, 1118
waveform generator function, 1091
waveform generator high-level voltage, 1119
waveform generator low-level voltage, 1120
waveform generator offset, 1121
waveform generator output control, 1109
waveform generator output load impedance, 1110
waveform generator output polarity, 1111
waveform generator period, 1112

waveform generator pulse width, 1094
 waveform generator ramp symmetry, 1095
 waveform generator reset defaults, 1113
 waveform generator square wave duty cycle, 1096
 waveform introduction, 1045
 waveform maximum vertical value measured, 484
 waveform minimum vertical value measured, 485
 waveform must cross voltage level to be an occurrence, 1168
 WAVEform parameters, 62
 waveform peak-to-peak vertical value measured, 445, 486
 waveform period, 457
 waveform persistence, 316
 waveform RMS value measured, 446, 488
 waveform save option for segments, 654
 waveform source, 1065
 waveform source subsource, 1069
 waveform standard deviation value measured, 466
 waveform vertical amplitude, 442, 481
 waveform voltage measured at marker, 1172, 1173
 waveform, byte order, 1051
 waveform, count, 1052
 waveform, data, 1053
 waveform, format, 1055
 waveform, points, 1056, 1058
 waveform, preamble, 1060
 waveform, type, 1070
 waveform, unsigned, 1071
 waveform, view, 1072
 waveform, X increment, 1073
 waveform, X origin, 1074
 waveform, X reference, 1075
 waveform, Y increment, 1076
 waveform, Y origin, 1077
 waveform, Y reference, 1078
 WAVEform:FORMat, 62
 waveforms, mask test run, 543
 Web control, 67
 WGEN commands, 1079
 WGEN trigger source, 989
 what's new, 33
 width, 808, 997
 window, 956, 957, 958
 window time, 951
 window time base mode, 949
 window, measurement, 491
 windows, 357
 windows as filters to Fast Fourier Transforms, 357
 windows for Fast Fourier Transform functions, 357
 WMEMory commands, 1123
 word counter (ARINC 429), 668
 word counter (ARINC 429), reset, 667
 word format, 1055
 word format for data transfer, 1049

word format, ARINC 429, 669
 word select (WS) low, I2S trigger, 739
 word select (WS) source, I2S, 729
 word width, SPI decode, 786
 write text to display, 936
 WriteEEBlock method, 59, 65
 WriteList method, 59
 WriteNumber method, 59
 WriteString method, 59

X

X axis markers, 398
 X cursor units, 407, 408
 X delta, 400, 406
 X delta, mask scaling, 546
 X1 and X2 cursor value difference, 400, 406
 X1 cursor, 398, 402, 403
 X1, mask scaling, 545
 X2 cursor, 398, 404, 405
 X-axis functions, 948
 X-increment, 1073
 X-of-max measurement, 492
 X-of-min measurement, 493
 X-origin, 1074
 X-reference, 1075
 X-Y mode, 948, 949

Y

Y axis markers, 398
 Y cursor units, 412, 413
 Y offset, reference waveform, 1130
 Y range, reference waveform, 1131
 Y scale, reference waveform, 1132
 Y1 and Y2 cursor value difference, 411
 Y1 cursor, 398, 403, 409, 411
 Y1, mask scaling, 547
 Y2 cursor, 398, 405, 410, 411
 Y2, mask scaling, 548
 Y-axis value, 1077
 Y-increment, 1076
 Y-origin, 1077, 1078
 Y-reference, 1078

Z

zero values in waveform data, 1053
 zone 1 or zone 2 mode, 1039
 zone 1 or zone 2 placement, 1040
 zone 1 or zone 2 state, 1042
 zone 1 or zone 2 validity, 1041
 zone qualified trigger source, 1037
 zone qualified trigger state, 1038
 ZONE trigger commands, 1036
 zoomed time base, 949
 zoomed time base measurement window, 491
 zoomed time base mode, how autoscale affects, 197
 zoomed window horizontal scale, 958

