

# **Agilent InfiniiVision 5000 Series Oscilloscopes**

**Programmer's Guide**



**Agilent Technologies**

# Notices

© Agilent Technologies, Inc. 2007-2008

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

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

## Manual Part Number

Version 05.20.0000

## Edition

December 5, 2008

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 5000 Series oscilloscopes:

**Table 1** InfiniiVision 5000 Series Oscilloscope Models

Channels	Input Bandwidth (Maximum Sample Rate)		
	500 MHz (4 GSa/s)	300 MHz (2 GSa/s)	100 MHz (2 GSa/s)
4 analog	DSO5054A	DSO5034A	DSO5014A
2 analog	DSO5052A	DSO5032A	DSO5012A

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

- Chapter 1, "[What's New](#)" on page 19, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, "[Setting Up](#)" on page 31, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, "[Getting Started](#)" on page 41, 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](#)" on page 55, is a brief listing of the 5000 Series oscilloscope commands and syntax.

The next chapters provide reference information:

- Chapter 5, "[Commands by Subsystem](#)" on page 95, describes the set of commands that belong to an individual subsystem and explains the function of each command. Command arguments and syntax are described. Some command descriptions have example code.
- Chapter 6, "[Commands A-Z](#)" on page 535, contains an alphabetical listing of all command elements.
- Chapter 7, "[Obsolete and Discontinued Commands](#)" on page 563, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 8, "[Error Messages](#)" on page 615, 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 9, "[Status Reporting](#)" on page 623, describes the oscilloscope's status registers and how to check the status of the instrument.

- Chapter 10, "Synchronizing Acquisitions" on page 647, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- Chapter 11, "More About Oscilloscope Commands" on page 657, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 12, "Programming Examples" on page 679.

**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 82350A 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](http://www.agilent.com)" and search for "Connectivity Guide".
- For the latest versions of this and other manuals, see:  
["http://www.agilent.com/find/5000manual"](http://www.agilent.com/find/5000manual)

# Contents

In This Book 3

## 1 What's New

What's New in Version 5.20	<span style="color: blue;">20</span>
What's New in Version 5.15	<span style="color: blue;">23</span>
What's New in Version 5.10	<span style="color: blue;">25</span>
What's New in Version 5.00	<span style="color: blue;">26</span>
What's New in Version 4.10	<span style="color: blue;">28</span>
Version 4.00 at Introduction	<span style="color: blue;">29</span>

## 2 Setting Up

Step 1. Install Agilent IO Libraries Suite software	<span style="color: blue;">32</span>
Step 2. Connect and set up the oscilloscope	<span style="color: blue;">33</span>
Using the USB (Device) Interface	<span style="color: blue;">33</span>
Using the LAN Interface	<span style="color: blue;">33</span>
Using the GPIB Interface	<span style="color: blue;">34</span>
Step 3. Verify the oscilloscope connection	<span style="color: blue;">35</span>

## 3 Getting Started

Basic Oscilloscope Program Structure	<span style="color: blue;">42</span>
Initializing	<span style="color: blue;">42</span>
Capturing Data	<span style="color: blue;">42</span>
Analyzing Captured Data	<span style="color: blue;">43</span>

Programming the Oscilloscope	44
Referencing the IO Library	44
Opening the Oscilloscope Connection via the IO Library	45
Initializing the Interface and the Oscilloscope	45
Using :AUToscale to Automate Oscilloscope Setup	46
Using Other Oscilloscope Setup Commands	46
Capturing Data with the :DIGITIZE Command	47
Reading Query Responses from the Oscilloscope	49
Reading Query Results into String Variables	50
Reading Query Results into Numeric Variables	50
Reading Definite-Length Block Query Response Data	50
Sending Multiple Queries and Reading Results	51
Checking Instrument Status	52
Other Ways of Sending Commands	53
Telnet Sockets	53
Sending SCPI Commands Using Browser Web Control	53

## 4 Commands Quick Reference

Command Summary	56
Syntax Elements	92
Number Format	92
<NL> (Line Terminator)	92
[ ] (Optional Syntax Terms)	92
{ } (Braces)	92
::= (Defined As)	92
< > (Angle Brackets)	93
... (Ellipsis)	93
n,...p (Value Ranges)	93
d (Digits)	93
Quoted ASCII String	93
Definite-Length Block Response Data	93

## 5 Commands by Subsystem

Common (*) Commands	97
*CLS (Clear Status)	101
*ESE (Standard Event Status Enable)	102
*ESR (Standard Event Status Register)	104
*IDN (Identification Number)	106
*LRN (Learn Device Setup)	107
*OPC (Operation Complete)	108

*OPT (Option Identification)	109
*RCL (Recall)	110
*RST (Reset)	111
*SAV (Save)	114
*SRE (Service Request Enable)	115
*STB (Read Status Byte)	117
*TRG (Trigger)	119
*TST (Self Test)	120
*WAI (Wait To Continue)	121
Root (:) Commands	122
:AER (Arm Event Register)	125
:AUToscale	126
:AUToscale:AMODE	128
:AUToscale:CHANnels	129
:BLANK	130
:CDISplay	131
:DIGItize	132
:HWEnable (Hardware Event Enable Register)	134
:HWERegister:CONDition (Hardware Event Condition Register)	136
:HWERegister[:EVENT] (Hardware Event Event Register)	138
:MERGe	140
:MTEnable (Mask Test Event Enable Register)	141
:MTERegister[:EVENT] (Mask Test Event Event Register)	143
:OPEE (Operation Status Enable Register)	145
:OPERegister:CONDition (Operation Status Condition Register)	147
:OPERegister[:EVENT] (Operation Status Event Register)	149
:OVLenable (Overload Event Enable Register)	151
:OVLRegister (Overload Event Register)	153
:PRINt	155
:RUN	156
:SERial	157
:SINGLe	158
:STATus	159
:STOP	160
:TER (Trigger Event Register)	161
:VIEW	162
:ACQuire Commands	163
:ACQuire:AALias	165
:ACQuire:COMplete	166
:ACQuire:COUNt	167
:ACQuire:DAALias	168

:ACQuire:MODE	169
:ACQuire:POINts	170
:ACQuire:SEGMed:ANALyze	171
:ACQuire:SEGMed:COUNt	172
:ACQuire:SEGMed:INDex	173
:ACQuire:SRATe	176
:ACQuire:TYPE	177
:CALibrate Commands	179
:CALibrate:DATE	181
:CALibrate:LABEL	182
:CALibrate:OUTPut	183
:CALibrate:STARt	184
:CALibrate:STATus	185
:CALibrate:SWITch	186
:CALibrate:TEMPerature	187
:CALibrate:TIME	188
:CHANnel<n> Commands	189
:CHANnel<n>:BWLimit	192
:CHANnel<n>:COUpling	193
:CHANnel<n>:DISPlay	194
:CHANnel<n>:IMPedance	195
:CHANnel<n>:INVert	196
:CHANnel<n>:LABEL	197
:CHANnel<n>:OFFSet	198
:CHANnel<n>:PROBe	199
:CHANnel<n>:PROBe:ID	200
:CHANnel<n>:PROBe:SKEW	201
:CHANnel<n>:PROBe:STYPe	202
:CHANnel<n>:PROTection	203
:CHANnel<n>:RANGE	204
:CHANnel<n>:SCALe	205
:CHANnel<n>:UNITs	206
:CHANnel<n>:VERNier	207
:DISPlay Commands	208
:DISPlay:CLEar	210
:DISPlay:DATA	211
:DISPlay:LABEL	213
:DISPlay:LABList	214
:DISPlay:PERSistence	215
:DISPlay:SOURce	216

:DISPlay:VECTors	217
:EXternal Trigger Commands	218
:EXternal:BWLimit	220
:EXternal:IMPedance	221
:EXternal:PROBe	222
:EXternal:PROBe:ID	223
:EXternal:PROBe:STYPe	224
:EXternal:PROTection	225
:EXternal:RANGE	226
:EXternal:UNITS	227
:FUNCtion Commands	228
:FUNCtion:CENTER	231
:FUNCtion:DISPlay	232
:FUNCtion:GOFT:OPERation	233
:FUNCtion:GOFT:SOURce1	234
:FUNCtion:GOFT:SOURce2	235
:FUNCtion:OFFSet	236
:FUNCtion:OPERation	237
:FUNCtion:RANGE	238
:FUNCtion:REFerence	239
:FUNCtion:SCALe	240
:FUNCtion:SOURce1	241
:FUNCtion:SOURce2	242
:FUNCtion:SPAN	243
:FUNCtion:WINDOW	244
:HARDcopy Commands	245
:HARDcopy:AREA	247
:HARDcopy:APRinter	248
:HARDcopy:FACTors	249
:HARDcopy:FFEed	250
:HARDcopy:INKSaver	251
:HARDcopy:LAYout	252
:HARDcopy:PAlette	253
:HARDcopy:PRINTER:LIST	254
:HARDcopy:STARt	255
:MARKer Commands	256
:MARKer:MODE	258
:MARKer:X1Position	259
:MARKer:X1Y1source	260
:MARKer:X2Position	261

:MARKer:X2Y2source 262  
:MARKer:XDELta 263  
:MARKer:Y1Position 264  
:MARKer:Y2Position 265  
:MARKer:YDELta 266  
  
:MEASure Commands 267  
  :MEASure:CLEAR 274  
  :MEASure:COUNter 275  
  :MEASure:DEFine 276  
  :MEASure:DELay 279  
  :MEASure:DUTYcycle 281  
  :MEASure:FALLtime 282  
  :MEASure:FREQuency 283  
  :MEASure:NWIDth 284  
  :MEASure:OVERshoot 285  
  :MEASure:PERiod 287  
  :MEASure:PHASE 288  
  :MEASure:PREShoot 289  
  :MEASure:PVIDth 290  
  :MEASure:RESults 291  
  :MEASure:RISetime 294  
  :MEASure:SDEViation 295  
  :MEASure:SHOW 296  
  :MEASure:SOURce 297  
  :MEASure:STATistics 299  
  :MEASure:STATistics:INCRement 300  
  :MEASure:STATistics:RESET 301  
  :MEASure:TEDGE 302  
  :MEASure:TVALue 304  
  :MEASure:VAMPLitude 306  
  :MEASure:VAVerage 307  
  :MEASure:VBASe 308  
  :MEASure:VMAX 309  
  :MEASure:VMIN 310  
  :MEASure:VPP 311  
  :MEASure:VRATio 312  
  :MEASure:VRMS 313  
  :MEASure:VTIMe 314  
  :MEASure:VTOP 315  
  :MEASure:XMAX 316  
  :MEASure:XMIN 317

:MTESt Commands 318  
  :MTESt:AMASK:CREate 323  
  :MTESt:AMASK:SOURce 324  
  :MTESt:AMASK:UNITs 325  
  :MTESt:AMASK:XDELta 326  
  :MTESt:AMASK:YDELta 327  
  :MTESt:COUNT:FWAVeforms 328  
  :MTESt:COUNT:RESet 329  
  :MTESt:COUNT:TIME 330  
  :MTESt:COUNT:WAveforms 331  
  :MTESt:DATA 332  
  :MTESt:DELete 333  
  :MTESt:ENABLE 334  
  :MTESt:LOCK 335  
  :MTESt:OUTPut 336  
  :MTESt:RMODE 337  
  :MTESt:RMODE:FACTion:PRINT 338  
  :MTESt:RMODE:FACTion:SAVE 339  
  :MTESt:RMODE:FACTion:STOP 340  
  :MTESt:RMODE:SIGMa 341  
  :MTESt:RMODE:TIME 342  
  :MTESt:RMODE:WAveforms 343  
  :MTESt:SCALe:BIND 344  
  :MTESt:SCALe:X1 345  
  :MTESt:SCALe:XDELta 346  
  :MTESt:SCALe:Y1 347  
  :MTESt:SCALe:Y2 348  
  :MTESt:SOURce 349  
  :MTESt:TITLe 350

:RECall Commands 351  
  :RECall:FILEname 352  
  :RECall:IMAGe[:STARt] 353  
  :RECall:MASK[:STARt] 354  
  :RECall:PWD 355  
  :RECall:SETup[:STARt] 356

:SAVE Commands 357  
  :SAVE:FILEname 359  
  :SAVE:IMAGe[:STARt] 360  
  :SAVE:IMAGe:AREA 361  
  :SAVE:IMAGe:FACTors 362  
  :SAVE:IMAGe:FORMAT 363

:SAVE:IMAGe:INKSaver	364
:SAVE:IMAGe:PAlette	365
:SAVE:MASK[:STARt]	366
:SAVE:PWD	367
:SAVE:SETup[:STARt]	368
:SAVE:WAveform[:STARt]	369
:SAVE:WAveform:FORMAT	370
:SAVE:WAveform:LENGTH	371
:SAVE:WAveform:SEGmented	372
 :SBUS Commands	373
:SBUS:CAN:COUNt:ERRor	375
:SBUS:CAN:COUNt:OVERload	376
:SBUS:CAN:COUNt:RESet	377
:SBUS:CAN:COUNt:TOTal	378
:SBUS:CAN:COUNt:UTILization	379
:SBUS:DISPlay	380
:SBUS:IIC:ASIZe	381
:SBUS:LIN:PARity	382
:SBUS:MODE	383
:SBUS:SPI:WIDTh	384
:SBUS:UART:BASE	385
:SBUS:UART:COUNt:ERRor	386
:SBUS:UART:COUNt:RESet	387
:SBUS:UART:COUNt:RXFRAMES	388
:SBUS:UART:COUNt:TXFRAMES	389
:SBUS:UART:FRAMing	390
 :SYSTem Commands	391
:SYSTem:DATE	392
:SYSTem:DSP	393
:SYSTem:ERRor	394
:SYSTem:LOCK	395
:SYSTem:PROTection:LOCK	396
:SYSTem:SETup	397
:SYSTem:TIME	399
 :TIMEbase Commands	400
:TIMEbase:MODE	402
:TIMEbase:POSition	403
:TIMEbase:RANGE	404
:TIMEbase:REFerence	405
:TIMEbase:SCALe	406

:TIMEbase:VERNier	407
:TIMEbase:WINDOW:POSITION	408
:TIMEbase:WINDOW:RANGE	409
:TIMEbase:WINDOW:SCALE	410
:TRIGger Commands	411
General :TRIGger Commands	414
:TRIGger:HReject	415
:TRIGger:HOLDoff	416
:TRIGger:MODE	417
:TRIGger:NREject	418
:TRIGger:PATTERn	419
:TRIGger:SWEep	421
:TRIGger:CAN Commands	422
:TRIGger:CAN:PATTERn:DATA	424
:TRIGger:CAN:PATTERn:DATA:LENGTH	425
:TRIGger:CAN:PATTERn:ID	426
:TRIGger:CAN:PATTERn:ID:MODE	427
:TRIGger:CAN:SAMPLEpoint	428
:TRIGger:CAN:SIGNAl:BAUDrate	429
:TRIGger:CAN:SOURce	430
:TRIGger:CAN:TRIGger	431
:TRIGger:DURation Commands	433
:TRIGger:DURation:GREaterthan	434
:TRIGger:DURation:LESSthan	435
:TRIGger:DURation:PATTERn	436
:TRIGger:DURation:QUALifier	437
:TRIGger:DURation:RANGE	438
:TRIGger[:EDGE] Commands	439
:TRIGger[:EDGE]:COUpling	440
:TRIGger[:EDGE]:LEVel	441
:TRIGger[:EDGE]:REject	442
:TRIGger[:EDGE]:SLOPe	443
:TRIGger[:EDGE]:SOURce	444
:TRIGger:GLITch Commands	445
:TRIGger:GLITch:GREaterthan	446
:TRIGger:GLITch:LESSthan	447
:TRIGger:GLITch:LEVel	448
:TRIGger:GLITch:POLarity	449
:TRIGger:GLITch:QUALifier	450
:TRIGger:GLITch:RANGE	451
:TRIGger:GLITch:SOURce	452

:TRIGger:IIC Commands 453  
:TRIGger:IIC:PATTern:ADDReSS 454  
:TRIGger:IIC:PATTern:DATA 455  
:TRIGger:IIC:PATTern:DATa2 456  
:TRIGger:IIC[:SOURce]:CLOCK 457  
:TRIGger:IIC[:SOURce]:DATA 458  
:TRIGger:IIC:TRIGger:QUALifier 459  
:TRIGger:IIC:TRIGger[:TYPE] 460  
:TRIGger:LIN Commands 462  
:TRIGger:LIN:ID 463  
:TRIGger:LIN:SAMPLEpoint 464  
:TRIGger:LIN:SIGNal:BAUDrate 465  
:TRIGger:LIN:SOURce 466  
:TRIGger:LIN:STANDARD 467  
:TRIGger:LIN:SYNCbreak 468  
:TRIGger:LIN:TRIGger 469  
:TRIGger:SPI Commands 470  
:TRIGger:SPI:CLOCK:SLOPe 471  
:TRIGger:SPI:CLOCK:TIMEout 472  
:TRIGger:SPI:FRAMing 473  
:TRIGger:SPI:PATTern:DATA 474  
:TRIGger:SPI:PATTern:WIDTh 475  
:TRIGger:SPI:SOURce:CLOCK 476  
:TRIGger:SPI:SOURce:DATA 477  
:TRIGger:SPI:SOURce:FRAMe 478  
:TRIGger:TV Commands 479  
:TRIGger:TV:LINE 480  
:TRIGger:TV:MODE 481  
:TRIGger:TV:POLarity 482  
:TRIGger:TV:SOURce 483  
:TRIGger:TV:STANDARD 484  
:TRIGger:UART Commands 485  
:TRIGger:UART:BASE 487  
:TRIGger:UART:BAUDrate 488  
:TRIGger:UART:BITorder 489  
:TRIGger:UART:BURSt 490  
:TRIGger:UART:DATA 491  
:TRIGger:UART:IDLE 492  
:TRIGger:UART:PARity 493  
:TRIGger:UART:POLarity 494  
:TRIGger:UART:QUALifier 495  
:TRIGger:UART:SOURce:RX 496

:TRIGger:UART:SOURce:TX 497  
:TRIGger:UART:TYPE 498  
:TRIGger:UART:WIDTh 499

:WAveform Commands 500  
:WAveform:BYTeorder 507  
:WAveform:COUNt 508  
:WAveform:DATA 509  
:WAveform:FORMat 511  
:WAveform:POINts 512  
:WAveform:POINts:MODE 514  
:WAveform:PREamble 516  
:WAveform:SEGmented:COUNt 519  
:WAveform:SEGmented:TTAG 520  
:WAveform:SOURce 521  
:WAveform:SOURce:SUBSource 525  
:WAveform:TYPE 526  
:WAveform:UNSIGNED 527  
:WAveform:VIEW 528  
:WAveform:XINCrement 529  
:WAveform:XORigin 530  
:WAveform:XREFerence 531  
:WAveform:YINCrement 532  
:WAveform:YORigin 533  
:WAveform:YREFerence 534

## 6 Commands A-Z

## 7 Obsolete and Discontinued Commands

:CHANnel:LABEL 568  
:CHANnel2:SKEW 569  
:CHANnel<n>:INPut 570  
:CHANnel<n>:PMODe 571  
:DISPlay:CONNect 572  
:ERASe 573  
:EXTernal:INPut 574  
:EXTernal:PMODE 575  
:FUNCTION:SOURce 576  
:FUNCTION:VIEW 577  
:HARDcopy:DESTination 578  
:HARDcopy:DEvice 579  
:HARDcopy:FILEname 580

:HARDcopy:FORMAT 581  
:HARDcopy:GRAYscale 582  
:HARDcopy:IGColors 583  
:HARDcopy:PDRiver 584  
:MEASure:LOWER 585  
:MEASure:SCRatch 586  
:MEASure:TDELta 587  
:MEASure:THResholds 588  
:MEASure:TMAX 589  
:MEASure:TMIN 590  
:MEASure:TSTArt 591  
:MEASure:TSTOP 592  
:MEASure:TVOLt 593  
:MEASure:UPPer 595  
:MEASure:VDELta 596  
:MEASure:VSTArt 597  
:MEASure:VSTOP 598  
:MTEST:AMASK:{SAVE | STORe} 599  
:MTEST:AVERage 600  
:MTEST:AVERage:COUNT 601  
:MTEST:LOAD 602  
:MTEST:RUMode 603  
:MTEST:RUMode:SOFailure 604  
:MTEST:{STARt | STOP} 605  
:MTEST:TRIGger:SOURce 606  
:PRINt? 607  
:TIMEbase:DELay 609  
:TRIGger:CAN:ACKNowledge 610  
:TRIGger:CAN:SIGNal:DEFinition 611  
:TRIGger:LIN:SIGNal:DEFinition 612  
:TRIGger:TV:TVMode 613

## 8 Error Messages

## 9 Status Reporting

Status Reporting Data Structures 626  
Status Byte Register (STB) 629  
Service Request Enable Register (SRE) 631  
Trigger Event Register (TER) 632  
Output Queue 633

Message Queue	634
(Standard) Event Status Register (ESR)	635
(Standard) Event Status Enable Register (ESE)	636
Error Queue	637
Operation Status Event Register (:OPERegister[:EVENT])	638
Operation Status Condition Register (:OPERegister:CONDITION)	639
Arm Event Register (AER)	640
Overload Event Register (:OVLRegister)	641
Hardware Event Event Register (:HWERegister[:EVENT])	642
Hardware Event Condition Register (:HWERegister:CONDITION)	643
Mask Test Event Event Register (:MTERegister[:EVENT])	644
Clearing Registers and Queues	645
Status Reporting Decision Chart	646

## **10 Synchronizing Acquisitions**

Synchronization in the Programming Flow	648
Set Up the Oscilloscope	648
Acquire a Waveform	648
Retrieve Results	648
Blocking Synchronization	649
Polling Synchronization With Timeout	650
Synchronizing with a Single-Shot Device Under Test (DUT)	652
Synchronization with an Averaging Acquisition	654

## **11 More About Oscilloscope Commands**

Command Classifications	658
Core Commands	658
Non-Core Commands	658
Obsolete Commands	658
Valid Command/Query Strings	659
Program Message Syntax	659
Command Tree	663
Duplicate Mnemonics	674
Tree Traversal Rules and Multiple Commands	675
Query Return Values	677

All Oscilloscope Commands Are Sequential 678

## 12 Programming Examples

SICL Examples 680

    SICL Example in C 680

    SICL Example in Visual Basic 689

VISA Examples 698

    VISA Example in C 698

    VISA Example in Visual Basic 707

    VISA Example in C# 717

    VISA Example in Visual Basic .NET 731

VISA COM Examples 744

    VISA COM Example in Visual Basic 744

    VISA COM Example in C# 754

    VISA COM Example in Visual Basic .NET 765

## Index

## 1 **What's New**

What's New in Version 5.20	20
What's New in Version 5.15	23
What's New in Version 5.10	25
What's New in Version 5.00	26
What's New in Version 4.10	28
Version 4.00 at Introduction	29



## What's New in Version 5.20

New features in version 5.20 of the InfiniiVision 5000 Series oscilloscope software are:

- Mask testing, enabled with Option LMT.
- Tracking cursors (markers) have been added.
- Measurement statistics have been added.
- Labels can now be up to 10 characters.

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

### New Commands

Command	Description
:ACQuire:SEGMENTed:ANALyze (see <a href="#">page 171</a> )	Calculates measurement statistics and/or infinite persistence over all segments that have been acquired.
:CALibrate:OUTPut (see <a href="#">page 183</a> )	Selects the signal output on the rear panel TRIG OUT BNC.
:HARDcopy:LAYOUT (see <a href="#">page 252</a> )	Sets the hardcopy layout mode.
:MEASure:RESults (see <a href="#">page 291</a> )	Returns measurement statistics values.
:MEASure:STATistics (see <a href="#">page 299</a> )	Sets the type of measurement statistics to return.
:MEASure:STATistics:INCRement (see <a href="#">page 300</a> )	Updates the statistics once (incrementing the count by one) using the current measurement values.
:MEASure:STATistics:RESET (see <a href="#">page 301</a> )	Resets the measurement statistics values.
:MTEnable (Mask Test Event Enable Register) (see <a href="#">page 141</a> )	Sets a mask in the Mask Test Event Enable register.
:MTERegister[:EVENT] (Mask Test Event Event Register) (see <a href="#">page 143</a> )	Returns the integer value contained in the Mask Test Event Event Register and clears the register.
:MTEST Commands (see <a href="#">page 318</a> )	Commands and queries to control the mask test (Option LMT) features.
:RECall:MASK[:STARt] (see <a href="#">page 366</a> )	Recalls a mask.
:SAVE:MASK[:STARt] (see <a href="#">page 366</a> )	Saves the current mask.
:SAVE:WAVEform:SEGMENTed (see <a href="#">page 372</a> )	Specifies which segments are included when the waveform is saved.
:TRIGger:UART:BASE (see <a href="#">page 487</a> )	Selects the front panel UART/RS232 trigger setup data selection option from HEX or BINary.

**Changed Commands**

Command	Differences
:CHANnel<n>:LABEL (see <a href="#">page 197</a> )	Labels can now be up to 10 characters.
:DISPlay:LABList (see <a href="#">page 214</a> )	Labels can now be up to 10 characters.
:MARKer:MODE (see <a href="#">page 258</a> )	You can now select the WAVEform tracking cursors mode.
:RECall:PWD (see <a href="#">page 355</a> )	You can set the present working directory in addition to querying for this information.
:SAVE:IMAGe[:STARt] (see <a href="#">page 360</a> )	The file extension specified will change the :SAVE:IMAGe:FORMAT setting if it is a valid image file extension.
:SAVE:PWD (see <a href="#">page 367</a> )	You can set the present working directory in addition to querying for this information.
:SAVE:WAVEform[:STARt] (see <a href="#">page 360</a> )	The file extension specified will change the :SAVE:WAVEform:FORMAT setting if it is a valid waveform file extension.
:TRIGger:CAN:SIGNal:BAUDrate (see <a href="#">page 429</a> )	The baud rate value can now be set in 100 b/s increments.
:TRIGger:LIN:SIGNal:BAUDrate (see <a href="#">page 465</a> )	The baud rate value can now be set in 100 b/s increments.
:TRIGger:UART:BAUDrate (see <a href="#">page 488</a> )	The baud rate value can now be set in 100 b/s increments and the maximum baud rate is now 3 Mb/s.
:TRIGger:UART:DATA (see <a href="#">page 491</a> )	You can now specify the data value using a quoted ASCII character.

**Obsolete Commands**

Obsolete Command	Current Command Equivalent	Behavior Differences
:MTEST:AMASK:{SAVE   STORe} (see <a href="#">page 599</a> )	:SAVE:MASK[:STARt] (see <a href="#">page 366</a> )	
:MTEST:AVERage (see <a href="#">page 600</a> )	:ACQuire:TYPE AVERage (see <a href="#">page 177</a> )	
:MTEST:AVERage:COUNT (see <a href="#">page 601</a> )	:ACQuire:COUNT (see <a href="#">page 167</a> )	
:MTEST:LOAD (see <a href="#">page 602</a> )	:RECall:MASK[:STARt] (see <a href="#">page 354</a> )	
:MTEST:RUMode (see <a href="#">page 603</a> )	:MTEST:RMODE (see <a href="#">page 337</a> )	
:MTEST:RUMode:SOFailure (see <a href="#">page 604</a> )	:MTEST:RMODE:FACTion:STOP (see <a href="#">page 340</a> )	

## 1 What's New

Obsolete Command	Current Command Equivalent	Behavior Differences
:MTEST:{STARt   STOP} (see page 605)	:RUN (see <a href="#">page 156</a> ) or :STOP (see <a href="#">page 160</a> )	
:MTEST:TRIGger:SOURce (see page 606)	:TRIGger Commands (see <a href="#">page 411</a> )	There are various commands for setting the source with different types of triggers.

## What's New in Version 5.15

New features in version 5.15 of the InfiniiVision 5000 Series oscilloscope software are:

- Waveform math can be performed using channels 3 and 4, and there is a new ADD operator.
- Ratio of AC RMS values measurement.
- Analog channel impedance protection lock.

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

### New Commands

Command	Description
:FUNCtion:GOFT:OPERation (see <a href="#">page 233</a> )	Selects the math operation for the internal g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNCtion:GOFT:SOURce1 (see <a href="#">page 234</a> )	Selects the first input channel for the g(t) source.
:FUNCtion:GOFT:SOURce2 (see <a href="#">page 235</a> )	Selects the second input channel for the g(t) source.
:FUNCtion:SOURce1 (see <a href="#">page 241</a> )	Selects the first source for the ADD, SUBTract, and MULTiply arithmetic operations or the single source for the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNCtion:SOURce2 (see <a href="#">page 242</a> )	Selects the second input channel for the ADD, SUBTract, and MULTiply arithmetic operations.
:MEASure:VRATio (see <a href="#">page 312</a> )	Measures and returns the ratio of AC RMS values of the specified sources expressed in dB.
:SYSTem:PROTection:LOCK (see <a href="#">page 396</a> )	Disables/enables the fifty ohm input impedance setting.

**Changed Commands**

<b>Command</b>	<b>Differences</b>
:ACQuire:COUNt (see <a href="#">page 167</a> )	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.
:FUNCtion:OPERation (see <a href="#">page 237</a> )	The ADD parameter is new, and now that waveform math can be performed using channels 3 and 4, this command selects the operation only.
:FUNCtion:WINDOW (see <a href="#">page 244</a> )	You can now select the Blackman-Harris FFT window.

**Obsolete Commands**

<b>Obsolete Command</b>	<b>Current Command Equivalent</b>	<b>Behavior Differences</b>
:FUNCtion:SOURce (see <a href="#">page 576</a> )	:FUNCtion:SOURce1 (see <a href="#">page 241</a> )	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.

## What's New in Version 5.10

New features in version 5.10 of the InfiniiVision 5000 Series oscilloscope software are:

- Segmented memory acquisition mode, enabled with Option SGM.

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

### New Commands

Command	Description
:ACQuire:SEGMENTed:COUNt (see page 172)	Sets the number of memory segments.
:ACQuire:SEGMENTed:INDex (see page 173)	Selects the segmented memory index.
:WAVeform:SEGMENTed:COUNt (see page 519)	Returns the number of segments in the currently acquired waveform data.
:WAVeform:SEGMENTed:TTAG (see page 520)	Returns the time tag for the selected segmented memory index.

### Changed Commands

Command	Differences
:ACQuire:MODE (see page 169)	You can now select the SEGMENTed memory mode.

## What's New in Version 5.00

New features in version 5.00 of the InfiniiVision 5000 Series oscilloscope software are:

- Serial triggering and decode options are now available.
- The :SAVE and :RECall command subsystems.
- Changes to the :HARDcopy command subsystem to make a clearer distinction between printing and save/recall functionality.

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

### New Commands

Command	Description
:HARDcopy:STARt (see <a href="#">page 255</a> )	Starts a print job.
:HARDcopy:APRinter (see <a href="#">page 248</a> )	Sets the active printer.
:HARDcopy:AREA (see <a href="#">page 247</a> )	Specifies the area of the display to print (currently SCReen only).
:HARDcopy:INKSaver (see <a href="#">page 251</a> )	Inverts screen colors to save ink when printing.
:HARDcopy:PRINTER:LIST (see <a href="#">page 254</a> )	Returns a list of the available printers.
:RECall Commands (see <a href="#">page 351</a> )	Commands for recalling previously saved oscilloscope setups and traces.
:SAVE Commands (see <a href="#">page 357</a> )	Commands for saving oscilloscope setups and traces, screen images, and data.
:SBUS Commands (see <a href="#">page 373</a> )	Commands for controlling oscilloscope functions associated with the serial decode bus.
:TRIGger:CAN Commands (see <a href="#">page 422</a> )	Commands for triggering on Controller Area Network (CAN) version 2.0A and 2.0B signals.
:TRIGger:IIC Commands (see <a href="#">page 453</a> )	Commands for triggering on Inter-IC (IIC) signals.
:TRIGger:LIN Commands (see <a href="#">page 462</a> )	Commands for triggering on Local Interconnect Network (LIN) signals.
:TRIGger:SPI Commands (see <a href="#">page 470</a> )	Commands for triggering on Serial Peripheral Interface (SPI) signals.
:TRIGger:UART Commands (see <a href="#">page 485</a> )	Commands for triggering on UART/RS-232 signals.
:WAVEform:SOURce:SUBSource (see <a href="#">page 525</a> )	Selects subsource when :WAVEform:SOURce is SBUS (serial decode).

**Changed Commands**

Command	Differences
:BLANK (see page 130)	Now, you can also use this command with the serial decode bus.
:DIGItize (see page 132)	Now, you can also use this command with the serial decode bus.
:STATUs (see page 159)	Now, you can also use this command with the serial decode bus.
:TRIGger:MODE (see page 417)	You can now select the serial triggering modes.
:VIEW (see page 162)	Now, you can now use this command with the serial decode bus.
:WAVeform:SOURce (see page 521)	Now, you can also use this command with the serial decode bus.

**Obsolete Commands**

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:FILEname (see page 580)	:RECall:FILEname (see page 352) :SAVE:FILEname (see page 352)	
:HARDcopy:FORMAT (see page 581)	:HARDcopy:APRinter (see page 248) :SAVE:IMAGe:FORMAT (see page 363) :SAVE:WAVeform:FORMAT (see page 370)	
:HARDcopy:IGColors (see page 583)	:HARDcopy:INKSaver (see page 251)	
:HARDcopy:PDRiver (see page 584)	:HARDcopy:APRinter (see page 248)	

## What's New in Version 4.10

New features in version 4.10 of the InfiniiVision 5000 Series oscilloscope software are:

- The square root waveform math function.
- Several new hardcopy printer drivers.

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

Changed Commands	Command	Differences
	:FUNCTION:OPERation (see <a href="#">page 237</a> )	You can now select the SQRT (square root) waveform math function.
	:HARDcopy:PDRiver (see <a href="#">page 584</a> )	You can now select the new DJPR0kx50, DJ55xx, PS470, and LJFastraster printer drivers.

## Version 4.00 at Introduction

The Agilent InfiniiVision 5000 Series oscilloscopes were introduced with version 4.00 of oscilloscope operating software. The command set is similar to the 6000 Series oscilloscopes (and the 54620/54640 Series oscilloscopes before them) except that digital channels, rear-panel 10 MHz reference BNC input/output, and serial bus triggering/decode features are not present.

## **1** What's New

## 2 **Setting Up**

Step 1. Install Agilent IO Libraries Suite software 32

Step 2. Connect and set up the oscilloscope 33

Step 3. Verify the oscilloscope connection 35

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**

Insert the Automation-Ready CD that was shipped with your oscilloscope into the controller PC's CD-ROM drive, and follow its installation instructions.

You can also download the Agilent IO Libraries Suite software from the web at:

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

## Step 2. Connect and set up the oscilloscope

The 5000 Series oscilloscope has three different interfaces you can use for programming: USB (device), LAN, or GPIB.

All three interfaces are "live" by default, but you can turn them off if desired. To access these settings press the **Utility** key on the front panel, then press the **I/O** softkey, then press the **Control** softkey.



**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.
- 2 On the oscilloscope, verify that the controller interface is enabled:
  - a Press the **Utility** button.
  - b Using the softkeys, press **I/O** and **Control**.
  - c Ensure the box next to **USB** is selected (█). If not (□), use the Entry knob to select **USB**; then, press the **Control** softkey again.

### Using the LAN Interface

- 1 If the controller PC isn't already connected to the local area network (LAN), do that first.
- 2 Get the oscilloscope's network parameters (hostname, domain, IP address, subnet mask, gateway IP, DNS IP, etc.) from your network administrator.
- 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** On the oscilloscope, verify that the controller interface is enabled:
  - a** Press the **Utility** button.
  - b** Using the softkeys, press **I/O** and **Control**.
  - c** Ensure the box next to **LAN** is selected (). If not (, use the Entry knob to select **LAN**; then, press the **Control** softkey again.
- 5** Configure the oscilloscope's LAN interface:
  - a** Press the **Configure** softkey until "LAN" is selected.
  - b** Press the **LAN Settings** softkey.
  - c** Press the **Addresses** softkey. Use the **IP Options** softkey and the Entry knob to select DHCP, AutoIP, or netBIOS. 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 return (up arrow) softkey.
  - d** Press the **Domain** softkey. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the Host name and the Domain name. When you are done, press the return (up arrow) softkey.

## Using the GPIB Interface

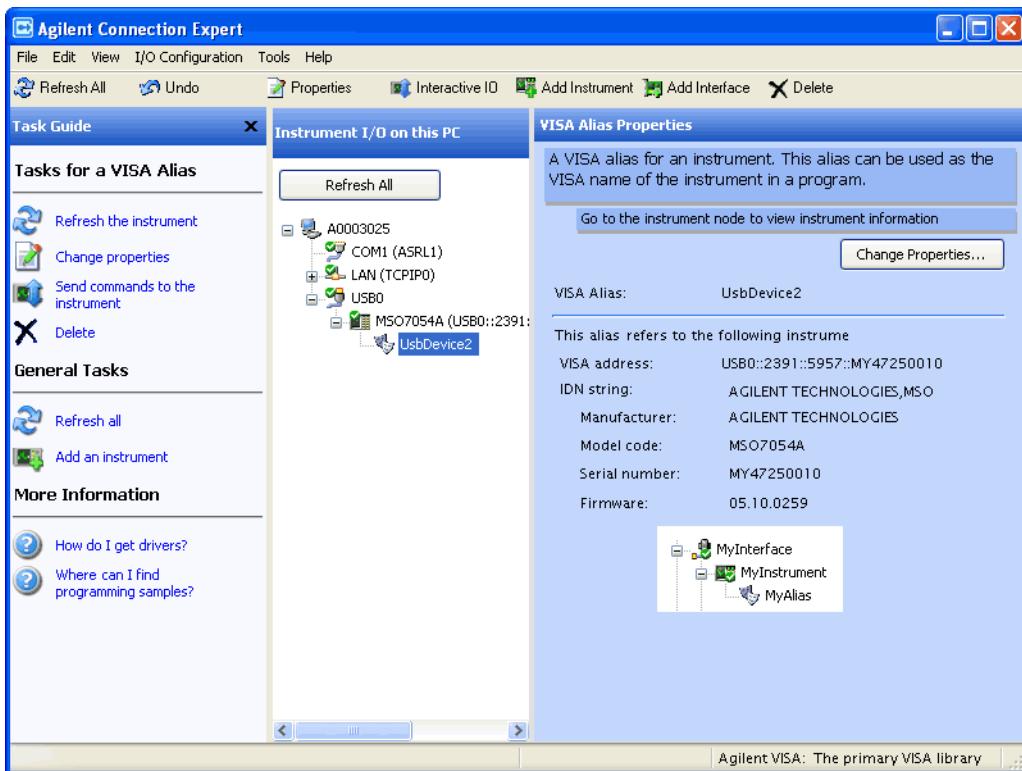
- 1** Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the back of the oscilloscope.
- 2** On the oscilloscope, verify that the controller interface is enabled:
  - a** Press the **Utility** button.
  - b** Using the softkeys, press **I/O** and **Control**.
  - c** Use the Entry knob to select "GPIB"; then, press the **Control** softkey again.  
Ensure the box next to **GPIB** is selected (). If not (, use the Entry knob to select **GPIB**; then, press the **Control** softkey again.
- 3** Configure the oscilloscope's GPIB interface:
  - a** Press the **Configure** softkey until "GPIB" is selected.
  - b** Use the Entry knob to select the **Address** value.

## Step 3. Verify the oscilloscope connection

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



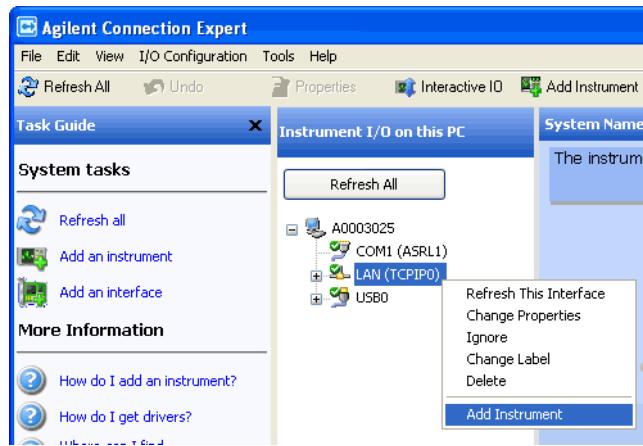
- 2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



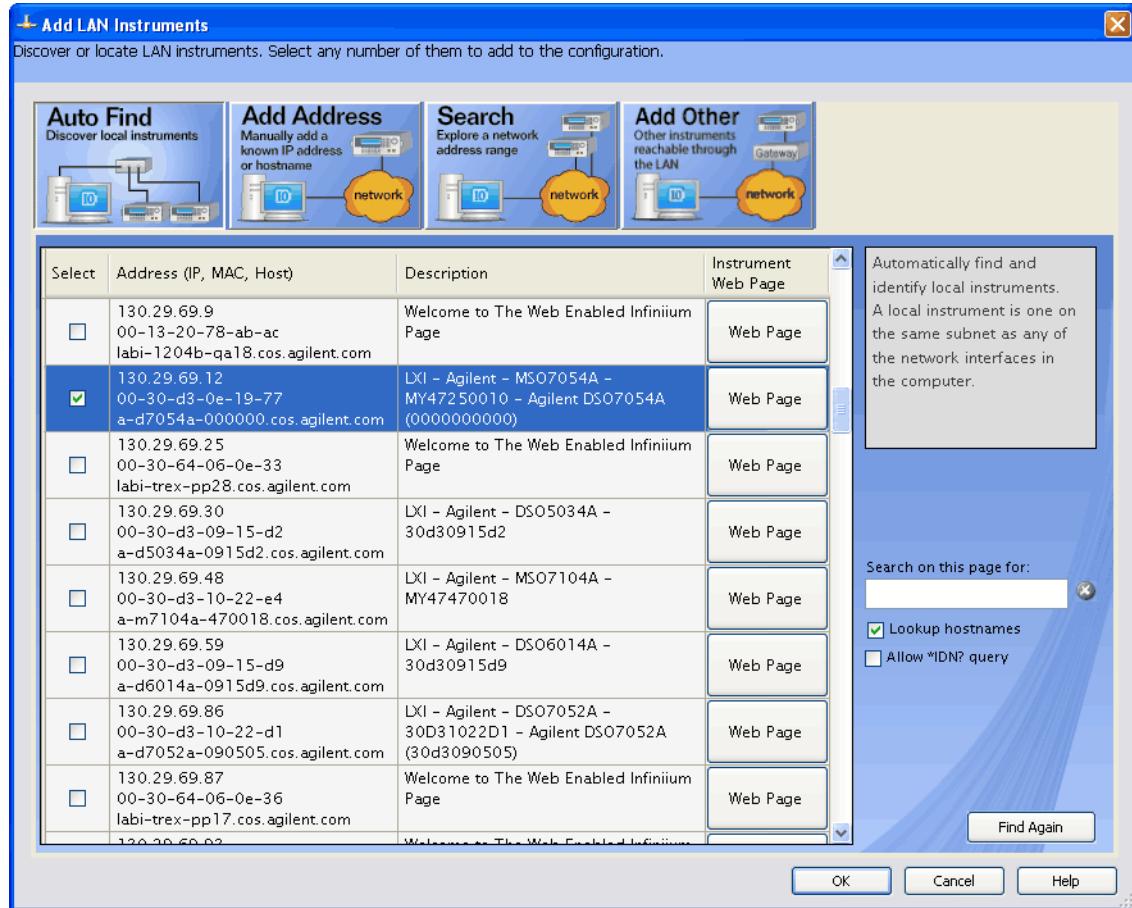
## 2 Setting Up

You must manually add instruments on LAN interfaces:

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



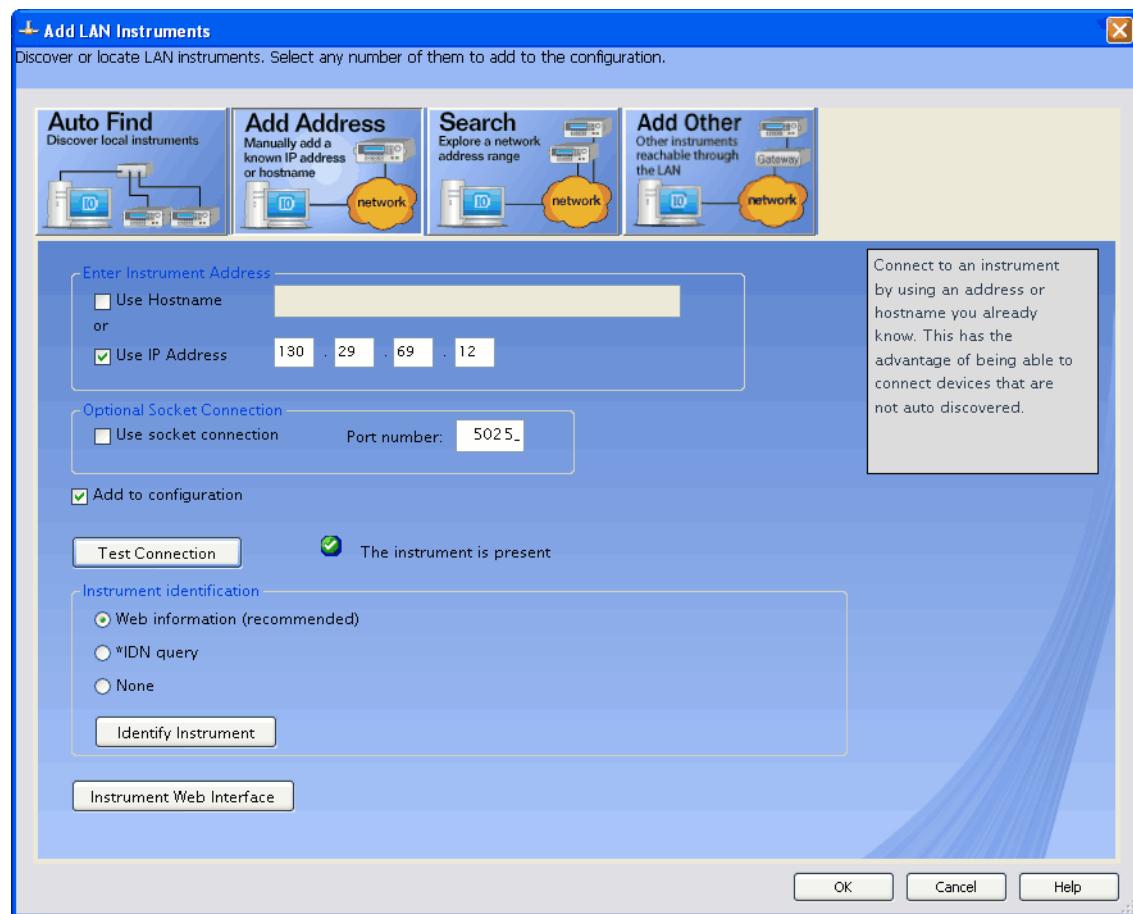
- b If the oscilloscope is on the same subnet, select it, and click **OK**.



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**.

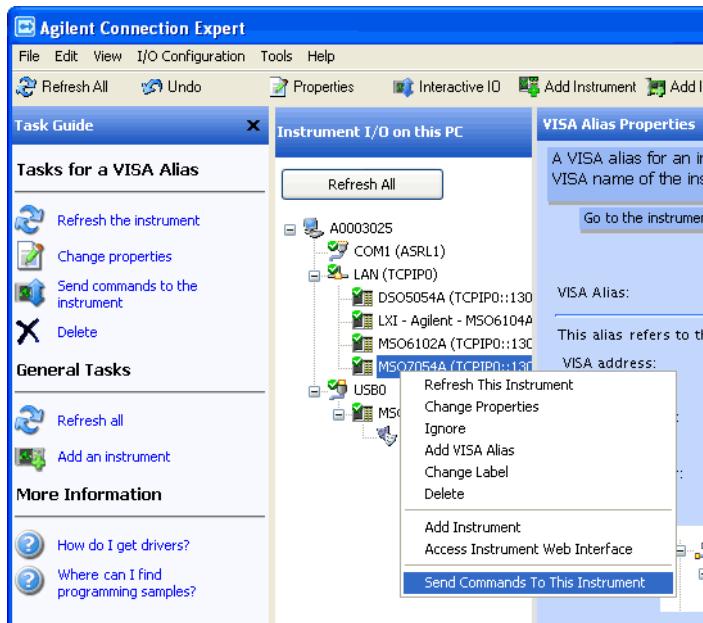
## 2 Setting Up



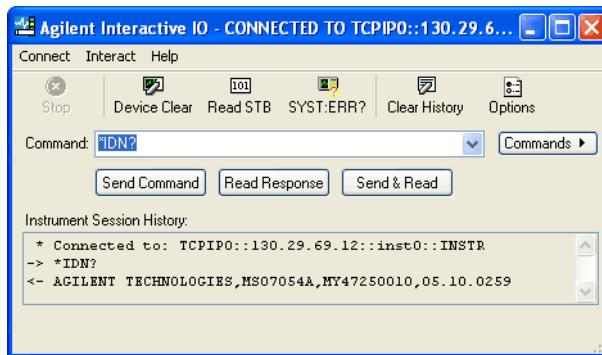
- iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

**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 42
- Programming the Oscilloscope 44
- Other Ways of Sending Commands 53

This chapter gives you an overview of programming the 5000 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**

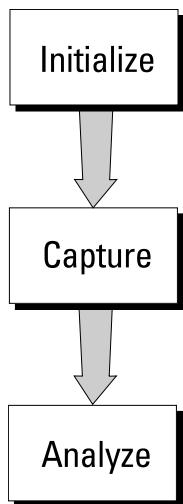
The programming examples in this guide are written in Visual Basic using the Agilent VISA COM library.

---



## 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 trace 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 44
- "Opening the Oscilloscope Connection via the IO Library" on page 45
- "Using :AUToscale to Automate Oscilloscope Setup" on page 46
- "Using Other Oscilloscope Setup Commands" on page 46
- "Capturing Data with the :DIGitize Command" on page 47
- "Reading Query Responses from the Oscilloscope" on page 49
- "Reading Query Results into String Variables" on page 50
- "Reading Query Results into Numeric Variables" on page 50
- "Reading Definite-Length Block Query Response Data" on page 50
- "Sending Multiple Queries and Reading Results" on page 51
- "Checking Instrument Status" on page 52

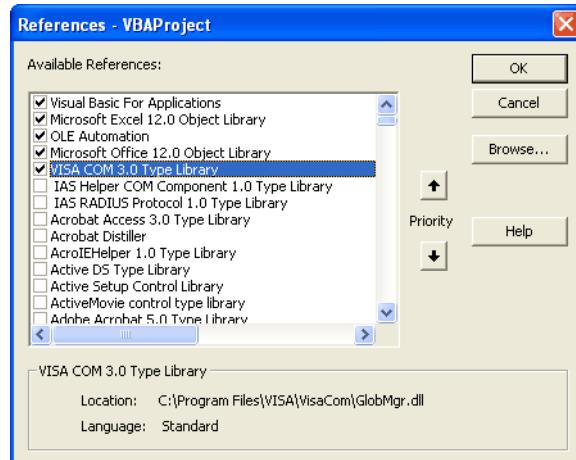
### Referencing the IO Library

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 659.

## 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.
myScope.IO.Clear
```

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 "[Common \(\\*\) Commands](#)" on page 97.

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  $\mu$ 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

' 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 -.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 -.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 "[:WAVeform Commands](#)" on page 500.

#### 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 in ["Commands by Subsystem" on page 95](#) 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
strQueryResult = myScope.ReadNumber
MsgBox "Range (variant):" + CStr(varQueryResult)
```

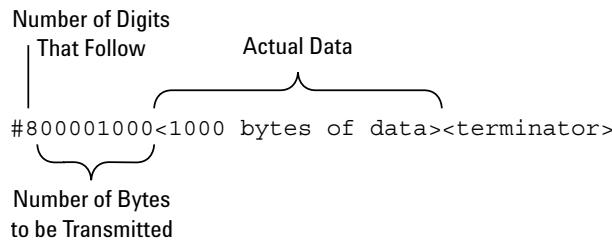
After running this program, the controller displays:

**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(ASCIIIType_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 "[Status Reporting](#)" on page 623 which explains how to check the status of the instrument.

## Other Ways of Sending Commands

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

### Telnet Sockets

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

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

```
telnet <hostname> 5024
```

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

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

```
telnet <hostname> 5025
```

### Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the *5000 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 56

Syntax Elements 92



## Command Summary

**Table 2** Common (\*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 101</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 102</a> )	*ESE? (see <a href="#">page 103</a> )	<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 <a href="#">page 104</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 104</a> )	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 <a href="#">page 107</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 108</a> )	*OPC? (see <a href="#">page 108</a> )	ASCII "1" is placed in the output queue when all pending device operations have completed.

**Table 2** Common (\*) Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	*OPT? (see <a href="#">page 109</a> )	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Secure>, <reserved>, <reserved>, <reserved>, <RS-232/UART Serial>, <reserved> <All field> ::= {0   All} <reserved> ::= 0 <Low Speed Serial> ::= {0   LSS} <Automotive Serial> ::= {0   AMS} <Secure> ::= {0   SEC} <RS-232/UART Serial> ::= {0   232}
*RCL <value> (see <a href="#">page 110</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*RST (see <a href="#">page 111</a> )	n/a	See *RST (Reset) (see <a href="#">page 111</a> )
*SAV <value> (see <a href="#">page 114</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*SRE <mask> (see <a href="#">page 115</a> )	*SRE? (see <a href="#">page 116</a> )	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:  Bit Weight Name Enables --- ----- --- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger

**Table 2** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see <a href="#">page 117</a> )	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following:  Bit Weight Name "1" Indicates --- ----- --- 7 128 OPER Operation status condition occurred. 6 64 RQS/ Instrument is MSS 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 <a href="#">page 119</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 120</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 121</a> )	n/a	n/a

**Table 3** Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see <a href="#">page 125</a> )	{0   1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see <a href="#">page 126</a> )	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value> (see <a href="#">page 128</a> )	:AUToscale:AMODE? (see <a href="#">page 128</a> )	<value> ::= {NORMal   CURRent}
:AUToscale:CHANnels <value> (see <a href="#">page 129</a> )	:AUToscale:CHANnels? (see <a href="#">page 129</a> )	<value> ::= {ALL   DISPlayed}
:BLANK [<source>] (see <a href="#">page 130</a> )	n/a	<source> ::= {CHANnel<n>}   FUNCTION   MATH <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see <a href="#">page 131</a> )	n/a	n/a

**Table 3** Root (:) Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>																																				
:DIGItize [ <i>&lt;source&gt;</i> [,..., <i>&lt;source&gt;</i> ]] (see <a href="#">page 132</a> )	n/a	<i>&lt;source&gt;</i> ::= {CHANnel<n>   FUNCtion   MATH} <i>&lt;source&gt;</i> can be repeated up to 5 times <i>&lt;n&gt;</i> ::= 1-2 or 1-4 in NR1 format																																				
:HWEnable <n> (see <a href="#">page 134</a> )	:HWEnable? (see <a href="#">page 134</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
n/a	:HWERegister:CONDition? (see <a href="#">page 136</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
n/a	:HWERegister[:EVENT]? (see <a href="#">page 138</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
:MERGe <pixel memory> (see <a href="#">page 140</a> )	n/a	<i>&lt;pixel memory&gt;</i> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}																																				
:MTEnable <n> (see <a href="#">page 141</a> )	:MTEnable? (see <a href="#">page 141</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
n/a	:MTERegister[:EVENT]? (see <a href="#">page 143</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
:OPEE <n> (see <a href="#">page 145</a> )	:OPEE? (see <a href="#">page 146</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
n/a	:OPERregister:CONDition? (see <a href="#">page 147</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
n/a	:OPERregister[:EVENT]? (see <a href="#">page 149</a> )	<i>&lt;n&gt;</i> ::= 16-bit integer in NR1 format																																				
:OVLenable <mask> (see <a href="#">page 151</a> )	:OVLenable? (see <a href="#">page 152</a> )	<i>&lt;mask&gt;</i> ::= 16-bit integer in NR1 format as shown: <table style="margin-left: 20px;"> <tr><th>Bit</th><th>Weight</th><th>Input</th></tr> <tr><td>---</td><td>-----</td><td>-----</td></tr> <tr><td>10</td><td>1024</td><td>Ext Trigger Fault</td></tr> <tr><td>9</td><td>512</td><td>Channel 4 Fault</td></tr> <tr><td>8</td><td>256</td><td>Channel 3 Fault</td></tr> <tr><td>7</td><td>128</td><td>Channel 2 Fault</td></tr> <tr><td>6</td><td>64</td><td>Channel 1 Fault</td></tr> <tr><td>4</td><td>16</td><td>Ext Trigger OVL</td></tr> <tr><td>3</td><td>8</td><td>Channel 4 OVL</td></tr> <tr><td>2</td><td>4</td><td>Channel 3 OVL</td></tr> <tr><td>1</td><td>2</td><td>Channel 2 OVL</td></tr> <tr><td>0</td><td>1</td><td>Channel 1 OVL</td></tr> </table>	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
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 <a href="#">page 153</a> )	<i>&lt;value&gt;</i> ::= integer in NR1 format. See OVLenable for <i>&lt;value&gt;</i>																																				

## 4 Commands Quick Reference

**Table 3** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:PRINT [<options>] (see <a href="#">page 155</a> )	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 <a href="#">page 156</a> )	n/a	n/a
n/a	:SERial (see <a href="#">page 157</a> )	<return value> ::= unquoted string containing serial number
:SINGle (see <a href="#">page 158</a> )	n/a	n/a
n/a	:STATus? <display> (see <a href="#">page 159</a> )	{0   1} <display> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format
:STOP (see <a href="#">page 160</a> )	n/a	n/a
n/a	:TER? (see <a href="#">page 161</a> )	{0   1}
:VIEW <source> (see <a href="#">page 162</a> )	n/a	<source> ::= {CHANnel<n>   PMEMORY{0   1   2   3   4   5   6   7   8   9}   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format

**Table 4** :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see <a href="#">page 165</a> )	{1   0}
:ACQuire:COMPLETE <complete> (see <a href="#">page 166</a> )	:ACQuire:COMPLETE? (see <a href="#">page 166</a> )	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see <a href="#">page 167</a> )	:ACQuire:COUNT? (see <a href="#">page 167</a> )	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALIAS <mode> (see <a href="#">page 168</a> )	:ACQuire:DAALIAS? (see <a href="#">page 168</a> )	<mode> ::= {DISable   AUTO}
:ACQuire:MODE <mode> (see <a href="#">page 169</a> )	:ACQuire:MODE? (see <a href="#">page 169</a> )	<mode> ::= {RTIMe   ETIMe   SEGmented}

**Table 4** :ACQuire Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:ACQuire:POINTS? (see page 170)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMENTed:AN ALyze (see page 171)	n/a	n/a (with Option SGM)
:ACQuire:SEGMENTed:CO UNT <count> (see page 172)	:ACQuire:SEGMENTed:CO UNT? (see page 172)	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
:ACQuire:SEGMENTed:IN Dex <index> (see page 173)	:ACQuire:SEGMENTed:IN Dex? (see page 173)	<index> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATE? (see page 176)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 177)	:ACQuire:TYPE? (see page 177)	<type> ::= {NORMal   AVERage   HRESolution   PEAK}

**Table 5** :CALibrate Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:CALibrate:DATE? (see page 181)	<return value> ::= <day>, <month>, <year>; all in NR1 format
:CALibrate:LABEL <string> (see page 182)	:CALibrate:LABEL? (see page 182)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 183)	:CALibrate:OUTPut? (see page 183)	<signal> ::= {TRIGgers   SOURCE   DSOurce   MASK}
:CALibrate:START (see page 184)	n/a	n/a
n/a	:CALibrate:STATUS? (see page 185)	<return value> ::= ALL, <status_code>, <status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITch? (see page 186)	{PROTected   UNPProtected}

## 4 Commands Quick Reference

**Table 5** :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPeratur e? (see page 187)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 188)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

**Table 6** :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0   OFF}   {1   ON}} (see page 192)	:CHANnel<n>:BWLimit? (see page 192)	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 193)	:CHANnel<n>:COUPling? (see page 193)	<coupling> ::= {AC   DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{0   OFF}   {1   ON}} (see page 194)	:CHANnel<n>:DISPlay? (see page 194)	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 195)	:CHANnel<n>:IMPedance ? (see page 195)	<impedance> ::= {ONEMeg   FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{0   OFF}   {1   ON}} (see page 196)	:CHANnel<n>:INVert? (see page 196)	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABEL <string> (see page 197)	:CHANnel<n>:LABEL? (see page 197)	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 198)	:CHANnel<n>:OFFSet? (see page 198)	<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 199)	:CHANnel<n>:PROBe? (see page 199)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 200)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format

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

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 201)	:CHANnel<n>:PROBe:SKEW? (see page 201)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 202)	:CHANnel<n>:PROBe:STYPe? (see page 202)	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTectioN (see page 203)	:CHANnel<n>:PROTectioN? (see page 203)	{NORM   TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 204)	:CHANnel<n>:RANGE? (see page 204)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALE <scale>[suffix] (see page 205)	:CHANnel<n>:SCALE? (see page 205)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITS <units> (see page 206)	:CHANnel<n>:UNITs? (see page 206)	<units> ::= {VOLT   AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0   OFF}   {1   ON}} (see page 207)	:CHANnel<n>:VERNier? (see page 207)	{0   1} <n> ::= 1-2 or 1-4 in NR1 format

**Table 7** :DISPlay Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:DISPlay:CLEar (see page 210)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data> (see page 211)	:DISPlay:DATA? [<format>][,][<area>][,][<palette>] (see page 211)	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF   BMP   BMP8bit   PNG} (query) <area> ::= {GRATicule   SCReen} (query) <palette> ::= {MONochrome   GRAYscale   COLOR} (query) <display data> ::= data in IEEE 488.2 # format

## 4 Commands Quick Reference

**Table 7** :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:LABel {{0   OFF}   {1   ON}} (see page 213)	:DISPlay:LABel? (see page 213)	{0   1}
:DISPlay:LABList <binary block> (see page 214)	:DISPlay:LABList? (see page 214)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERSistence <value> (see page 215)	:DISPlay:PERSistence? (see page 215)	<value> ::= {MINimum   INFinite}
:DISPlay:SOURce <value> (see page 216)	:DISPlay:SOURce? (see page 216)	<value> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:DISPlay:VECTors {{1   ON}   {0   OFF}} (see page 217)	:DISPlay:VECTors? (see page 217)	{1   0}

**Table 8** :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 220)	:EXTernal:BWLimit? (see page 220)	<bwlimit> ::= {0   OFF}
:EXTernal:IMPedance <value> (see page 221)	:EXTernal:IMPedance? (see page 221)	<impedance> ::= {ONEMeg   FIFTy}
:EXTernal:PROBe <attenuation> (see page 222)	:EXTernal:PROBe? (see page 222)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXTernal:PROBe:ID? (see page 223)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXTernal:PROBe:STYPe <signal type> (see page 224)	:EXTernal:PROBe:STYPe? (see page 224)	<signal type> ::= {DIFFerential   SINGLE}
:EXTernal:PROTection[:CLEar] (see page 225)	:EXTernal:PROTection? (see page 225)	{NORM   TRIP}

**Table 8** :EXternal Trigger Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:EXternal:RANGE <range>[<suffix>] (see <a href="#">page 226</a> )	:EXternal:RANGE? (see <a href="#">page 226</a> )	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXternal:UNITS <units> (see <a href="#">page 227</a> )	:EXternal:UNITS? (see <a href="#">page 227</a> )	<units> ::= {VOLT   AMPere}

**Table 9** :FUNCTION Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:FUNCTION:CENTER <frequency> (see <a href="#">page 231</a> )	:FUNCTION:CENTER? (see <a href="#">page 231</a> )	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCTION:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 232</a> )	:FUNCTION:DISPlay? (see <a href="#">page 232</a> )	{0   1}
:FUNCTION:GOFT:OPERation <operation> (see <a href="#">page 233</a> )	:FUNCTION:GOFT:OPERation? (see <a href="#">page 233</a> )	<operation> ::= {ADD   SUBTract   MULTIply}
:FUNCTION:GOFT:SOURce 1 <source> (see <a href="#">page 234</a> )	:FUNCTION:GOFT:SOURce 1? (see <a href="#">page 234</a> )	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see <a href="#">page 235</a> )	:FUNCTION:GOFT:SOURce 2? (see <a href="#">page 235</a> )	<source> ::= CHANnel<n> <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURcel selection <n> ::= {1   2} for 2ch models
:FUNCTION:OFFSet <offset> (see <a href="#">page 236</a> )	:FUNCTION:OFFSet? (see <a href="#">page 236</a> )	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:OPERation <operation> (see <a href="#">page 237</a> )	:FUNCTION:OPERation? (see <a href="#">page 237</a> )	<operation> ::= {ADD   SUBTract   MULTIply   INTegrate   DIFFerentiate   FFT   SQRT}

**Table 9** :FUNCTION Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:FUNCTION:RANGE <range> (see <a href="#">page 238</a> )	:FUNCTION:RANGE? (see <a href="#">page 238</a> )	<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 DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed).  The range for the FFT function is 8 to 800 dBV.
:FUNCTION:REFERENCE <level> (see <a href="#">page 239</a> )	:FUNCTION:REFERENCE? (see <a href="#">page 239</a> )	<level> ::= the value at center screen in NR3 format.  The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:SCALE <scale value>[<suffix>] (see <a href="#">page 240</a> )	:FUNCTION:SCALE? (see <a href="#">page 240</a> )	<scale value> ::= integer in NR1 format  <suffix> ::= {V   dB}
:FUNCTION:SOURcel <source> (see <a href="#">page 241</a> )	:FUNCTION:SOURcel? (see <a href="#">page 241</a> )	<source> ::= {CHANnel<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models  <n> ::= {1   2} for 2ch models GOFT is only for FFT, INTegrate, DIFFerentiate, and SQRT operations.
:FUNCTION:SOURce2 <source> (see <a href="#">page 242</a> )	:FUNCTION:SOURce2? (see <a href="#">page 242</a> )	<source> ::= {CHANnel<n>   NONE} <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURcel selection  <n> ::= {1   2} for 2ch models
:FUNCTION:SPAN <span> (see <a href="#">page 243</a> )	:FUNCTION:SPAN? (see <a href="#">page 243</a> )	<span> ::= the current frequency span in NR3 format.  Legal values are 1 Hz to 100 GHz.
:FUNCTION:WINDOW <>window> (see <a href="#">page 244</a> )	:FUNCTION:WINDOW? (see <a href="#">page 244</a> )	<window> ::= {RECTangular   HANNing   FLATtop   BHARris}

**Table 10** :HARDcopy Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:HARDcopy:AREA <area> (see <a href="#">page 247</a> )	:HARDcopy:AREA? (see <a href="#">page 247</a> )	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see <a href="#">page 248</a> )	:HARDcopy:APRinter? (see <a href="#">page 248</a> )	<active_printer> ::= {<index>   <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 249</a> )	:HARDcopy:FACTors? (see <a href="#">page 249</a> )	{0   1}
:HARDcopy:FFEed {{0   OFF}   {1   ON}} (see <a href="#">page 250</a> )	:HARDcopy:FFEed? (see <a href="#">page 250</a> )	{0   1}
:HARDcopy:INKSaver { {0   OFF}   {1   ON}} (see <a href="#">page 251</a> )	:HARDcopy:INKSaver? (see <a href="#">page 251</a> )	{0   1}
:HARDcopy:LAYout <layout> (see <a href="#">page 252</a> )	:HARDcopy:LAYout? (see <a href="#">page 252</a> )	<layout> ::= {LANDscape   PORTRait}
:HARDcopy:PAlette <palette> (see <a href="#">page 253</a> )	:HARDcopy:PAlette? (see <a href="#">page 253</a> )	<palette> ::= {COLor   GRAYscale   NONE}
n/a	:HARDcopy:PRINTER:LIS T? (see <a href="#">page 254</a> )	<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 <a href="#">page 255</a> )	n/a	n/a

**Table 11** :MARKer Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MARKer:MODE <mode> (see <a href="#">page 258</a> )	:MARKer:MODE? (see <a href="#">page 258</a> )	<mode> ::= {OFF   MEASurement   MANual   WAVEform}
:MARKer:X1Position <position>[suffix] (see <a href="#">page 259</a> )	:MARKer:X1Position? (see <a href="#">page 259</a> )	<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 <a href="#">page 260</a> )	:MARKer:X1Y1source? (see <a href="#">page 260</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see <a href="#">page 261</a> )	:MARKer:X2Position? (see <a href="#">page 261</a> )	<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 <a href="#">page 262</a> )	:MARKer:X2Y2source? (see <a href="#">page 262</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see <a href="#">page 263</a> )	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see <a href="#">page 264</a> )	:MARKer:Y1Position? (see <a href="#">page 264</a> )	<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 <a href="#">page 265</a> )	:MARKer:Y2Position? (see <a href="#">page 265</a> )	<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 <a href="#">page 266</a> )	<return_value> ::= Y cursors delta value in NR3 format

**Table 12** :MEASure Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:CLEar (see page 274)	n/a	n/a
:MEASure:COUNTER [ <source/> ] (see page 275)	:MEASure:COUNTER? [ <source/> ] (see page 275)	<source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 276)	:MEASure:DEFine? DELay (see page 277)	<delay spec> ::= <edge_spec1>, <edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+   -} <occurrence> ::= integer
:MEASure:DEFine THresholds, <threshold spec> (see page 276)	:MEASure:DEFine? THresholds (see page 277)	<threshold spec> ::= {STANDARD}   {<threshold mode>, <upper>, <middle>, <lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DELay [ <source1>] [, &lt;source2&gt;] (see page 279)</source1>	:MEASure:DELay? [ <source1>] [, &lt;source2&gt;] (see page 279)</source1>	<source1,2> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 281)	:MEASure:DUTYcycle? [<source>] (see page 281)	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 282)	:MEASure:FALLtime? [<source>] (see page 282)	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format

**Table 12** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:FREQuency [<source>] (see page 283)	:MEASure:FREQuency? [<source>] (see page 283)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 284)	:MEASure:NWIDth? [<source>] (see page 284)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 285)	:MEASure:OVERshoot? [<source>] (see page 285)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 287)	:MEASure:PERiod? [<source>] (see page 287)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASE [<source1>] [,<source2>] (see page 288)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 288)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 289)	:MEASure:PREShoot? [<source>] (see page 289)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDTH [<source>] (see page 290)	:MEASure:PWIDTH? [<source>] (see page 290)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 291)	<result_list> ::= comma-separated list of measurement results

**Table 12** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:RISEtime [ <source/> ] (see page 294)	:MEASure:RISEtime? [ <source/> ] (see page 294)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [ <source/> ] (see page 295)	:MEASure:SDEViation? [ <source/> ] (see page 295)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1   ON} (see page 296)	:MEASure:SHOW? (see page 296)	{1}
:MEASure:SOURce <source1> [,<source2>] (see page 297)	:MEASure:SOURce? (see page 297)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH   EXTernal} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= {<source>   NONE}
:MEASure:STATistics <type> (see page 299)	:MEASure:STATistics? (see page 299)	<type> ::= {{ON   1}   CURRent   MEAN   MINimum   MAXimum   STDDev   COUNT} ON ::= all statistics returned
:MEASure:STATistics:INCrement (see page 300)	n/a	n/a
:MEASure:STATistics:RESet (see page 301)	n/a	n/a
n/a	:MEASure:TEDGE? <slope><occurrence>[,<source>] (see page 302)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition

**Table 12** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:MEASure:TValue? <value>, [<slope>]<occurrence> [,<source>] (see <a href="#">page 304</a> )	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMplitude [<source>] (see <a href="#">page 306</a> )	:MEASure:VAMplitude? [<source>] (see <a href="#">page 306</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<source>] (see <a href="#">page 307</a> )	:MEASure:VAverage? [<source>] (see <a href="#">page 307</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see <a href="#">page 308</a> )	:MEASure:VBASe? [<source>] (see <a href="#">page 308</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see <a href="#">page 309</a> )	:MEASure:VMAX? [<source>] (see <a href="#">page 309</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see <a href="#">page 310</a> )	:MEASure:VMIN? [<source>] (see <a href="#">page 310</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format

**Table 12** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:VPP [<source>] (see page 311)	:MEASure:VPP? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 288)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 312)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 313)	:MEASure:VRMS? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtim>[,<source>] (see page 314)	<vtim> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see page 315)	:MEASure:VTOP? [<source>] (see page 315)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:XMAX [<source>] (see page 316)	:MEASure:XMAX? [<source>] (see page 316)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 317)	:MEASure:XMIN? [<source>] (see page 317)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the minimum in NR3 format

**Table 13** :MTEST Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MTEST:AMASK:CREAtE (see <a href="#">page 323</a> )	n/a	n/a
:MTEST:AMASK:SOURce <source> (see <a href="#">page 324</a> )	:MTEST:AMASK:SOURce? (see <a href="#">page 324</a> )	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:MTEST:AMASK:UNITS <units> (see <a href="#">page 325</a> )	:MTEST:AMASK:UNITS? (see <a href="#">page 325</a> )	<units> ::= {CURRent   DIVisions}
:MTEST:AMASK:XDELta <value> (see <a href="#">page 326</a> )	:MTEST:AMASK:XDELta? (see <a href="#">page 326</a> )	<value> ::= X delta value in NR3 format
:MTEST:AMASK:YDELta <value> (see <a href="#">page 327</a> )	:MTEST:AMASK:YDELta? (see <a href="#">page 327</a> )	<value> ::= Y delta value in NR3 format
n/a	:MTEST:COUNT:FWAvefor ms? [CHANnel<n>] (see <a href="#">page 328</a> )	<failed> ::= number of failed waveforms in NR1 format
:MTEST:COUNT:RESet (see <a href="#">page 329</a> )	n/a	n/a
n/a	:MTEST:COUNT:TIME? (see <a href="#">page 330</a> )	<time> ::= elapsed seconds in NR3 format
n/a	:MTEST:COUNT:WAVeform s? (see <a href="#">page 331</a> )	<count> ::= number of waveforms in NR1 format
:MTEST:DATA <mask> (see <a href="#">page 332</a> )	:MTEST:DATA? (see <a href="#">page 332</a> )	<mask> ::= data in IEEE 488.2 # format.
:MTEST:DELetE (see <a href="#">page 333</a> )	n/a	n/a
:MTEST:ENABle {{0   OFF}   {1   ON}} (see <a href="#">page 334</a> )	:MTEST:ENABle? (see <a href="#">page 334</a> )	{0   1}
:MTEST:LOCK {{0   OFF}   {1   ON}} (see <a href="#">page 335</a> )	:MTEST:LOCK? (see <a href="#">page 335</a> )	{0   1}
:MTEST:OUTPut <signal> (see <a href="#">page 336</a> )	:MTEST:OUTPut? (see <a href="#">page 336</a> )	<signal> ::= {FAIL   PASS}
:MTEST:RMODe <rmode> (see <a href="#">page 337</a> )	:MTEST:RMODe? (see <a href="#">page 337</a> )	<rmode> ::= {FORever   TIME   SIGMa   WAveforms}

**Table 13** :MTEST Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MTEST:RMODE:FACTion: PRINT {{0   OFF}   {1   ON}} (see <a href="#">page 338</a> )	:MTEST:RMODE:FACTion: PRINT? (see <a href="#">page 338</a> )	{0   1}
:MTEST:RMODE:FACTion: SAVE {{0   OFF}   {1   ON}} (see <a href="#">page 339</a> )	:MTEST:RMODE:FACTion: SAVE? (see <a href="#">page 339</a> )	{0   1}
:MTEST:RMODE:FACTion: STOP {{0   OFF}   {1   ON}} (see <a href="#">page 340</a> )	:MTEST:RMODE:FACTion: STOP? (see <a href="#">page 340</a> )	{0   1}
:MTEST:RMODE:SIGMa <level> (see <a href="#">page 341</a> )	:MTEST:RMODE:SIGMa? (see <a href="#">page 341</a> )	<level> ::= from 0.1 to 9.3 in NR3 format
:MTEST:RMODE:TIME <seconds> (see <a href="#">page 342</a> )	:MTEST:RMODE:TIME? (see <a href="#">page 342</a> )	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAVeform s <count> (see <a href="#">page 343</a> )	:MTEST:RMODE:WAVeform s? (see <a href="#">page 343</a> )	<count> ::= number of waveforms in NR1 format
:MTEST:SCALe:BIND {{0   OFF}   {1   ON}} (see <a href="#">page 344</a> )	:MTEST:SCALe:BIND? (see <a href="#">page 344</a> )	{0   1}
:MTEST:SCALe:X1 <x1_value> (see <a href="#">page 345</a> )	:MTEST:SCALe:X1? (see <a href="#">page 345</a> )	<x1_value> ::= X1 value in NR3 format
:MTEST:SCALe:XDELta <xdelta_value> (see <a href="#">page 346</a> )	:MTEST:SCALe:XDELta? (see <a href="#">page 346</a> )	<xdelta_value> ::= X delta value in NR3 format
:MTEST:SCALe:Y1 <y1_value> (see <a href="#">page 347</a> )	:MTEST:SCALe:Y1? (see <a href="#">page 347</a> )	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALe:Y2 <y2_value> (see <a href="#">page 348</a> )	:MTEST:SCALe:Y2? (see <a href="#">page 348</a> )	<y2_value> ::= Y2 value in NR3 format
:MTEST:SOURce <source> (see <a href="#">page 349</a> )	:MTEST:SOURce? (see <a href="#">page 349</a> )	<source> ::= {CHANnel<n>   NONE} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
n/a	:MTEST:TITLe? (see <a href="#">page 350</a> )	<title> ::= a string of up to 128 ASCII characters

**Table 14** :RECall Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:RECall:FILEname <base_name> (see <a href="#">page 352</a> )	:RECall:FILEname? (see <a href="#">page 352</a> )	<base_name> ::= quoted ASCII string
:RECall:IMAGE[:START] [<file_spec>] (see <a href="#">page 353</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see <a href="#">page 354</a> )	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 <a href="#">page 355</a> )	:RECall:PWD? (see <a href="#">page 355</a> )	<path_name> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see <a href="#">page 356</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string

**Table 15** :SAVE Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:SAVE:FILEname <base_name> (see <a href="#">page 359</a> )	:SAVE:FILEname? (see <a href="#">page 359</a> )	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_spec>] (see <a href="#">page 360</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
n/a	:SAVE:IMAGE:AREA? (see <a href="#">page 361</a> )	<area> ::= {GRAT   SCR}

**Table 15** :SAVE Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:SAVE:IMAGE:FACTOrs { {0   OFF}   {1   ON}} (see <a href="#">page 362</a> )	:SAVE:IMAGE:FACTOrs? (see <a href="#">page 362</a> )	{0   1}
:SAVE:IMAGE:FORMAT <format> (see <a href="#">page 363</a> )	:SAVE:IMAGE:FORMAT? (see <a href="#">page 363</a> )	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGE:INKSaver { {0   OFF}   {1   ON}} (see <a href="#">page 364</a> )	:SAVE:IMAGE:INKSaver? (see <a href="#">page 364</a> )	{0   1}
:SAVE:IMAGE:PALETTE <palette> (see <a href="#">page 365</a> )	:SAVE:IMAGE:PALETTE? (see <a href="#">page 365</a> )	<palette> ::= {COLOR   GRAYscale   MONochrome}
:SAVE:MASK[:START] [<file_spec>] (see <a href="#">page 366</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see <a href="#">page 367</a> )	:SAVE:PWD? (see <a href="#">page 367</a> )	<path_name> ::= quoted ASCII string
:SAVE:SETUp[:START] [<file_spec>] (see <a href="#">page 368</a> )	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 <a href="#">page 369</a> )	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVEform:FORMAT <format> (see <a href="#">page 370</a> )	:SAVE:WAVEform:FORMAT? (see <a href="#">page 370</a> )	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVEform:LENGTH <length> (see <a href="#">page 371</a> )	:SAVE:WAVEform:LENGTH? (see <a href="#">page 371</a> )	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVEform:SEGmented <option> (see <a href="#">page 372</a> )	:SAVE:WAVEform:SEGmented? (see <a href="#">page 372</a> )	<option> ::= {ALL   CURRent}

**Table 16** :SBUS Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:SBUS:CAN:COUNT:ERRor? ? (see <a href="#">page 375</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:OVERload? ? (see <a href="#">page 376</a> )	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see <a href="#">page 377</a> )	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal ? (see <a href="#">page 378</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILization? ? (see <a href="#">page 379</a> )	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 380</a> )	:SBUS:DISPlay? ? (see <a href="#">page 380</a> )	{0   1}
:SBUS:IIC:ASIZE <size> (see <a href="#">page 381</a> )	:SBUS:IIC:ASIZE? ? (see <a href="#">page 381</a> )	<size> ::= {BIT7   BIT8}
:SBUS:LIN:PARity {{0   OFF}   {1   ON}} (see <a href="#">page 382</a> )	:SBUS:LIN:PARity? ? (see <a href="#">page 382</a> )	{0   1}
:SBUS:MODE <mode> (see <a href="#">page 383</a> )	:SBUS:MODE? ? (see <a href="#">page 383</a> )	<mode> ::= {IIC   SPI   CAN   LIN   FLEXray   UART}
:SBUS:SPI:WIDTh <word_width> (see <a href="#">page 384</a> )	:SBUS:SPI:WIDTh? ? (see <a href="#">page 384</a> )	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see <a href="#">page 385</a> )	:SBUS:UART:BASE? ? (see <a href="#">page 385</a> )	<base> ::= {ASCII   BINary   HEX}
n/a	:SBUS:UART:COUNT:ERRo r? ? (see <a href="#">page 386</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESe t (see <a href="#">page 387</a> )	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFR ames? ? (see <a href="#">page 388</a> )	<frame_count> ::= integer in NR1 format

**Table 16** :SBUS Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:SBUS:UART:COUNT:TXFR ames? (see <a href="#">page 389</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see <a href="#">page 390</a> )	:SBUS:UART:FRAMing? (see <a href="#">page 390</a> )	<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

**Table 17** :SYSTem Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:SYSTem:DATE <date> (see <a href="#">page 392</a> )	:SYSTem:DATE? (see <a href="#">page 392</a> )	<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 <a href="#">page 393</a> )	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see <a href="#">page 394</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 615</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 395</a> )	:SYSTem:LOCK? (see <a href="#">page 395</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PROTection:LO CK <value> (see <a href="#">page 396</a> )	:SYSTem:PROTection:LO CK? (see <a href="#">page 396</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 397</a> )	:SYSTem:SETup? (see <a href="#">page 397</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 399</a> )	:SYSTem:TIME? (see <a href="#">page 399</a> )	<time> ::= hours,minutes,seconds in NR1 format

## 4 Commands Quick Reference

**Table 18** :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see page 402)	:TIMEbase:MODE? (see page 402)	<value> ::= {MAIN   WINDOW   XY   ROLL}
:TIMEbase:POSIon <pos> (see page 403)	:TIMEbase:POSIon? (see page 403)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMEbase:RANGE <range_value> (see page 404)	:TIMEbase:RANGE? (see page 404)	<range_value> ::= 10 ns through 500 s in NR3 format
:TIMEbase:REFERENCE {LEFT   CENTER   RIGHT} (see page 405)	:TIMEbase:REFERENCE? (see page 405)	<return_value> ::= {LEFT   CENTER   RIGHT}
:TIMEbase:SCALe <scale_value> (see page 406)	:TIMEbase:SCALe? (see page 406)	<scale_value> ::= scale value in seconds in NR3 format
:TIMEbase:VERNier {{0   OFF}   {1   ON}} (see page 407)	:TIMEbase:VERNier? (see page 407)	{0   1}
:TIMEbase:WINDOW:POSIon <pos> (see page 408)	:TIMEbase:WINDOW:POSIon? (see page 408)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMEbase:WINDOW:RANGe <range_value> (see page 409)	:TIMEbase:WINDOW:RANGe? (see page 409)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALe <scale_value> (see page 410)	:TIMEbase:WINDOW:SCALe? (see page 410)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

**Table 19** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}   {1   ON}} (see page 415)	:TRIGger:HFReject? (see page 415)	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see page 416)	:TRIGger:HOLDoff? (see page 416)	<holdoff_time> ::= 60 ns to 10 s in NR3 format

**Table 19** General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:MODE <mode> (see <a href="#">page 417</a> )	:TRIGger:MODE? (see <a href="#">page 417</a> )	<mode> ::= {EDGE   GLITCH   PATTern   DURation   TV} <return_value> ::= {<mode>   <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREJect {{0   OFF}   {1   ON}} (see <a href="#">page 418</a> )	:TRIGger:NREJect? (see <a href="#">page 418</a> )	{0   1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>] (see <a href="#">page 419</a> )	:TRIGger:PATTern? (see <a href="#">page 419</a> )	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnn"; n ::= {0,...,9   A,...,F} (# bits = # channels) <edge source> ::= {CHANnel<n>   EXTERNAL   NONE} <edge> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see <a href="#">page 421</a> )	:TRIGger:SWEep? (see <a href="#">page 421</a> )	<sweep> ::= {AUTO   NORMAL}

**Table 20** :TRIGger:CAN Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:CAN:PATTern:DATA <value>, <mask> (see <a href="#">page 424</a> )	:TRIGger:CAN:PATTern:DATA? (see <a href="#">page 424</a> )	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-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
:TRIGger:CAN:PATTern:DATA:LENGTH <length> (see <a href="#">page 425</a> )	:TRIGger:CAN:PATTern:DATA:LENGTH? (see <a href="#">page 425</a> )	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATTern:ID <value>, <mask> (see <a href="#">page 426</a> )	:TRIGger:CAN:PATTern:ID? (see <a href="#">page 426</a> )	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <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
:TRIGger:CAN:PATTern:ID:MODE <value> (see <a href="#">page 427</a> )	:TRIGger:CAN:PATTern:ID:MODE? (see <a href="#">page 427</a> )	<value> ::= {STANDARD   EXTENDED} (with Option AMS)
:TRIGger:CAN:SAMPLEport <value> (see <a href="#">page 428</a> )	:TRIGger:CAN:SAMPLEport? (see <a href="#">page 428</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:CAN:SIGNAl:B AUDrate <baudrate> (see <a href="#">page 429</a> )	:TRIGger:CAN:SIGNAl:B AUDrate? (see <a href="#">page 429</a> )	<baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

**Table 20** :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see page 430)	:TRIGger:CAN:SOURce? (see <a href="#">page 430</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGItal0,..,DIGItal15   } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 431)	:TRIGger:CAN:TRIGger? (see <a href="#">page 432</a> )	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF   DATA   ERRor   IDData   IDEither   IDRmote   ALLerrors   OVERload   ACKerror} (with Option AMS)

**Table 21** :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see <a href="#">page 434</a> )	:TRIGger:DURation:GREaterthan? (see <a href="#">page 434</a> )	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see <a href="#">page 435</a> )	:TRIGger:DURation:LESSthan? (see <a href="#">page 435</a> )	<less_than_time> ::= floating-point number from in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:PATtern <value>, <mask> (see <a href="#">page 436</a> )	:TRIGger:DURation:PATtern? (see <a href="#">page 436</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= ""0xnnnnnnn"" n ::= {0,...,9   A,...,F}
:TRIGger:DURation:QUALifier <qualifier> (see <a href="#">page 437</a> )	:TRIGger:DURation:QUALifier? (see <a href="#">page 437</a> )	<qualifier> ::= {GREaterthan   LESSthan   INRange   OUTRange   TIMEOUT}
:TRIGger:DURation:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see <a href="#">page 438</a> )	:TRIGger:DURation:RANGE? (see <a href="#">page 438</a> )	<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}

## 4 Commands Quick Reference

**Table 22** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LF} (see <a href="#">page 440</a> )	:TRIGger[:EDGE]:COUPling? (see <a href="#">page 440</a> )	{AC   DC   LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see <a href="#">page 441</a> )	:TRIGger[:EDGE]:LEVel? [<source>] (see <a href="#">page 441</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LF   HF} (see <a href="#">page 442</a> )	:TRIGger[:EDGE]:REJect? (see <a href="#">page 442</a> )	{OFF   LF   HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see <a href="#">page 443</a> )	:TRIGger[:EDGE]:SLOPe? (see <a href="#">page 443</a> )	<polarity> ::= {POSitive   NEGative   EITHer   ALTerNate}
:TRIGger[:EDGE]:SOURce <source> (see <a href="#">page 444</a> )	:TRIGger[:EDGE]:SOURce? (see <a href="#">page 444</a> )	<source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format

**Table 23** :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREAterthan <greater_than_time>[suffix] (see <a href="#">page 446</a> )	:TRIGger:GLITch:GREAterthan? (see <a href="#">page 446</a> )	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see <a href="#">page 447</a> )	:TRIGger:GLITch:LESSthan? (see <a href="#">page 447</a> )	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}

**Table 23** :TRIGger:GLITch Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:GLITch:LEVel <level> [<source>] (see page 448)	:TRIGger:GLITch:LEVel? ? (see page 448)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 449)	:TRIGger:GLITch:POLarity? ? (see page 449)	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 450)	:TRIGger:GLITch:QUALifier? ? (see page 450)	<qualifier> ::= {GREaterthan   LESSthan   RANGE}
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 451)	:TRIGger:GLITch:RANGE? ? (see page 451)	<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 452)	:TRIGger:GLITch:SOURce? ? (see page 452)	<source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format

**Table 24** :TRIGger:IIC Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:IIC:PATTERn:ADDResS <value> (see page 454)	:TRIGger:IIC:PATTERn:ADDResS? ? (see page 454)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATTERn:DATA <value> (see page 455)	:TRIGger:IIC:PATTERn:DATA? ? (see page 455)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATTERn:DATA2 <value> (see page 456)	:TRIGger:IIC:PATTERn:DATA2? ? (see page 456)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}

## 4 Commands Quick Reference

**Table 24** :TRIGger:IIC Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:IIC[:SOURce] :CLOCk <source> (see page 457)	:TRIGger:IIC[:SOURce] :CLOCK? (see page 457)	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce] :DATA <source> (see page 458)	:TRIGger:IIC[:SOURce] :DATA? (see page 458)	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGGER: QUALifier <value> (see page 459)	:TRIGger:IIC:TRIGger: QUALifier? (see page 459)	<value> ::= {EQUal   NOTequal   LESSthan   GREaterthan}
:TRIGger:IIC:TRIGGER[: TYPE] <type> (see page 460)	:TRIGger:IIC:TRIGger[: TYPE]? (see page 460)	<type> ::= {START   STOP   READ7   READEprom   WRITE7   WRITE10   NACKnowledge   ANACKnowledge   R7Data2   W7Data2   RESTart}

**Table 25** :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see page 463)	:TRIGger:LIN:ID? (see page 463)	<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
:TRIGger:LIN:SAMPLEpo int <value> (see page 464)	:TRIGger:LIN:SAMPLEpo int? (see page 464)	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:LIN:SIGNAl:B AUDrate <baudrate> (see page 465)	:TRIGger:LIN:SIGNAl:B AUDrate? (see page 465)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments

**Table 25** :TRIGger:LIN Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:LIN:SOURce <source> (see page 466)	:TRIGger:LIN:SOURce? (see <a href="#">page 466</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGITAL0,..,DIGITAL15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANDARD <std> (see <a href="#">page 467</a> )	:TRIGger:LIN:STANDARD? (see <a href="#">page 467</a> )	<std> ::= {LIN13   LIN20}
:TRIGger:LIN:SYNCbreak <value> (see page 468)	:TRIGger:LIN:SYNCbreak? (see <a href="#">page 468</a> )	<value> ::= integer = {11   12   13}
:TRIGger:LIN:TRIGGER <condition> (see page 469)	:TRIGger:LIN:TRIGGER? (see <a href="#">page 469</a> )	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak   ID} (with Option AMS)

**Table 26** :TRIGger:TV Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:TV:LINE <line number> (see <a href="#">page 480</a> )	:TRIGger:TV:LINE? (see <a href="#">page 480</a> )	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see <a href="#">page 481</a> )	:TRIGger:TV:MODE? (see <a href="#">page 481</a> )	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LINE   VERTICAL   LFIELD1   LFIELD2   LALTERNATE   LVERTICAL}
:TRIGger:TV:Polarity <polarity> (see <a href="#">page 482</a> )	:TRIGger:TV:Polarity? (see <a href="#">page 482</a> )	<polarity> ::= {POSITIVE   NEGATIVE}
:TRIGger:TV:SOURce <source> (see <a href="#">page 483</a> )	:TRIGger:TV:SOURce? (see <a href="#">page 483</a> )	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANDARD <standard> (see <a href="#">page 484</a> )	:TRIGger:TV:STANDARD? (see <a href="#">page 484</a> )	<standard> ::= {GENERIC   NTSC   PALM   PAL   SECAM   {P480L60HZ   P480}   {P720L60HZ   P720}   {P1080L24HZ   P1080}   P1080L25HZ   {I1080L50HZ   I1080}   I1080L60HZ}

**Table 27** :TRIGger:UART Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:UART:BASE <base> (see <a href="#">page 487</a> )	:TRIGger:UART:BASE? (see <a href="#">page 487</a> )	<base> ::= {ASCII   HEX}
:TRIGger:UART:BAUDrate <baudrate> (see <a href="#">page 488</a> )	:TRIGger:UART:BAUDrate? (see <a href="#">page 488</a> )	<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments
:TRIGger:UART:BITorder <bitorder> (see <a href="#">page 489</a> )	:TRIGger:UART:BITorder? (see <a href="#">page 489</a> )	<bitorder> ::= {LSBFIRST   MSBFIRST}
:TRIGger:UART:BURSt <value> (see <a href="#">page 490</a> )	:TRIGger:UART:BURSt? (see <a href="#">page 490</a> )	<value> ::= {OFF   1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see <a href="#">page 491</a> )	:TRIGger:UART:DATA? (see <a href="#">page 491</a> )	<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)
:TRIGger:UART:IDLE <time_value> (see <a href="#">page 492</a> )	:TRIGger:UART:IDLE? (see <a href="#">page 492</a> )	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see <a href="#">page 493</a> )	:TRIGger:UART:PARity? (see <a href="#">page 493</a> )	<parity> ::= {EVEN   ODD   NONE}
:TRIGger:UART:Polarit y <polarity> (see <a href="#">page 494</a> )	:TRIGger:UART:Polarit y? (see <a href="#">page 494</a> )	<polarity> ::= {HIGH   LOW}
:TRIGger:UART:QUALifier <value> (see <a href="#">page 495</a> )	:TRIGger:UART:QUALifier? (see <a href="#">page 495</a> )	<value> ::= {EQUAL   NOTEQUAL   GREATERthan   LESSthan}
:TRIGger:UART:SOURce:RX <source> (see <a href="#">page 496</a> )	:TRIGger:UART:SOURce:RX? (see <a href="#">page 496</a> )	<source> ::= {CHANNEL<n>   EXTERNAL} for DSO models <source> ::= {CHANNEL<n>   DIGITAL0,...,DIGITAL15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

**Table 27** :TRIGger:UART Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:UART:SOURce: TX <source> (see <a href="#">page 497</a> )	:TRIGger:UART:SOURce: TX? (see <a href="#">page 497</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGItal0,..,DIGItal15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see <a href="#">page 498</a> )	:TRIGger:UART:TYPE? (see <a href="#">page 498</a> )	<value> ::= {RSTArt   RSTOP   RDATA   RD1   RD0   RDX   PARityerror   TSTArt   TSTOP   TDATA   TD1   TD0   TDX}
:TRIGger:UART:WIDTH <width> (see <a href="#">page 499</a> )	:TRIGger:UART:WIDTH? (see <a href="#">page 499</a> )	<width> ::= {5   6   7   8   9}

**Table 28** :WAVEform Commands Summary

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:WAVEform:BYTeorder <value> (see <a href="#">page 507</a> )	:WAVEform:BYTeorder? (see <a href="#">page 507</a> )	<value> ::= {LSBFFirst   MSBFFirst}
n/a	:WAVEform:COUNT? (see <a href="#">page 508</a> )	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see <a href="#">page 509</a> )	<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 <a href="#">page 511</a> )	:WAVEform:FORMAT? (see <a href="#">page 511</a> )	<value> ::= {WORD   BYTE   ASCII}

**Table 28** :WAVEform Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:WAVEform:POINTs <# points> (see page 512)	:WAVEform:POINTs? (see <a href="#">page 512</a> )	<# 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 514)	:WAVEform:POINTs:MODE? (see <a href="#">page 514</a> )	<points_mode> ::= {NORMal   MAXimum   RAW}
n/a	:WAVEform:PREamble? (see <a href="#">page 516</a> )	<preamble_block> ::= <format NR1>, <type NR1>, <points NR1>, <count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>, <yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: <ul style="list-style-type: none"><li>• 0 for BYTE format</li><li>• 1 for WORD format</li><li>• 2 for ASCII format</li></ul> <type> ::= an integer in NR1 format: <ul style="list-style-type: none"><li>• 0 for NORMAL type</li><li>• 1 for PEAK detect type</li><li>• 2 for AVERAGE type</li><li>• 3 for HRESolution type</li></ul> <count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format
n/a	:WAVEform:SEGmented:COUNT? (see <a href="#">page 519</a> )	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see <a href="#">page 520</a> )	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see page 521)	:WAVEform:SOURce? (see <a href="#">page 521</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format

**Table 28** :WAVEform Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:WAVEform:SOURce:SUBS ource <subsource> (see page 525)	:WAVEform:SOURce:SUBS ource? (see <a href="#">page 525</a> )	<subsource> ::= {{NONE   RX}   TX}
n/a	:WAVEform:TYPE? (see <a href="#">page 526</a> )	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVEform:UNSIGNED { {0   OFF}   {1   ON} } (see <a href="#">page 527</a> )	:WAVEform:UNSIGNED? (see <a href="#">page 527</a> )	{0   1}
:WAVEform:VIEW <view> (see <a href="#">page 528</a> )	:WAVEform:VIEW? (see <a href="#">page 528</a> )	<view> ::= {MAIN}
n/a	:WAVEform:XINCREMENT? (see <a href="#">page 529</a> )	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVEform:XORIGIN? (see <a href="#">page 530</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVEform:XREFERENCE? (see <a href="#">page 531</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVEform:YINCREMENT? (see <a href="#">page 532</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVEform:YORIGIN? (see <a href="#">page 533</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVEform:YREFERENCE? (see <a href="#">page 534</a> )	<return_value> ::= y-reference value in the current preamble in NR1 format

## Syntax Elements

- "Number Format" on page 92
- "<NL> (Line Terminator)" on page 92
- "[ ] (Optional Syntax Terms)" on page 92
- "{ } (Braces)" on page 92
- "::= (Defined As)" on page 92
- "< > (Angle Brackets)" on page 93
- "... (Ellipsis)" on page 93
- "n,..,p (Value Ranges)" on page 93
- "d (Digits)" on page 93
- "Quoted ASCII String" on page 93
- "Definite-Length Block Response Data" on page 93

### Number Format

NR1 specifies integer data.

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

### <NL> (Line Terminator)

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

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

### [ ] (Optional Syntax Terms)

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

### { } (Braces)

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

### ::= (Defined As)

::= means "defined as".

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

## <> (Angle Brackets)

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

## ... (Ellipsis)

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

## n,...,p (Value Ranges)

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

## d (Digits)

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

## Quoted ASCII String

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

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

has a quoted ASCII string of:

```
'One'
```

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

## Definite-Length Block Response Data

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

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

## 4 Commands Quick Reference

#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

## Commands by Subsystem

Subsystem	Description
<a href="#">"Common (*) Commands" on page 97</a>	Commands defined by IEEE 488.2 standard that are common to all instruments.
<a href="#">"Root (:) Commands" on page 122</a>	Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.
<a href="#">":ACQuire Commands" on page 163</a>	Set the parameters for acquiring and storing data.
<a href="#">":CALibrate Commands" on page 179</a>	Utility commands for determining the state of the calibration factor protection switch.
<a href="#">":CHANnel&lt;n&gt; Commands" on page 189</a>	Control all oscilloscope functions associated with individual analog channels or groups of channels.
<a href="#">":DISPlay Commands" on page 208</a>	Control how waveforms, graticule, and text are displayed and written on the screen.
<a href="#">":EXTernal Trigger Commands" on page 218</a>	Control the input characteristics of the external trigger input.
<a href="#">":FUNCtion Commands" on page 228</a>	Control functions in the measurement/storage module.
<a href="#">":HARDcopy Commands" on page 245</a>	Set and query the selection of hardcopy device and formatting options.
<a href="#">":MARKer Commands" on page 256</a>	Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
<a href="#">":MEASure Commands" on page 267</a>	Select automatic measurements to be made and control time markers.
<a href="#">":MTEST Commands" on page 318</a>	Control the mask test features provided with Option LMT.
<a href="#">":RECall Commands" on page 351</a>	Recall previously saved oscilloscope setups and traces.



Subsystem	Description
<a href="#">":SAVE Commands" on page 357</a>	Save oscilloscope setups and traces, screen images, and data.
<a href="#">":SBUS Commands" on page 373</a>	Control oscilloscope functions associated with the serial decode bus.
<a href="#">":SYSTem Commands" on page 391</a>	Control basic system functions of the oscilloscope.
<a href="#">":TIMEbase Commands" on page 400</a>	Control all horizontal sweep functions.
<a href="#">":TRIGger Commands" on page 411</a>	Control the trigger modes and parameters for each trigger type.
<a href="#">":WAVeform Commands" on page 500</a>	Provide access to waveform data.

**Command Types** Three types of commands are used:

- **Common (\*) Commands** – See ["Introduction to Common \(\\*\) Commands" on page 99](#) for more information.
- **Root Level (:) Commands** – See ["Introduction to Root \(:\) Commands" on page 124](#) for more information.
- **Subsystem Commands** – Subsystem commands are grouped together under a common node of the ["Command Tree" on page 663](#), such as the :TIMEbase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree; therefore, no subsystem is selected.

## Common (\*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See "Introduction to Common (\*) Commands" on page 99.

**Table 29** Common (\*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 101</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 102</a> )	*ESE? (see <a href="#">page 103</a> )	<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 <a href="#">page 104</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 104</a> )	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 <a href="#">page 107</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 108</a> )	*OPC? (see <a href="#">page 108</a> )	ASCII "1" is placed in the output queue when all pending device operations have completed.

## 5 Commands by Subsystem

**Table 29** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns																																				
n/a	*OPT? (see <a href="#">page 109</a> )	<return_value> ::= 0,0,<license info> <license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Secure>, <reserved>, <reserved>, <reserved>, <RS-232/UART Serial>, <reserved> <All field> ::= {0   All} <reserved> ::= 0 <Low Speed Serial> ::= {0   LSS} <Automotive Serial> ::= {0   AMS} <Secure> ::= {0   SEC} <RS-232/UART Serial> ::= {0   232}																																				
*RCL <value> (see <a href="#">page 110</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}																																				
*RST (see <a href="#">page 111</a> )	n/a	See *RST (Reset) (see <a href="#">page 111</a> )																																				
*SAV <value> (see <a href="#">page 114</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}																																				
*SRE <mask> (see <a href="#">page 115</a> )	*SRE? (see <a href="#">page 116</a> )	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: <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																																			

**Table 29** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see <a href="#">page 117</a> )	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following:  Bit Weight Name "1" Indicates --- ----- 7 128 OPER Operation status condition occurred. 6 64 RQS/ Instrument is MSS 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 <a href="#">page 119</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 120</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 121</a> )	n/a	n/a

### Introduction to Common (\*) Commands

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

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

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

## 5 Commands by Subsystem

### 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 658)

### 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 99
- "["\\*STB \(Read Status Byte\)"](#) on page 117
- "["\\*ESE \(Standard Event Status Enable\)"](#) on page 102
- "["\\*ESR \(Standard Event Status Register\)"](#) on page 104
- "["\\*SRE \(Service Request Enable\)"](#) on page 115
- "[":SYSTem:ERRor"](#) on page 394

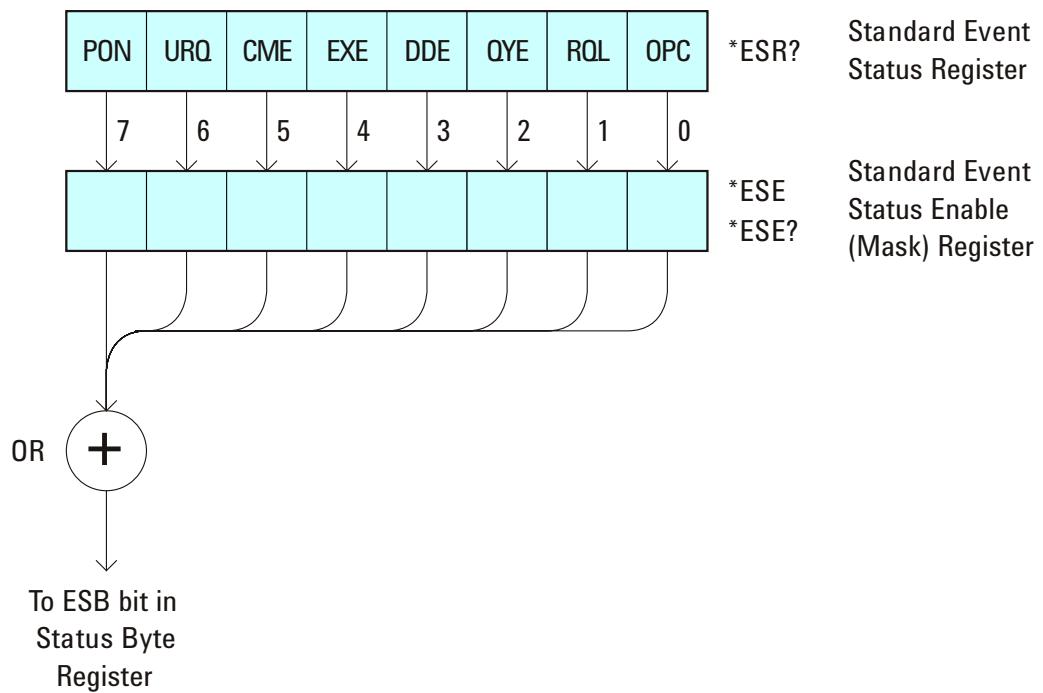
**\*ESE (Standard Event Status Enable)**

**C** (see page 658)

**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 30** 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.

**Table 30** Standard Event Status Enable (ESE) (continued)

<b>Bit</b>	<b>Name</b>	<b>Description</b>	<b>When Set (1 = High = True), Enables:</b>
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 99
- "[\\*ESR \(Standard Event Status Register\)](#)" on page 104
- "[\\*OPC \(Operation Complete\)](#)" on page 108
- "[\\*CLS \(Clear Status\)](#)" on page 101

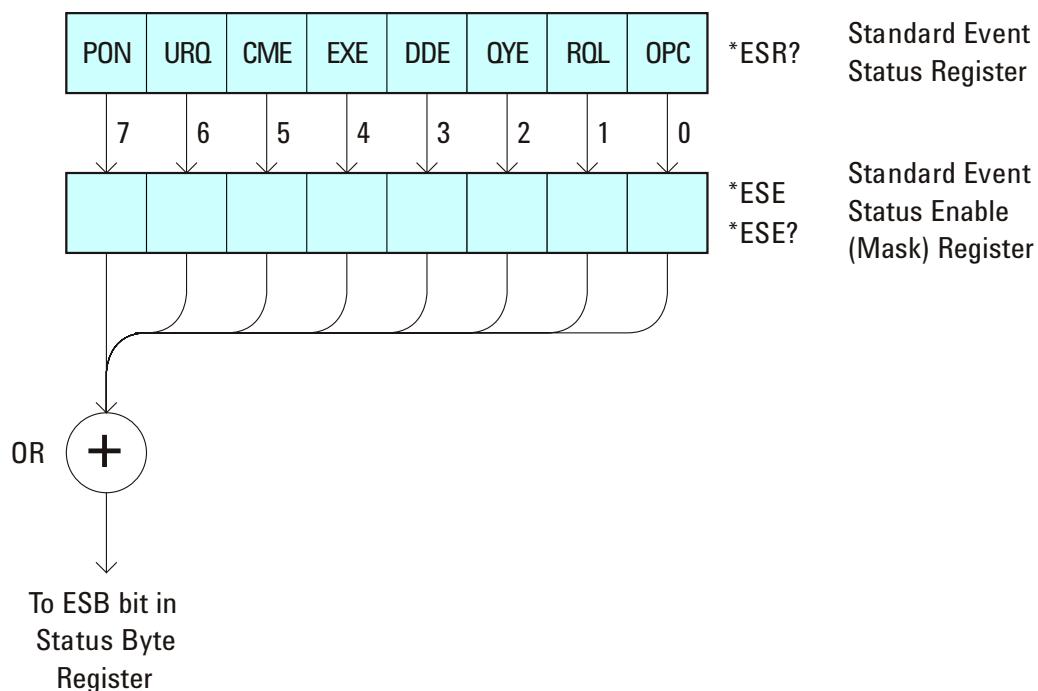
**\*ESR (Standard Event Status Register)**

**C** (see page 658)

**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 31** 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.

**Table 31** Standard Event Status Register (ESR) (continued)

<b>Bit</b>	<b>Name</b>	<b>Description</b>	<b>When Set (1 = High = True), Indicates:</b>
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 99
- "[\\*ESE \(Standard Event Status Enable\)](#)" on page 102
- "[\\*OPC \(Operation Complete\)](#)" on page 108
- "[\\*CLS \(Clear Status\)](#)" on page 101
- "[:SYSTem:ERRor](#)" on page 394

## **\*IDN (Identification Number)**



(see page 658)

**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 99
  - "\*OPT (Option Identification)" on page 109

## \*LRN (Learn Device Setup)



(see page 658)

### Query Syntax

\*LRN?

The \*LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see [page 397](#)) 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 99
- "["\\*RCL \(Recall\)](#)" on page 110
- "["\\*SAV \(Save\)](#)" on page 114
- "[":SYSTem:SETup](#)" on page 397

## **\*OPC (Operation Complete)**



(see page 658)

**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 99
  - "["\\*ESE \(Standard Event Status Enable\)](#)" on page 102
  - "["\\*ESR \(Standard Event Status Register\)](#)" on page 104
  - "["\\*CLS \(Clear Status\)](#)" on page 101

## **\*OPT (Option Identification)**

**C** (see page 658)

## Query Syntax \*OPT?

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

**Return Format** 0.0 <license info>

```
<license info> ::= <All field>, <reserved>, <reserved>, <reserved>,
                  <reserved>, <reserved>, <Low Speed Serial>,
                  <Automotive Serial>, <reserved>, <Secure>, <reserved>,
                  <reserved>, <reserved>, <reserved>,
                  <RS-232/UART Serial>, <reserved>, <Segmented Memory>,
                  <Mask Test>, <reserved>

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

<reserved> ::= 0

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

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

<Secure> ::= {0 | SEC}

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

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

<Mask Test> ::= {0 | LMT}
```

The \*OPT? query returns the following:

**See Also** • ["Introduction to Common \(\\*\) Commands" on page 99](#)  
• ["\\*IDN \(Identification Number\)" on page 106](#)

## **\*RCL (Recall)**



(see page 658)

**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 99
- "["\\*SAV \(Save\)](#)" on page 114

**\*RST (Reset)**

(see page 658)

**Command Syntax**

\*RST

The \*RST command places the instrument in a known state. Reset conditions are:

Acquire Menu	
Mode	Normal
Realtime	On
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

## 5 Commands by Subsystem

Display Menu	
Definite persistence	Off
Grid	33%
Vectors	On

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off
Holdoff	60 ns
External probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1

Trigger Menu	
External Units	Volts
External Impedance	1 M Ohm

- See Also** • ["Introduction to Common \(\\*\) Commands" on page 99](#)

**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.
```

Example program from the start: ["VISA COM Example in Visual Basic" on page 744](#)

### \*SAV (Save)



(see page 658)

#### 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 99
- "[\\*RCL \(Recall\)](#)" on page 110

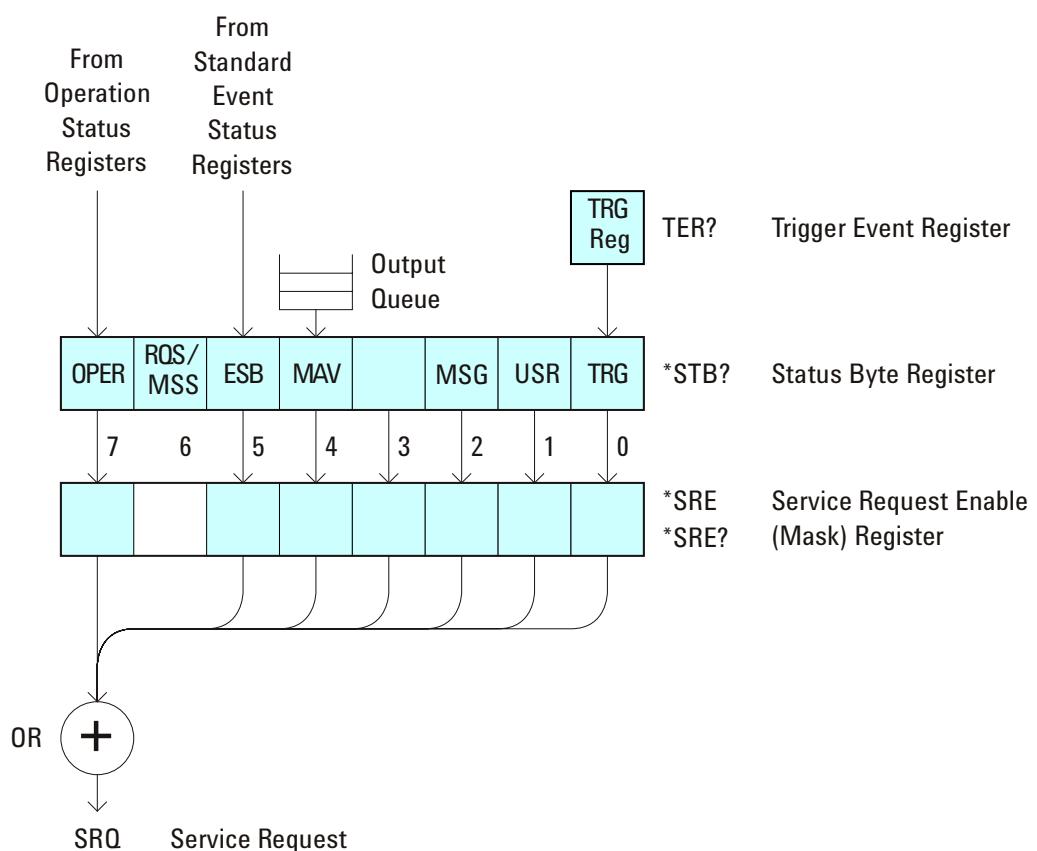
## \*SRE (Service Request Enable)

**C** (see page 658)

**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 32** 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.)

**Table 32** Service Request Enable Register (SRE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
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 99
  - "["\\*STB \(Read Status Byte\)"](#) on page 117
  - "["\\*CLS \(Clear Status\)"](#) on page 101

**\*STB (Read Status Byte)**

**C** (see page 658)

**Query Syntax**

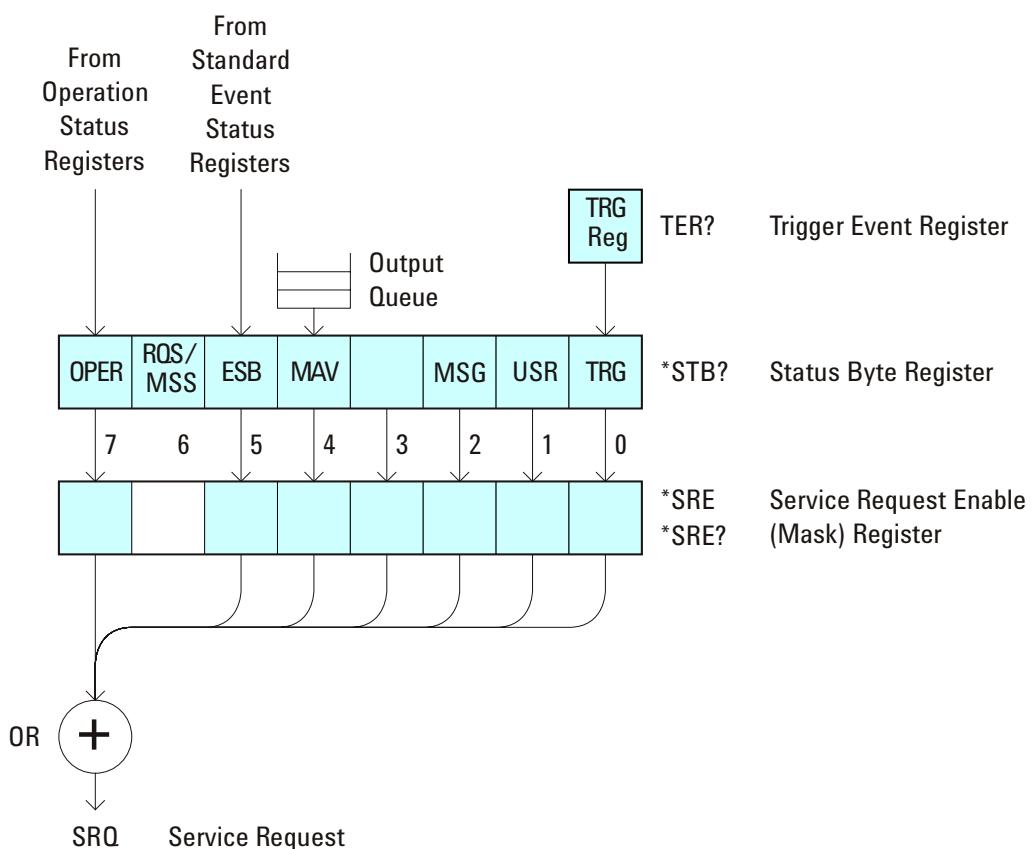
\*STB?

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 33** 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.

**Table 33** Status Byte Register (STB) (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
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 99
- "["\\*SRE \(Service Request Enable\)](#)" on page 115

**\*TRG (Trigger)** (see page 658)**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 99
- "[":DIGITIZE](#)" on page 132
- "[":RUN](#)" on page 156
- "[":STOP](#)" on page 160

## **\*TST (Self Test)**



(see page 658)

**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 99

**\*WAI (Wait To Continue)**

(see page 658)

**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 99

## 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 124.

**Table 34** Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see <a href="#">page 125</a> )	{0   1}; an integer in NR1 format
:AUToscale [ <source/> [,...,<source>]] (see <a href="#">page 126</a> )	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value> (see <a href="#">page 128</a> )	:AUToscale:AMODE? (see <a href="#">page 128</a> )	<value> ::= {NORMAL   CURRent}}
:AUToscale:CHANnels <value> (see <a href="#">page 129</a> )	:AUToscale:CHANnels? (see <a href="#">page 129</a> )	<value> ::= {ALL   DISPlayed}}
:BLANK [<source>] (see <a href="#">page 130</a> )	n/a	<source> ::= {CHANnel<n>}   FUNCTION   MATH <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see <a href="#">page 131</a> )	n/a	n/a
:DIGItize [ <source/> [,...,<source>]] (see <a href="#">page 132</a> )	n/a	<source> ::= {CHANnel<n>}   FUNCTION   MATH <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n> (see <a href="#">page 134</a> )	:HWEenable? (see <a href="#">page 134</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister:CONDition? (see <a href="#">page 136</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENT]? (see <a href="#">page 138</a> )	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see <a href="#">page 140</a> )	n/a	<pixel memory> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:MTEenable <n> (see <a href="#">page 141</a> )	:MTEenable? (see <a href="#">page 141</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:MTERegister[:EVENT]? (see <a href="#">page 143</a> )	<n> ::= 16-bit integer in NR1 format

**Table 34** Root (:) Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>																																	
:OPEE <n> (see page 145)	:OPEE? (see <a href="#">page 146</a> )	<n> ::= 16-bit integer in NR1 format																																	
n/a	:OPERegister:CONDition? (see <a href="#">page 147</a> )	<n> ::= 16-bit integer in NR1 format																																	
n/a	:OPERegister[:EVENT]? (see <a href="#">page 149</a> )	<n> ::= 16-bit integer in NR1 format																																	
:OVLenable <mask> (see <a href="#">page 151</a> )	:OVLenable? (see <a href="#">page 152</a> )	<mask> ::= 16-bit integer in NR1 format as shown: <table> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Input</th> </tr> </thead> <tbody> <tr><td>10</td><td>1024</td><td>Ext Trigger Fault</td></tr> <tr><td>9</td><td>512</td><td>Channel 4 Fault</td></tr> <tr><td>8</td><td>256</td><td>Channel 3 Fault</td></tr> <tr><td>7</td><td>128</td><td>Channel 2 Fault</td></tr> <tr><td>6</td><td>64</td><td>Channel 1 Fault</td></tr> <tr><td>4</td><td>16</td><td>Ext Trigger OVL</td></tr> <tr><td>3</td><td>8</td><td>Channel 4 OVL</td></tr> <tr><td>2</td><td>4</td><td>Channel 3 OVL</td></tr> <tr><td>1</td><td>2</td><td>Channel 2 OVL</td></tr> <tr><td>0</td><td>1</td><td>Channel 1 OVL</td></tr> </tbody> </table>	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
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 <a href="#">page 153</a> )	<value> ::= integer in NR1 format. See OVLenable for <value>																																	
:PRINT [<options>] (see <a href="#">page 155</a> )	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 <a href="#">page 156</a> )	n/a	n/a																																	
n/a	:SERial (see <a href="#">page 157</a> )	<return value> ::= unquoted string containing serial number																																	
:SINGle (see <a href="#">page 158</a> )	n/a	n/a																																	
n/a	:STATus? <display> (see <a href="#">page 159</a> )	{0   1} <display> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format																																	
:STOP (see <a href="#">page 160</a> )	n/a	n/a																																	

**Table 34** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see <a href="#">page 161</a> )	{0   1}
:VIEW <source> (see <a href="#">page 162</a> )	n/a	<source> ::= {CHANnel<n>   PMEMory{0   1   2   3   4   5   6   7   8   9}   FUNCtion   MATH} <n> ::= 1-2 or 1-4 in NR1 format

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

## :AER (Arm Event Register)



(see page 658)

### 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

&lt;value&gt;&lt;NL&gt;

&lt;value&gt; ::= {0 | 1}; an integer in NR1 format.

### See Also

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[":OPEE \(Operation Status Enable Register\)"](#) on page 145
- "[":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 147
- "[":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 149
- "[":\\*STB \(Read Status Byte\)"](#) on page 117
- "[":\\*SRE \(Service Request Enable\)"](#) on page 115

### :AUToscale

 (see page 658)

#### Command Syntax

```
:AUToscale  
:AUToscale [<source>[,...,<source>]]  
<source> ::= CHANnel<n>  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models  
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 Autoscale 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 129) is set to DISPlayed channels. Then, the autoscale is performed.

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

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

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

- Cursors.
- Measurements.
- Trace memories.
- Zoomed (delayed) time base mode.

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

#### See Also

- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
- "[:AUToscale:CHANnels](#)" on page 129

- [":AUToscale:AMODE"](#) on page 128

**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 Autoscale key.
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 744

### :AUToscale:AMODE

**N** (see page 658)

**Command Syntax** :AUToscale:AMODE <value>

<value> ::= {NORMAL | CURRent}

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

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

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

**Query Syntax** :AUToscale:AMODE?

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

**Return Format** <value><NL>

<value> ::= {NORM | CURR}

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 124
  - "[":AUToscale"](#) on page 126
  - "[":AUToscale:CHANnels"](#) on page 129
  - "[":ACQuire:TYPE"](#) on page 177
  - "[":ACQuire:MODE"](#) on page 169

## :AUToscale:CHANnels

**N** (see page 658)

**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 124
  - "[:AUToscale](#)" on page 126
  - "[:AUToscale:AMODE](#)" on page 128
  - "[:VIEW](#)" on page 162
  - "[:BLANK](#)" on page 130

### :BLANK

**N** (see page 658)

#### Command Syntax

```
:BLANK [<source>]  
<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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

#### NOTE

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

#### NOTE

MATH is an alias for FUNCtion.

#### See Also

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[:CDISplay](#)" on page 131
- "[:CHANnel<n>:DISPLAY](#)" on page 194
- "[:FUNCtion:DISPLAY](#)" on page 232
- "[:SBUS:DISPLAY](#)" on page 380
- "[:STATUS](#)" on page 159
- "[:VIEW](#)" on page 162

#### Example Code

- "[Example Code](#)" on page 162

**:CDISplay**

(see page 658)

**Command Syntax**`:CDISplay`

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all the data in active channels and functions is erased; however, new data is displayed on the next acquisition.

**See Also**

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[":DISPlay:CLEar](#)" on page 210

**:DIGItize**

(see page 658)

**Command Syntax**

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

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

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  

<n> ::= {1 | 2} for the two channel oscilloscope models  

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**

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

**NOTE**

MATH is an alias for FUNCtion.

**See Also**

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[":RUN"](#) on page 156
- "[":SINGle"](#) on page 158
- "[":STOP"](#) on page 160
- "[":ACQuire Commands](#)" on page 163
- "[":WAveform Commands](#)" on page 500

**Example Code**

```
' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface. Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.
'
' NOTE! The DIGITIZE command is highly recommended for triggering
' modes other than SINGLE. This ensures that sufficient data is
' available for measurement. If DIGITIZE is used with single mode,
' the completion criteria may never be met. The number of points
' gathered in Single mode is related to the sweep speed, memory
' depth, and maximum sample rate. For example, take an oscilloscope
' with a 1000-point memory, a sweep speed of 10 us/div (100 us
' total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s. Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
```

```
' because this is greater than the maximum sample rate by 5 times,  
' only 400 points (or 1/5 the points) can be gathered on a single  
' trigger. Keep in mind when the oscilloscope is running,  
' communication with the computer interrupts data acquisition.  
' Setting up the oscilloscope over the bus causes the data buffers  
' to be cleared and internal hardware to be reconfigured. If a  
' measurement is immediately requested, there may have not been  
' enough time for the data acquisition process to collect data, and  
' the results may not be accurate. An error value of 9.9E+37 may be  
' returned over the bus in this situation.  
  
myScope.WriteString ":DIGITIZE CHAN1"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

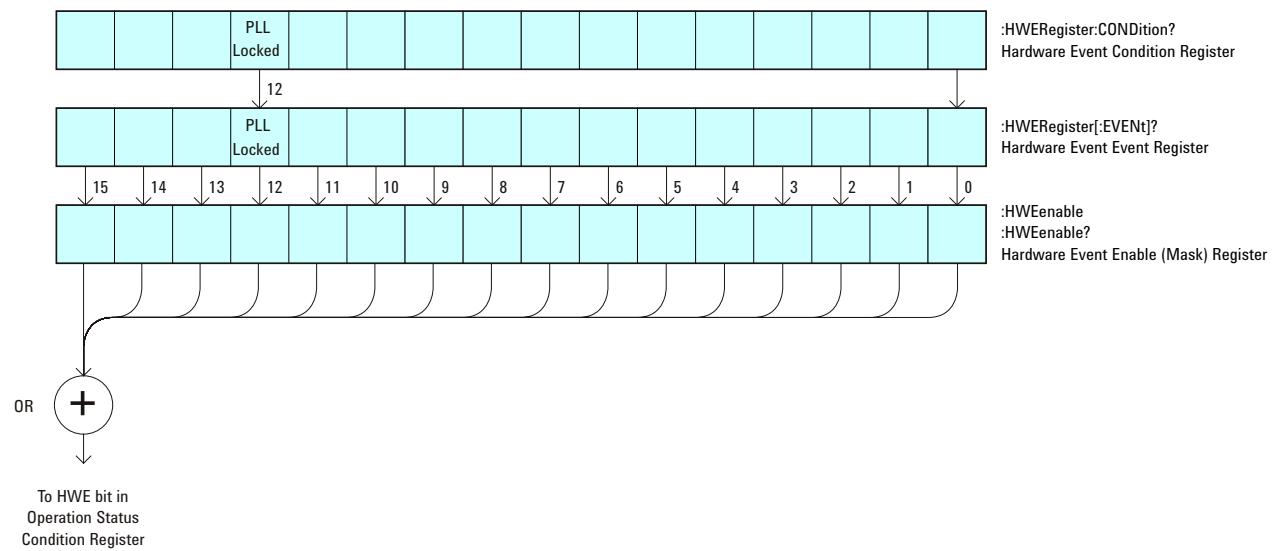
## :HWEenable (Hardware Event Enable Register)

**N** (see page 658)

**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 35** 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 124

- "[:AER \(Arm Event Register\)](#)" on page 125
- "[:CHANnel<n>:PROTection](#)" on page 203
- "[:EXTernal:PROTection](#)" on page 225
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 149
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[:\\*STB \(Read Status Byte\)](#)" on page 117
- "[:\\*SRE \(Service Request Enable\)](#)" on page 115

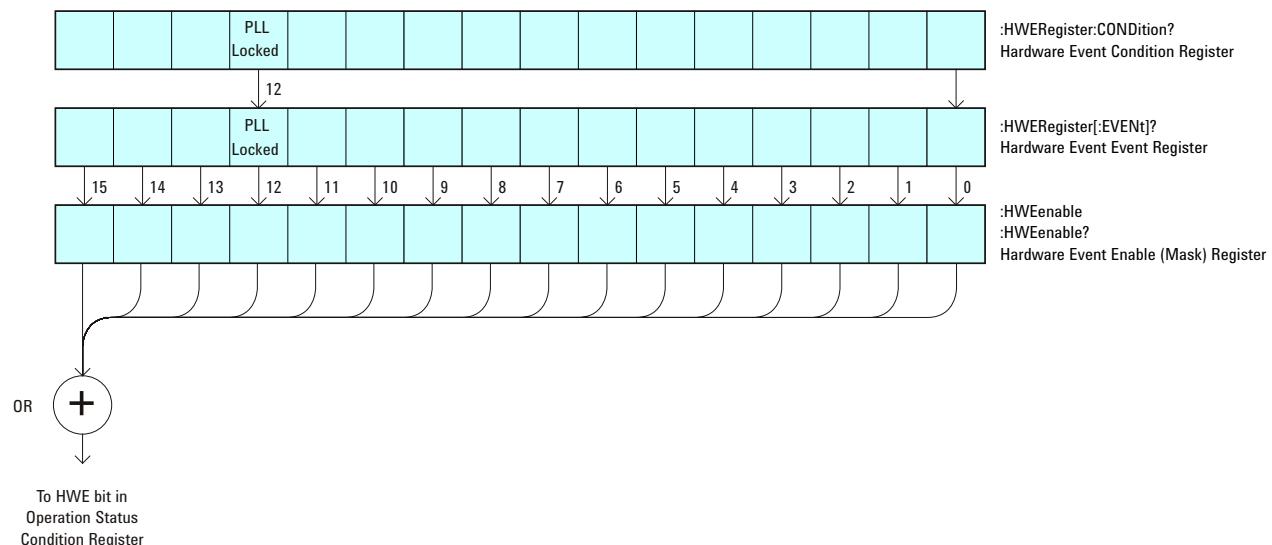
**:HWERegister:CONDition (Hardware Event Condition Register)**

**N** (see page 658)

**Query Syntax**

`:HWERegister:CONDition?`

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



**Table 36** 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 124
- "[":CHANnel<n>:PROTection](#)" on page 203
- "[":EXTernal:PROTection](#)" on page 225
- "[":OPEE \(Operation Status Enable Register\)"](#) on page 145
- "[":OPERegister\[:EVENTn\] \(Operation Status Event Register\)"](#) on page 149
- "[":OVLenable \(Overload Event Enable Register\)"](#) on page 151

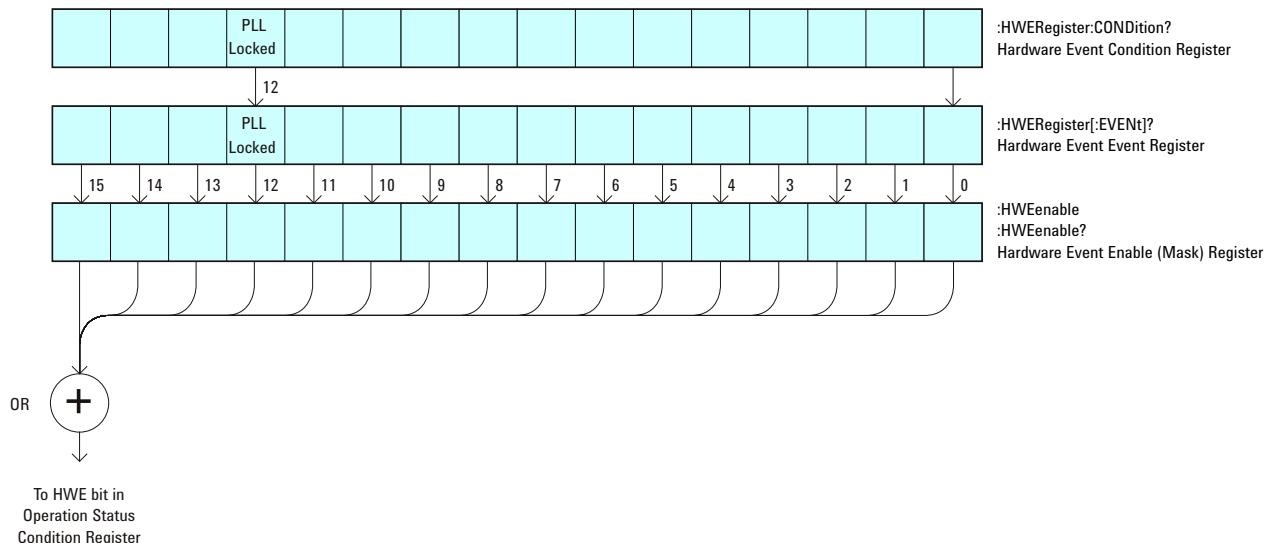
- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[\\*STB \(Read Status Byte\)](#)" on page 117
- "[\\*SRE \(Service Request Enable\)](#)" on page 115

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

**N** (see page 658)

**Query Syntax** :HWERegister[:EVENT]?

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



**Table 37** 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 124
- "[":CHANnel<n>:PROTection](#)" on page 203
- "[":EXTernal:PROTection](#)" on page 225
- "[":OPEE \(Operation Status Enable Register\)"](#) on page 145
- "[":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 147
- "[":OVLenable \(Overload Event Enable Register\)"](#) on page 151

- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[\\*STB \(Read Status Byte\)](#)" on page 117
- "[\\*SRE \(Service Request Enable\)](#)" on page 115

## :MERGe

**N** (see page 658)

**Command Syntax** :MERGe <pixel memory>

```
<pixel memory> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3  
| PMEMory4 | PMEMory5 | PMEMory6 | PMEMory7  
| PMEMory8 | PMEMory9}
```

The :MERGe command stores the contents of the active display in the specified pixel memory. The previous contents of the pixel memory are overwritten. The pixel memories are PMEMory0 through PMEMory9. This command is similar to the function of the "Save To: INTERN\_<n>" key in the Save/Recall menu.

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 124
  - "["\\*SAV \(Save\)"](#) on page 114
  - "["\\*RCL \(Recall\)"](#) on page 110
  - "[":VIEW"](#) on page 162
  - "[":BLANK"](#) on page 130

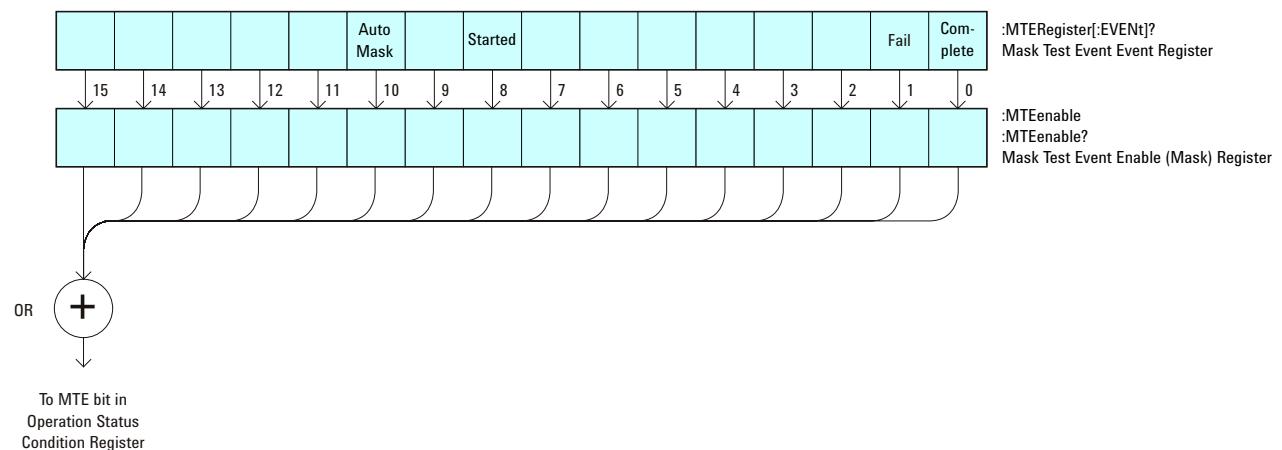
## :MTEenable (Mask Test Event Enable Register)

**N** (see page 658)

**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 38** 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.

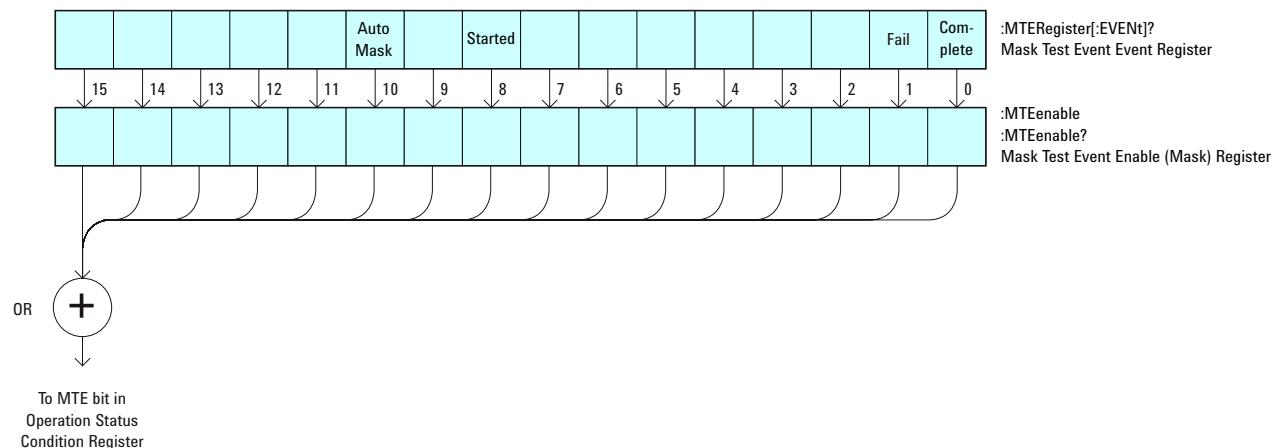
- See Also**
- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
  - "[:AER \(Arm Event Register\)](#)" on page 125
  - "[:CHANnel<n>:PROTection](#)" on page 203
  - "[:EXTernal:PROTection](#)" on page 225
  - "[:OPERegister\[:EVENT\]](#) ([Operation Status Event Register](#))" on page 149
  - "[:OVLenable](#) ([Overload Event Enable Register](#))" on page 151
  - "[:OVLRegister](#) ([Overload Event Register](#))" on page 153
  - "[:\\*STB \(Read Status Byte\)](#)" on page 117
  - "[:\\*SRE \(Service Request Enable\)](#)" on page 115

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

**N** (see page 658)

**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 39** 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 124

• ":CHANnel<n>:PROTection" on page 203

• ":EXTernal:PROTection" on page 225

## 5 Commands by Subsystem

- "[:OPEE \(Operation Status Enable Register\)](#)" on page 145
- "[:OPERegister:CONDITION \(Operation Status Condition Register\)](#)" on page 147
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[:\\*STB \(Read Status Byte\)](#)" on page 117
- "[:\\*SRE \(Service Request Enable\)](#)" on page 115

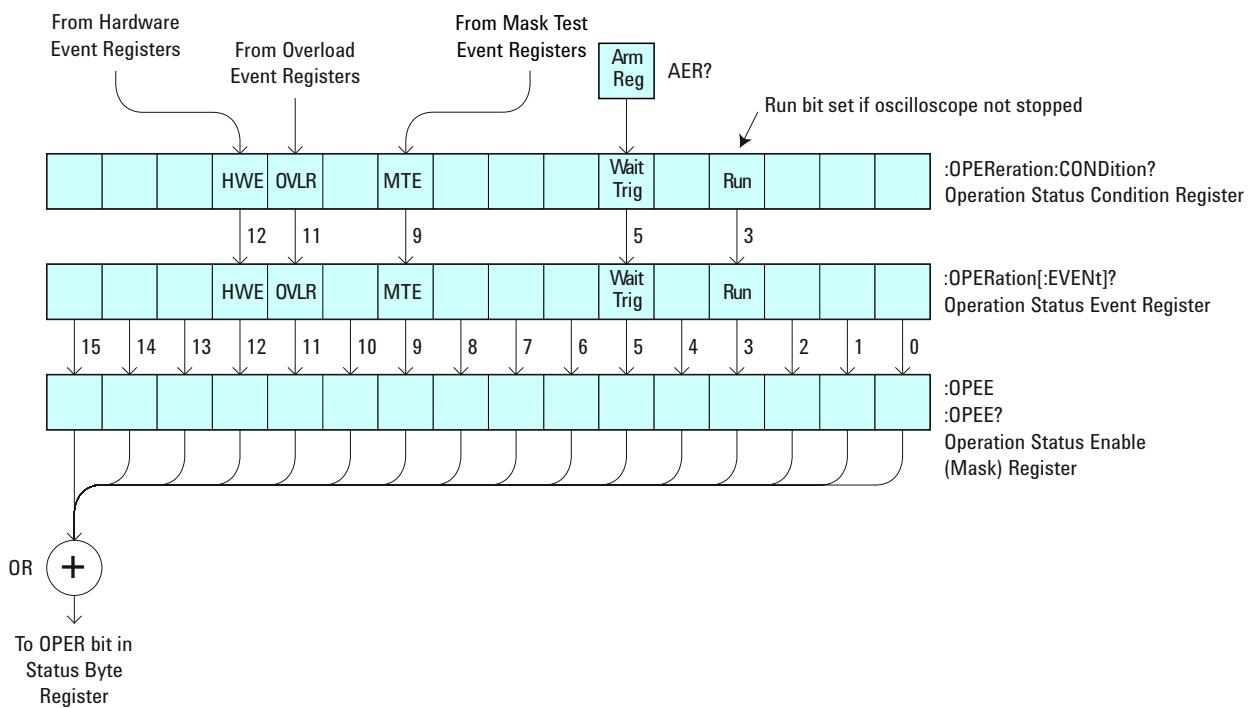
## :OPEE (Operation Status Enable Register)

**C** (see page 658)

**Command Syntax** :OPEE <mask>

<mask> ::= 16-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 40** Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
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.)

**Table 40** Operation Status Enable Register (OPEE) (continued)

<b>Bit</b>	<b>Name</b>	<b>Description</b>	<b>When Set (1 = High = True), Enables:</b>
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**

&lt;value&gt;&lt;NL&gt;

<value> ::= integer in NR1 format.

**See Also**

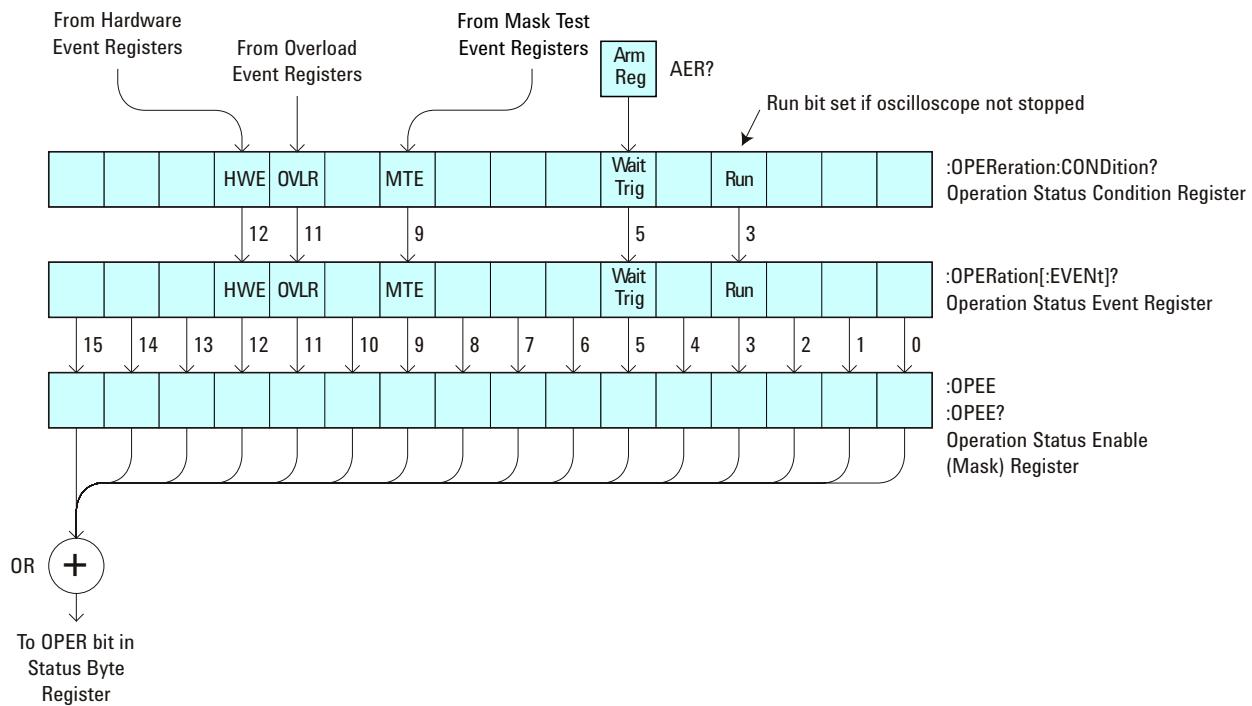
- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
- "[:AER \(Arm Event Register\)](#)" on page 125
- "[:CHANnel<n>:PROTection](#)" on page 203
- "[:EXTernal:PROTection](#)" on page 225
- "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 149
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[:\\*STB \(Read Status Byte\)](#)" on page 117
- "[:\\*SRE \(Service Request Enable\)](#)" on page 115

## :OPERegister:CONDition (Operation Status Condition Register)

**C** (see page 658)

**Query Syntax** :OPERegister:CONDition?

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



**Table 41** Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred..
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.)

**Table 41** Operation Status Condition Register (continued)

<b>Bit</b>	<b>Name</b>	<b>Description</b>	<b>When Set (1 = High = True), Indicates:</b>
3	Run	Running	The oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

**Return Format** <value><NL>

&lt;value&gt; ::= integer in NR1 format.

**See Also**

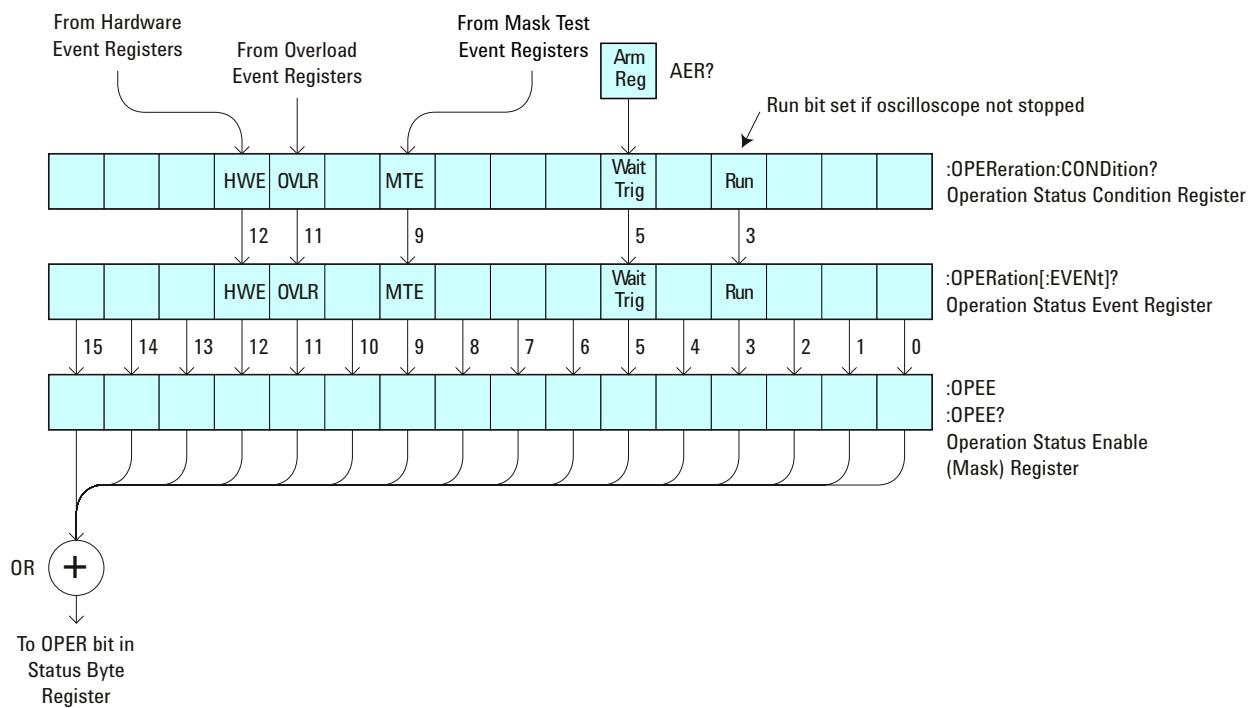
- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
- "[:CHANnel<n>:PROTection](#)" on page 203
- "[:EXTernal:PROTection](#)" on page 225
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 145
- "[:OPERegister\[:EVENT\]](#) ([Operation Status Event Register](#))" on page 149
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
- "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- "[\\*STB \(Read Status Byte\)](#)" on page 117
- "[\\*SRE \(Service Request Enable\)](#)" on page 115
- "[:HWERegister\[:EVENT\]](#) ([Hardware Event Event Register](#))" on page 138
- "[:HWEenable \(Hardware Event Enable Register\)](#)" on page 134
- "[:MTERegister\[:EVENT\]](#) ([Mask Test Event Event Register](#))" on page 143
- "[:MTEenable \(Mask Test Event Enable Register\)](#)" on page 141

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

**C** (see page 658)

**Query Syntax** :OPERegister [:EVENT]?

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



**Table 42** Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred.
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.)

**Table 42** Operation Status Event Register (continued)

<b>Bit</b>	<b>Name</b>	<b>Description</b>	<b>When Set (1 = High = True), Indicates:</b>
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 124
  - "[:CHANnel<n>:PROTection](#)" on page 203
  - "[:EXTernal:PROTection](#)" on page 225
  - "[:OPEE \(Operation Status Enable Register\)](#)" on page 145
  - "[:OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 147
  - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
  - "[:OVLRegister \(Overload Event Register\)](#)" on page 153
  - "[:\\*STB \(Read Status Byte\)](#)" on page 117
  - "[:\\*SRE \(Service Request Enable\)](#)" on page 115
  - "[:HWERegister\[:EVENT\]](#) [\(Hardware Event Event Register\)](#)" on page 138
  - "[:HWEenable \(Hardware Event Enable Register\)](#)" on page 134
  - "[:MTERegister\[:EVENT\]](#) [\(Mask Test Event Event Register\)](#)" on page 143
  - "[:MTEenable \(Mask Test Event Enable Register\)](#)" on page 141

## :OVLenable (Overload Event Enable Register)

**C** (see page 658)

**Command Syntax** :OVLenable <enable\_mask>

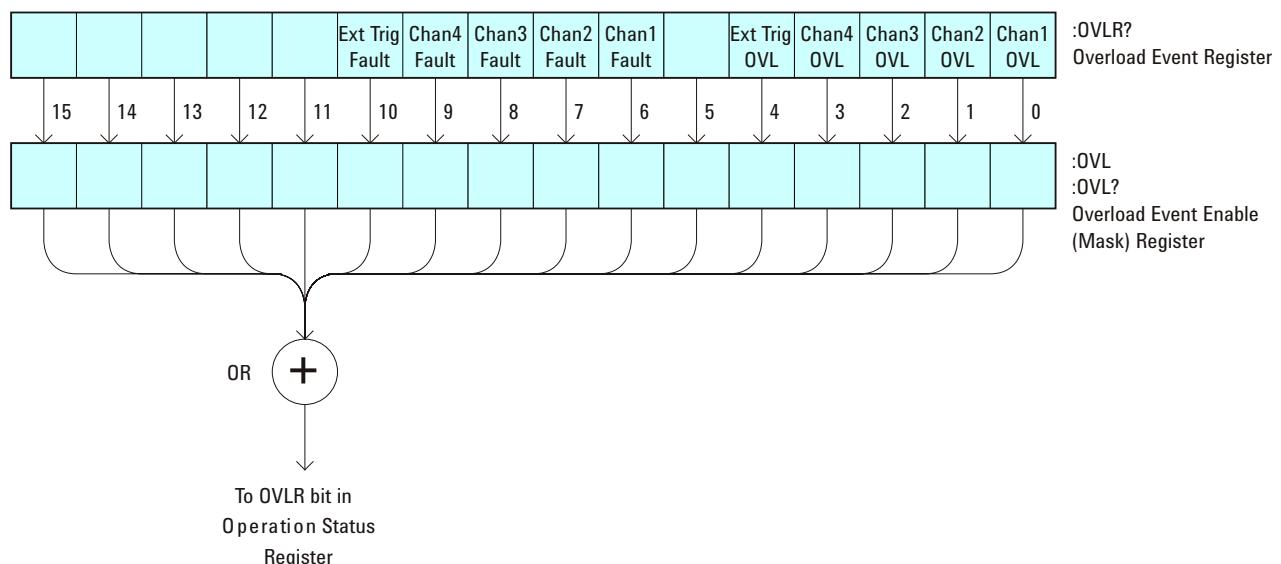
<enable\_mask> ::= 16-bit integer

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. If there are only two analog channels, you can also set external trigger input impedance to 50Ω.



**Table 43** Overload Event Enable Register (OVL)

Bit	Description	When Set (1 = High = True), Enables:
15-11	---	(Not used.)
10	External Trigger Fault	Event when fault occurs on External Trigger input.
9	Channel 4 Fault	Event when fault occurs on Channel 4 input.
8	Channel 3 Fault	Event when fault occurs on Channel 3 input.

**Table 43** Overload Event Enable Register (OVL) (continued)

<b>Bit</b>	<b>Description</b>	<b>When Set (1 = High = True), Enables:</b>
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	---	(Not used.)
4	External Trigger OVL	Event when overload occurs on External Trigger input.
3	Channel 4 OVL	Event when overload occurs on Channel 4 input.
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.
1	Channel 2 OVL	Event when overload occurs on Channel 2 input.
0	Channel 1 OVL	Event when overload occurs on Channel 1 input.

**Query Syntax** :OVLenable?

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

**Return Format** <enable\_mask><NL>

<enable\_mask> ::= integer in NR1 format.

- See Also**
- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
  - "[:CHANnel<n>:PROTection](#)" on page 203
  - "[:EXTernal:PROTection](#)" on page 225
  - "[:OPEE \(Operation Status Enable Register\)](#)" on page 145
  - "[:OPERegister:CONDition \(Operation Status Condition Register\)](#)" on page 147
  - "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 149
  - "[:OVLRegister \(Overload Event Register\)](#)" on page 153
  - "[\\*STB \(Read Status Byte\)](#)" on page 117
  - "[\\*SRE \(Service Request Enable\)](#)" on page 115

## :OVLRegister (Overload Event Register)

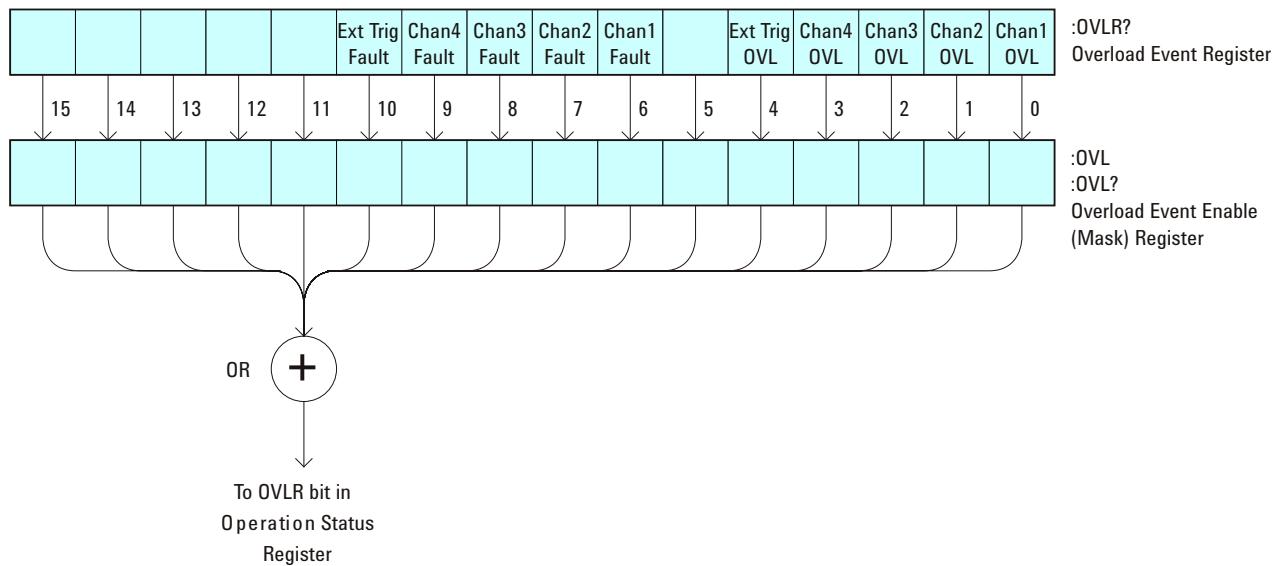
**C** (see page 658)

**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  $50\Omega$  input, the input will automatically switch to  $1 M\Omega$  input impedance. A "1" indicates an overload has occurred.

**NOTE**

You can set analog channel input impedance to  $50\Omega$ . If there are only two analog channels, you can also set external trigger input impedance to  $50\Omega$ .



**Table 44** Overload Event Register (OCLR)

Bit	Description	When Set (1 = High = True), Indicates:
15-11	---	(Not used.)
10	External Trigger Fault	Fault has occurred on External Trigger input.
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	---	(Not used.)

**Table 44** Overload Event Register (OVLR) (continued)

Bit	Description	When Set (1 = High = True), Indicates:
4	External Trigger OVL	Overload has occurred on External Trigger input.
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 124
- "[:CHANnel<n>:PROTection](#)" on page 203
- "[:EXTernal:PROTection](#)" on page 225
- "[:OPEE \(Operation Status Enable Register\)](#)" on page 145
- "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
- "[:\\*STB \(Read Status Byte\)](#)" on page 117
- "[:\\*SRE \(Service Request Enable\)](#)" on page 115

**:PRINt**

 (see page 658)

**Command Syntax**

```
:PRINt [<options>]
<options> ::= [<print option>][,...<print option>]
<print option> ::= {COLOR | GRAYscale | PRINTER0 | BMP8bit | BMP | PNG
                     | NOFactors | FACTors}
```

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

The PRINt command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used. Refer to "[:HARDcopy:FORMAT](#)" on page 581 for more information.

**See Also**

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[Introduction to :HARDcopy Commands](#)" on page 246
- "[:HARDcopy:FORMAT](#)" on page 581
- "[:HARDcopy:FACTors](#)" on page 249
- "[:HARDcopy:GRAYscale](#)" on page 582
- "[:DISPlay:DATA](#)" on page 211

### :RUN



(see page 658)

#### 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 124
- "[":SINGle"](#) on page 158
- "[":STOP"](#) on page 160

#### 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.
```

Example program from the start: "["VISA COM Example in Visual Basic"](#) on page 744

**:SERial**

**N** (see page 658)

**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 124

## **:SINGle**



(see page 658)

**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 124
- "[":RUN"](#) on page 156
- "[":STOP"](#) on page 160

**:STATus**

**N** (see page 658)

**Query Syntax**

```
:STATus? <source>
<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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

**NOTE**

MATH is an alias for FUNCtion.

**Return Format**

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

**See Also**

- "[Introduction to Root \(:\) Commands](#)" on page 124
- "[:BLANK](#)" on page 130
- "[:CHANnel<n>:DISPlay](#)" on page 194
- "[:FUNCtion:DISPlay](#)" on page 232
- "[:SBUS:DISPlay](#)" on page 380
- "[:VIEW](#)" on page 162

### :STOP



(see page 658)

#### 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 124
- "[":RUN"](#) on page 156
- "[":SINGLe"](#) on page 158

#### Example Code

- "["Example Code"](#) on page 156

## :TER (Trigger Event Register)



(see page 658)

**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 124
- "[\\*SRE \(Service Request Enable\)](#)" on page 115
- "[\\*STB \(Read Status Byte\)](#)" on page 117

### :VIEW

**N** (see page 658)

#### Command Syntax

```
:VIEW <source>
<source> ::= {CHANnel<n> | PMEMory0,...,PMEMory9 | FUNCtion | MATH
              | SBUS}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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

#### NOTE

MATH is an alias for FUNCtion.

#### See Also

- "[Introduction to Root \(:\)](#) [Commands](#)" on page 124
- "[:BLANK](#)" on page 130
- "[:CHANnel<n>:DISPlay](#)" on page 194
- "[:FUNCtion:DISPlay](#)" on page 232
- "[:SBUS:DISPlay](#)" on page 380
- "[:STATus](#)" on page 159

#### Example Code

```
' VIEW_BLANK - (not executed in this example)
'   - VIEW turns on (starts displaying) a channel or pixel memory.
'   - BLANK turns off (stops displaying) a channel or pixel memory.
' myScope.WriteString ":BLANK CHANNEL1"      ' Turn channel 1 off.
' myScope.WriteString ":VIEW CHANNEL1"        ' Turn channel 1 on.
```

Example program from the start: "["VISA COM Example in Visual Basic"](#) on page 744

## :ACQuire Commands

Set the parameters for acquiring and storing data. See "[Introduction to :ACQuire Commands](#)" on page 163.

**Table 45** :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see <a href="#">page 165</a> )	{1   0}
:ACQuire:COMPLETE <complete> (see <a href="#">page 166</a> )	:ACQuire:COMPLETE? (see <a href="#">page 166</a> )	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNT <count> (see <a href="#">page 167</a> )	:ACQuire:COUNT? (see <a href="#">page 167</a> )	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALIAS <mode> (see <a href="#">page 168</a> )	:ACQuire:DAALIAS? (see <a href="#">page 168</a> )	<mode> ::= {DISable   AUTO}
:ACQuire:MODE <mode> (see <a href="#">page 169</a> )	:ACQuire:MODE? (see <a href="#">page 169</a> )	<mode> ::= {RTIMe   ETIMe   SEGmented}
n/a	:ACQuire:POINTS? (see <a href="#">page 170</a> )	<# points> ::= an integer in NR1 format
:ACQuire:SEGmented:AN ALyze (see <a href="#">page 171</a> )	n/a	n/a (with Option SGM)
:ACQuire:SEGmented:COUNT <count> (see <a href="#">page 172</a> )	:ACQuire:SEGmented:COUNT? (see <a href="#">page 172</a> )	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
:ACQuire:SEGmented:INDEX <index> (see <a href="#">page 173</a> )	:ACQuire:SEGmented:INDEX? (see <a href="#">page 173</a> )	<index> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATE? (see <a href="#">page 176</a> )	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see <a href="#">page 177</a> )	:ACQuire:TYPE? (see <a href="#">page 177</a> )	<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. Two acquisition modes are available: real-time mode, and equivalent-time mode.

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.

### Real-time Mode

The :ACQuire:MODE RTIMe command sets the oscilloscope in real-time mode. This mode is useful to inhibit equivalent time sampling at fast sweep speeds.

### Equivalent-time Mode

The :ACQuire:MODE ETIME command sets the oscilloscope in equivalent-time mode.

### Reporting the Setup

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

### Return Format

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

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

## :ACQuire:AALias

**N** (see page 658)

**Query Syntax** :ACQuire:AALias?

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

**Return Format** <value><NL>

<value> ::= {1 | 0}

**See Also** • "Introduction to :ACQuire Commands" on page 163  
• ":ACQuire:DAALias" on page 168

### :ACQuire:COMplete



(see page 658)

#### **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 163
- "[":ACQuire:TYPE](#)" on page 177
- "[":DIGItize](#)" on page 132
- "[":WAVeform:POINTs](#)" on page 512

#### **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"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

**:ACQuire:COUNt**

(see page 658)

**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 163
- "[":ACQuire:TYPE"](#) on page 177
- "[":DIGitize"](#) on page 132
- "[":WAVeform:COUNt"](#) on page 508

## :ACQuire:DAALias

**N** (see page 658)

**Command Syntax** :ACQuire:DAALias <mode>  
<mode> ::= {DISable | AUTO}

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

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

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

**Query Syntax** :ACQuire:DAALias?

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

**Return Format** <mode><NL>  
<mode> ::= {DIS | AUTO}

**See Also** • "Introduction to :ACQuire Commands" on page 163  
• ":ACQuire:AALias" on page 165

**:ACQuire:MODE**

(see page 658)

**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. This mode is useful to inhibit equivalent time sampling at fast sweep speeds.

Real time mode is not available when averaging (:ACQuire:TYPE AVERage).

**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.
  - 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 163
- "[":ACQuire:TYPE"](#) on page 177

## **:ACQuire:POINts**



(see page 658)

### **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 163
- "[":DIGItize"](#)" on page 132
- "[":WAVeform:POINts"](#)" on page 512

**:ACQuire:SEGmented:ANALyze****N** (see page 658)**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 169
- "[:ACQuire:SEGmented:COUNt](#)" on page 172
- "[Introduction to :ACQuire Commands](#)" on page 163

## :ACQuire:SEGmented:COUNt

**N** (see page 658)

**Command Syntax** :ACQuire:SEGmented:COUNt <count>

<count> ::= an integer from 2 to 250 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.

**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 250 in NR1 format

**See Also**

- "[:ACQuire:MODE](#)" on page 169
- "[:DIGITIZE](#)" on page 132
- "[:SINGLE](#)" on page 158
- "[:RUN](#)" on page 156
- "[:WAVeform:SEGmented:COUNt](#)" on page 519
- "[:ACQuire:SEGmented:ANALyze](#)" on page 171
- "[Introduction to :ACQuire Commands](#)" on page 163

**Example Code** • "[Example Code](#)" on page 173

## :ACQuire:SEGmented:INDex

**N** (see page 658)

**Command Syntax** :ACQuire:SEGmented:INDex <index>

<index> ::= an integer from 2 to 250 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.

**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 2 to 250 in NR1 format

**See Also**

- "[:ACQuire:MODE](#)" on page 169
- "[:ACQuire:SEGmented:COUNt](#)" on page 172
- "[:DIGITIZE](#)" on page 132
- "[:SINGLE](#)" on page 158
- "[:RUN](#)" on page 156
- "[:WAveform:SEGmented:COUNt](#)" on page 519
- "[:WAveform:SEGmented:TTAG](#)" on page 520
- "[:ACQuire:SEGmented:ANALyze](#)" on page 171
- "[Introduction to :ACQuire Commands](#)" on page 163

**Example Code**

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

```
' -----
```

Option Explicit

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

## 5 Commands by Subsystem

```
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")
    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 50.
    myScope.WriteString ":ACQuire:SEGmented:COUNT 50"
    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 = 50

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

    Dim lngSegments As Long
    lngSegments = varQueryResult

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

    For lngI = lngSegments To 1 Step -1

        ' Set the segmented memory index.
        myScope.WriteString ":ACQuire:SEGmented:INDEX " + CStr(lngI)
```

```
myScope.WriteString ":ACQuire:SEGmented:INDEX?"
strQueryResult = myScope.ReadString
Debug.Print "Acquisition memory segment index: " + strQueryResult

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

Next lngI

Exit Sub

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

End Sub
```

## **:ACQuire:SRATe**

**N** (see page 658)

**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 163  
• "[:ACQuire:POINTs](#)" on page 170

**:ACQuire:TYPE**

(see page 658)

**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, AVERage, HRESolution, and PEAK.

- The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal mode.
- 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 1 to 65536. The COUNt value determines the number of averages that must be acquired.

Setting the :ACQuire:TYPE to AVERage automatically sets :ACQuire:MODE to ETIMe (equivalent time sampling).

The AVERage type is not available when in segmented memory mode (:ACQuire:MODE SEGmented).

- 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.

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.

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

**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 163
  - "[:ACQuire:COUNt](#)" on page 167
  - "[:ACQuire:MODE](#)" on page 169
  - "[:DIGitize](#)" on page 132
  - "[:WAVeform:TYPE](#)" on page 526
  - "[:WAVeform:PREamble](#)" on page 516

**Example Code**

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

Example program from the start: "["VISA COM Example in Visual Basic"](#)" on page 744

## :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 179.

**Table 46** :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see <a href="#">page 181</a> )	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABEL <string> (see <a href="#">page 182</a> )	:CALibrate:LABEL? (see <a href="#">page 182</a> )	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see <a href="#">page 183</a> )	:CALibrate:OUTPut? (see <a href="#">page 183</a> )	<signal> ::= {TRIGgers   SOURce   DSOurce   MASK}
:CALibrate:START (see <a href="#">page 184</a> )	n/a	n/a
n/a	:CALibrate:STATUS? (see <a href="#">page 185</a> )	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITch? (see <a href="#">page 186</a> )	{PROTected   UNPROtected}
n/a	:CALibrate:TEMPeratur e? (see <a href="#">page 187</a> )	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see <a href="#">page 188</a> )	<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.

## **5 Commands by Subsystem**

- Starting the user calibration procedure.

**:CALibrate:DATE**

**N** (see page 658)

**Query Syntax** :CALibrate:DATE?

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

**Return Format** <date><NL>

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

**See Also** • "Introduction to :CALibrate Commands" on page 179

### :CALibrate:LABel

**N** (see page 658)

**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 179

## :CALibrate:OUTPut

**N** (see page 658)

**Command Syntax** :CALibrate:OUTPut <signal>

```
<signal> ::= {TRIGgers | SOURce | DSOurce | MASK}
```

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

- TRIGgers – pulse when a trigger event occurs.
- SOURce – raw output of trigger comparator.
- DSOurce – SOURce frequency divided by 8.
- MASK – signal from mask test indicating a success or fail mask test.

**Query Syntax** :CALibrate:OUTPut?

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

**Return Format** <signal><NL>

```
<signal> ::= {TRIG | SOUR | DSO | MASK}
```

- See Also**
- "Introduction to :CALibrate Commands" on page 179
  - ":MTESt:OUTPut" on page 336

## **:CALibrate:STARt**

**N** (see page 658)

**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 179
- ":CALibrate:SWITch" on page 186

**:CALibrate:STATus**

**N** (see page 658)

**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> ::= ALL,<status_code>,<status_string>
<status_code> ::= an integer status code
<status_string> ::= an ASCII status string
```

**See Also** • "Introduction to :CALibrate Commands" on page 179

## **:CALibrate:SWITch**

**N** (see page 658)

**Query Syntax** :CALibrate:SWITch?

The :CALibrate:SWITch? query returns the rear-panel calibration protect (CAL PROTECT) switch 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 179

**:CALibrate:TEMPerature**

**N** (see page 658)

**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 179

## **:CALibrate:TIME**

**N** (see page 658)

**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 179

## :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "[Introduction to :CHANnel<n> Commands](#)" on page 190.

**Table 47** :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0   OFF}   {1   ON}} (see <a href="#">page 192</a> )	:CHANnel<n>:BWLimit? (see <a href="#">page 192</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see <a href="#">page 193</a> )	:CHANnel<n>:COUPling? (see <a href="#">page 193</a> )	<coupling> ::= {AC   DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 194</a> )	:CHANnel<n>:DISPlay? (see <a href="#">page 194</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see <a href="#">page 195</a> )	:CHANnel<n>:IMPedance? (see <a href="#">page 195</a> )	<impedance> ::= {ONEMeg   FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{0   OFF}   {1   ON}} (see <a href="#">page 196</a> )	:CHANnel<n>:INVert? (see <a href="#">page 196</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABEL <string> (see <a href="#">page 197</a> )	:CHANnel<n>:LABEL? (see <a href="#">page 197</a> )	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see <a href="#">page 198</a> )	:CHANnel<n>:OFFSet? (see <a href="#">page 198</a> )	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see <a href="#">page 199</a> )	:CHANnel<n>:PROBe? (see <a href="#">page 199</a> )	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see <a href="#">page 200</a> )	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see <a href="#">page 201</a> )	:CHANnel<n>:PROBe:SKEW? (see <a href="#">page 201</a> )	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format

**Table 47** :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:STY Pe <signal type> (see page 202)	:CHANnel<n>:PROBe:STY Pe? (see page 202)	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTectio n (see page 203)	:CHANnel<n>:PROTectio n? (see page 203)	{NORM   TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGE <range>[suffix] (see page 204)	:CHANnel<n>:RANGE? (see page 204)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALE <scale>[suffix] (see page 205)	:CHANnel<n>:SCALE? (see page 205)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITS <units> (see page 206)	:CHANnel<n>:UNITs? (see page 206)	<units> ::= {VOLT   AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {0   OFF}   {1   ON}} (see page 207)	:CHANnel<n>:VERNier? (see page 207)	{0   1} <n> ::= 1-2 or 1-4 in NR1 format

**Introduction to  
:CHANnel<n>  
Commands**

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 658)

**Command Syntax** :CHANnel<n>:BWLimit <bwlimit>

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

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 190

**:CHANnel<n>:COUPLing**

(see page 658)

**Command Syntax** :CHANnel<n>:COUPLing <coupling>

&lt;coupling&gt; ::= {AC | DC}

&lt;n&gt; ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

&lt;n&gt; ::= {1 | 2} for the two channel oscilloscope models

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>

&lt;coupling value&gt; ::= {AC | DC}

**See Also** • "Introduction to :CHANnel<n> Commands" on page 190

### :CHANnel<n>:DISPlay



(see page 658)

**Command Syntax** :CHANnel<n>:DISPlay <display value>  
<display value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

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 190
- "[":VIEW"](#) on page 162
- "[":BLANK"](#) on page 130
- "[":STATus"](#) on page 159

**:CHANnel<n>:IMPedance**

(see page 658)

**Command Syntax** :CHANnel<n>:IMPedance <impedance>

&lt;impedance&gt; ::= {ONEMeg | FIFTy}

&lt;n&gt; ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

&lt;n&gt; ::= {1 | 2} for the two channel oscilloscope models

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>

&lt;impedance value&gt; ::= {ONEM | FIFT}

**See Also** • "Introduction to :CHANnel<n> Commands" on page 190

### :CHANnel<n>:INVert

**N** (see page 658)

**Command Syntax** :CHANnel<n>:INVert <invert value>  
<invert value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

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 190

**:CHANnel<n>:LABel**

**N** (see page 658)

**Command Syntax**

```
:CHANnel<n>:LABel <string>
<string> ::= quoted ASCII string
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope 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. 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 190
- "[:DISPlay:LABel](#)" on page 213
- "[:DISPlay:LABList](#)" on page 214

**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".
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

### :CHANnel<n>:OFFSet



(see page 658)

#### Command Syntax

```
:CHANnel<n>:OFFSet <offset> [<suffix>]  
<offset> ::= Vertical offset value in NR3 format  
<suffix> ::= {V | mV}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 190
- "[":CHANnel<n>:RANGE](#)" on page 204
- "[":CHANnel<n>:SCALE](#)" on page 205
- "[":CHANnel<n>:PROBe](#)" on page 199

**:CHANnel<n>:PROBe**

(see page 658)

**Command Syntax** :CHANnel<n>:PROBe <attenuation>

&lt;attenuation&gt; ::= probe attenuation ratio in NR3 format

&lt;n&gt; ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

&lt;n&gt; ::= {1 | 2} for the two channel oscilloscope models

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>

&lt;attenuation&gt; ::= probe attenuation ratio in NR3 format

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 190
  - "[":CHANnel<n>:RANGE](#)" on page 204
  - "[":CHANnel<n>:SCALE](#)" on page 205
  - "[":CHANnel<n>:OFFSet](#)" on page 198

**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 ":CHAN1:PROBE 10" ' Set Probe to 10:1.
```

Example program from the start: "["VISA COM Example in Visual Basic"](#) on page 744

### :CHANnel<n>:PROBe:ID



(see page 658)

**Query Syntax** :CHANnel<n>:PROBe:ID?

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 190

**:CHANnel<n>:PROBe:SKEW**

(see page 658)

**Command Syntax** :CHANnel<n>:PROBe:SKEW <skew value>  
 <skew value> ::= skew time in NR3 format  
 <skew value> ::= -100 ns to +100 ns  
 <n> ::= {1 | 2 | 3 | 4}

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 190

## **:CHANnel<n>:PROBe:STYPe**



(see page 658)

### **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 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 190
- "[":CHANnel<n>:OFFSet](#)" on page 198

**:CHANnel<n>:PROTection**

**N** (see page 658)

**Command Syntax** :CHANnel<n>:PROTection[:CLEAR]  
 <n> ::= {1 | 2 | 3 | 4}

When the analog channel input impedance is set to  $50\Omega$ , 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  $50\Omega$  mode after the signal that caused the overload has been removed from the channel input. Reset the analog channel input impedance to  $50\Omega$  (see "[:CHANnel<n>:IMPedance](#)" on page 195) 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 190
- "[:CHANnel<n>:COUpling](#)" on page 193
- "[:CHANnel<n>:IMPedance](#)" on page 195
- "[:CHANnel<n>:PROBe](#)" on page 199

### :CHANnel<n>:RANGE



(see page 658)

#### Command Syntax

```
:CHANnel<n>:RANGE <range>[<suffix>]  
<range> ::= vertical full-scale range value in NR3 format  
<suffix> ::= {V | mV}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:RANGE command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, the range can be set to any value from:

- 16 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 190
- "[:CHANnel<n>:SCALE](#)" on page 205
- "[:CHANnel<n>:PROBe](#)" on page 199

#### 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.
```

Example program from the start: "["VISA COM Example in Visual Basic"](#)" on page 744

**:CHANnel<n>:SCALe**

**N** (see page 658)

**Command Syntax** :CHANnel<n>:SCALe <scale>[<suffix>]

<scale> ::= vertical units per division in NR3 format

<suffix> ::= {V | mV}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel. When using 1:1 probe attenuation, legal values for the scale range from:

- 2 mV to 5 V.

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 190
- ":CHANnel<n>:RANGE" on page 204
- ":CHANnel<n>:PROBe" on page 199

### :CHANnel<n>:UNITS

**N** (see page 658)

**Command Syntax** :CHANnel<n>:UNITS <units>

<units> ::= {VOLT | AMPere}

<n> ::= {1 | 2} for the two channel oscilloscope models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

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 190
  - "[:CHANnel<n>:RANGE](#)" on page 204
  - "[:CHANnel<n>:PROBe](#)" on page 199
  - "[:EXTernal:UNITS](#)" on page 227

**:CHANnel<n>:VERNier****N** (see page 658)

**Command Syntax** :CHANnel<n>:VERNier <vernier value>  
<vernier value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

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 190

## :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 208.

**Table 48** :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLEar (see page 210)	n/a	n/a
:DISPlay:DATA [ <format>] [,] [&lt;area&gt;] [,] [&lt;palette&gt;] &lt;display data&gt; (see page 211)</format>	:DISPlay:DATA? [<format>] [,] [<area>] [,] [<palette>] (see page 211)	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command)  <format> ::= {TIFF   BMP   BMP8bit   PNG} (query) <area> ::= {GRATicule   SCReen} (query) <palette> ::= {MONochrome   GRAYscale   COLOR} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0   OFF}   {1   ON}} (see page 213)	:DISPlay:LABel? (see page 213)	{0   1}
:DISPlay:LABList <binary block> (see page 214)	:DISPlay:LABList? (see page 214)	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERSistence <value> (see page 215)	:DISPlay:PERSistence? (see page 215)	<value> ::= {MINimum   INFinite}
:DISPlay:SOURce <value> (see page 216)	:DISPlay:SOURce? (see page 216)	<value> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:DISPlay:VECTors {{1   ON}   {0   OFF}} (see page 217)	:DISPlay:VECTors? (see page 217)	{1   0}

**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:CONN 1;PERS MIN;SOUR PMEM1
```

## **:DISPlay:CLEar**

**N** (see page 658)

**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 208
  - "[:CDISplay](#)" on page 131

## :DISPlay:DATA

**N** (see page 658)

**Command Syntax**

```
:DISPlay:DATA [<format>][,] [<area>][,] [<palette>]<display data>
<format> ::= {TIFF}
<area> ::= {GRATICule}
<palette> ::= {MONochrome}
<display data> ::= binary block data in IEEE-488.2 # format.
```

The :DISPlay:DATA command writes trace memory data (a display bitmap) to the display or to one of the trace memories in the instrument.

If a data format or area is specified, the :DISPlay:DATA command transfers the data directly to the display. If neither the data format nor the area is specified, the command transfers data to the trace memory specified by the :DISPlay:SOURce command. Available trace memories are PMEMory0-9 and these memories correspond to the INTERN\_0-9 files in the front panel Save/Recall menu.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. This is the same data saved using the front panel Save/Recall menu or the \*SAV (Save) command.

**Query Syntax**

```
:DISPlay:DATA? [<format>][,] [<area>][,] [<palette>]
<format> ::= {TIFF | BMP | BMP8bit | PNG}
<area> ::= {GRATICule | SCReen}
<palette> ::= {MONochrome | GRAYscale | COLOR}
```

The :DISPlay:DATA? query reads display data from the screen or from one of the trace memories in the instrument. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

If a data format or area is specified, the :DISPlay:DATA query transfers the data directly from the display. If neither the data format nor the area is specified, the query transfers data from the trace memory specified by the :DISPlay:SOURce command.

Screen data is the full display and is high resolution in grayscale or color. The :HARDcopy:INKSaver setting also affects the screen data. It may be read from the instrument in 24-bit bmp, 8-bit bmp, or 24-bit png format. This data cannot be sent back to the instrument.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. You can get this data and send it back to the oscilloscope.

**NOTE**

If the format is TIFF, the only valid value area parameter is GRATicule, and the only valid palette parameter is MONOchrome.

If the format is something other than TIFF, the only valid area parameter is SCReen, and the only valid values for palette are GRAYscale or COlor.

**Return Format**

```
<display data><NL>  
<display data> ::= binary block data in IEEE-488.2 # format.
```

**See Also**

- "[Introduction to :DISPLAY Commands](#)" on page 208
- "[:DISPLAY:SOURce](#)" on page 216
- "[:HARDcopy:INKSaver](#)" on page 251
- "[:MERGe](#)" on page 140
- "[:PRINT](#)" on page 155
- "[\\*:RCL \(Recall\)](#)" on page 110
- "[\\*:SAV \(Save\)](#)" on page 114
- "[:VIEW](#)" on page 162

**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, SCREEN, COLOR"  
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)  
' Output display data to a file:  
strPath = "c:\scope\data\screen.bmp"  
' Remove file if it exists.  
If Len(Dir(strPath)) Then  
    Kill strPath  
End If  
Close #1      ' If #1 is open, close it.  
Open strPath For Binary Access Write Lock Write As #1      ' Open file f  
or output.  
Put #1, , byteData      ' Write data.  
Close #1      ' Close file.  
myScope.IO.Timeout = 5000
```

Example program from the start: "["VISA COM Example in Visual Basic"](#)" on page 744

**:DISPlay:LABel**

**N** (see page 658)

**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 channel labels.

**Return Format** <value><NL>

<value> ::= {0 | 1}

- See Also**
- "Introduction to :DISPlay Commands" on page 208
  - ":CHANnel<n>:LABEL" on page 197

**Example Code**

```
' DISP_LABEL (not executed in this example)
' - Turns label names ON or OFF on the analyzer display.
myScope.WriteString ":DISPLAY:LABEL ON"      ' Turn on labels.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

## :DISPlay:LABList

**N** (see page 658)

**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 alphabetical 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 208
- "[":DISPLAY:LABEL"](#) on page 213
- "[":CHANnel<n>:LABEL"](#) on page 197

## :DISPlay:PERSistence

**N** (see page 658)

**Command Syntax** :DISPlay:PERSistence <value>  
<value> ::= {MINimum | INFinite}

The :DISPlay:PERSistence command specifies the persistence setting. MINimum indicates zero persistence and INFinite indicates infinite persistence. Use the :DISPlay:CLEar or :CDISplay root command to erase points stored by infinite persistence.

**Query Syntax** :DISPlay:PERSistence?

The :DISPlay:PERSistence? query returns the specified persistence value.

**Return Format** <value><NL>  
<value> ::= {MIN | INF}

**See Also** • "[Introduction to :DISPlay Commands](#)" on page 208  
• "[":DISPlay:CLEar](#)" on page 210  
• "[":CDISplay](#)" on page 131

### :DISPlay:SOURce

**N** (see page 658)

**Command Syntax** :DISPlay:SOURce <value>

```
<value> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3 | PMEMory4  
| PMEMory5 | PMEMory6 | PMEMory7 | PMEMory8 | PMEMory9}
```

PMEMory0-9 ::= pixel memory 0 through 9

The :DISPlay:SOURce command specifies the default source and destination for the :DISPlay:DATA command and query. PMEMory0-9 correspond to the INTERN\_0-9 files found in the front panel Save/Recall menu.

**Query Syntax** :DISPlay:SOURce?

The :DISPlay:SOURce? query returns the specified SOURce.

**Return Format** <value><NL>

```
<value> ::= {PMEM0 | PMEM1 | PMEM2 | PMEM3 | PMEM4 | PMEM5 | PMEM6  
| PMEM7 | PMEM8 | PMEM9}
```

**See Also** • "Introduction to :DISPlay Commands" on page 208

• ":DISPlay:DATA" on page 211

## :DISPlay:VECTors

**N** (see page 658)

**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. When vectors are turned off, only the sampled data is displayed.

**Query Syntax** :DISPlay:VECTors?

The :DISPlay:VECTors? query returns whether vector display is on or off.

**Return Format** <vectors><NL>

<vectors> ::= {1 | 0}

**See Also** • "Introduction to :DISPlay Commands" on page 208

## :EXternal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXternal Trigger Commands" on page 218.

**Table 49** :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see page 220)	:EXternal:BWLimit? (see <a href="#">page 220</a> )	<bwlimit> ::= {0   OFF}
:EXternal:IMPedance <value> (see page 221)	:EXternal:IMPedance? (see <a href="#">page 221</a> )	<impedance> ::= {ONEMeg   FIFTy}
:EXternal:PROBe <attenuation> (see page 222)	:EXternal:PROBe? (see <a href="#">page 222</a> )	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXternal:PROBe:ID? (see <a href="#">page 223</a> )	<probe id> ::= unquoted ASCII string up to 11 characters
:EXternal:PROBe:STYPe <signal type> (see page 224)	:EXternal:PROBe:STYPe? (see <a href="#">page 224</a> )	<signal type> ::= {DIFFerential   SINGLE}
:EXternal:PROTection[:CLEAR] (see page 225)	:EXternal:PROTection? (see <a href="#">page 225</a> )	{NORM   TRIP}
:EXternal:RANGE <range>[<suffix>] (see <a href="#">page 226</a> )	:EXternal:RANGE? (see <a href="#">page 226</a> )	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXternal:UNITS <units> (see page 227)	:EXternal:UNITS? (see <a href="#">page 227</a> )	<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 :EXTerinal query. In this case, the query was issued following a \*RST command.

```
:EXT:BWL 0;IMP ONEM;RANG +8.0E+00;UNIT VOLT;PROB +1.0E+00;PROB:STYP SING
```

### :EXternal:BWLIMIT



(see page 658)

#### Command Syntax

```
:EXternal:BWLIMIT <bwlimit>
<bwlimit> ::= {0 | OFF}
```

The :EXternal:BWLIMIT command is provided for product compatibility. The only legal value is 0 or OFF. Use the :TRIGger:HFReject command to limit bandwidth on the external trigger input.

#### Query Syntax

```
:EXternal:BWLIMIT?
```

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

#### Return Format

```
<bwlimit><NL>
```

```
<bwlimit> ::= 0
```

#### See Also

- "[Introduction to :EXternal Trigger Commands](#)" on page 218
- "[Introduction to :TRIGger Commands](#)" on page 411
- "[":TRIGger:HFReject](#)" on page 415

**:EXTernal:IMPedance**

(see page 658)

**Command Syntax** :EXTernal:IMPedance <value>  
<value> ::= {ONEMeg | FIFTY}

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTY (50Ω).

**Query Syntax** :EXTernal:IMPedance?

The :EXTernal:IMPedance? query returns the current input impedance setting for the external trigger.

**Return Format** <impedance value><NL>  
<impedance value> ::= {ONEM | FIFTY}

**See Also**

- "[Introduction to :EXTernal Trigger Commands](#)" on page 218
- "[Introduction to :TRIGger Commands](#)" on page 411
- "[":CHANnel<n>:IMPedance](#)" on page 195

### :EXternal:PROBe



(see page 658)

**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 218

• "[":EXternal:RANGE"](#) on page 226

• "[":Introduction to :TRIGger Commands"](#) on page 411

• "[":CHANnel<n>:PROBe"](#) on page 199

**:EXTernal:PROBe:ID**

(see page 658)

**Query Syntax** :EXTernal:PROBe:ID?

The :EXTernal:PROBe:ID? query returns the type of probe attached to the external trigger input.

**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 :EXTernal Trigger Commands" on page 218

## **:EXternal:PROBe:STYPe**



(see page 658)

### **Command Syntax**

#### **NOTE**

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

---

```
:EXternal:PROBe:STYPe <signal type>
<signal type> ::= {DIFFerential | SINGle}
```

The :EXternal:PROBe:STYPe command sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

### **Query Syntax**

```
:EXternal:PROBe:STYPe?
```

The :EXternal:PROBe:STYPe? query returns the current probe signal type setting for the external trigger.

### **Return Format**

```
<signal type><NL>
<signal type> ::= {DIFF | SING}
```

### **See Also**

- "Introduction to :EXternal Trigger Commands" on page 218

## :EXternal:PROtection

**N** (see page 658)

**Command Syntax** :EXternal:PROtection[:CLEAR]

When the external trigger input impedance is set to  $50\Omega$ , the external trigger input is protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the external trigger is automatically changed to  $1 M\Omega$ . The :EXternal:PROtection[:CLEAR] command is used to clear (reset) the overload protection. It allows the external trigger to be used again in  $50\Omega$  mode after the signal that caused the overload has been removed from the external trigger input. Reset the external trigger input impedance to  $50\Omega$  (see "[:EXternal:IMPedance](#)" on page 221) after clearing the overvoltage protection.

**Query Syntax** :EXternal:PROtection?

The :EXternal:PROtection query returns the state of the input protection for external trigger. If the external trigger input has experienced an overload, TRIP (trippped) will be returned; otherwise NORM (normal) is returned.

**Return Format** {NORM | TRIP}<NL>

- See Also**
- "[Introduction to :EXternal Trigger Commands](#)" on page 218
  - "[:EXternal:IMPedance](#)" on page 221
  - "[:EXternal:PROBe](#)" on page 222

### :EXternal:RANGE

 (see page 658)

**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:

- In 2-channel models, the range can be set to 1.0 V or 8.0 V.
- In 4-channel models, the range can only be set to 5.0 V.

If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

**Query Syntax** :EXternal:RANGE?

The :EXternal:RANGE? query returns the current full-scale range setting for the external trigger.

**Return Format** <range\_argument><NL>

<range\_argument> ::= external trigger range value in NR3 format

**See Also**

- "[Introduction to :EXternal Trigger Commands](#)" on page 218
- "[:EXternal:PROBe](#)" on page 222
- "[Introduction to :TRIGger Commands](#)" on page 411

## :EXternal:UNITS

**N** (see page 658)

**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 218
- "Introduction to :TRIGger Commands" on page 411
- ":EXternal:RANGE" on page 226
- ":EXternal:PROBe" on page 222
- ":CHANnel<n>:UNITS" on page 206

## :FUNCTION Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCTION Commands" on page 230.

**Table 50** :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTER <frequency> (see page 231)	:FUNCTION:CENTER? (see <a href="#">page 231</a> )	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCTION:DISPLAY {{0   OFF}   {1   ON}} (see <a href="#">page 232</a> )	:FUNCTION:DISPLAY? (see <a href="#">page 232</a> )	{0   1}
:FUNCTION:GOFT:OPERation <operation> (see page 233)	:FUNCTION:GOFT:OPERation? (see <a href="#">page 233</a> )	<operation> ::= {ADD   SUBTract   MULTiply}
:FUNCTION:GOFT:SOURce 1 <source> (see page 234)	:FUNCTION:GOFT:SOURce 1? (see <a href="#">page 234</a> )	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 235)	:FUNCTION:GOFT:SOURce 2? (see <a href="#">page 235</a> )	<source> ::= CHANnel<n> <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURce1 selection <n> ::= {1   2} for 2ch models
:FUNCTION:OFFSet <offset> (see page 236)	:FUNCTION:OFFSet? (see <a href="#">page 236</a> )	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:OPERation <operation> (see page 237)	:FUNCTION:OPERation? (see <a href="#">page 237</a> )	<operation> ::= {ADD   SUBTract   MULTiply   INTegrate   DIFFerentiate   FFT   SQRT}

**Table 50** :FUNCTION Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION:RANGE <range> (see <a href="#">page 238</a> )	:FUNCTION:RANGE? (see <a href="#">page 238</a> )	<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 DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed).  The range for the FFT function is 8 to 800 dBV.
:FUNCTION:REFERENCE <level> (see <a href="#">page 239</a> )	:FUNCTION:REFERENCE? (see <a href="#">page 239</a> )	<level> ::= the value at center screen in NR3 format.  The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:SCALE <scale value>[<suffix>] (see <a href="#">page 240</a> )	:FUNCTION:SCALE? (see <a href="#">page 240</a> )	<scale value> ::= integer in NR1 format  <suffix> ::= {V   dB}
:FUNCTION:SOURce1 <source> (see <a href="#">page 241</a> )	:FUNCTION:SOURce1? (see <a href="#">page 241</a> )	<source> ::= {CHANnel<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models  <n> ::= {1   2} for 2ch models GOFT is only for FFT, INTegrate, DIFFerentiate, and SQRT operations.
:FUNCTION:SOURce2 <source> (see <a href="#">page 242</a> )	:FUNCTION:SOURce2? (see <a href="#">page 242</a> )	<source> ::= {CHANnel<n>   NONE} <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURce1 selection  <n> ::= {1   2} for 2ch models
:FUNCTION:SPAN <span> (see <a href="#">page 243</a> )	:FUNCTION:SPAN? (see <a href="#">page 243</a> )	<span> ::= the current frequency span in NR3 format.  Legal values are 1 Hz to 100 GHz.
:FUNCTION:WINDOW <window> (see <a href="#">page 244</a> )	:FUNCTION:WINDOW? (see <a href="#">page 244</a> )	<window> ::= {RECTangular   HANNing   FLATtop   BHARris}

**Introduction to :FUNCTION Commands** The FUNCTION subsystem controls the math functions in the oscilloscope. Add, subtract, multiply, differentiate, integrate, square root, and FFT (Fast Fourier Transform) operations are available. These math operations only use the analog (vertical) channels.

The SOURCE1, DISPLAY, RANGE, and OFFSET commands apply to any function. The SPAN, CENTER, and WINDOW commands are only useful for FFT functions. When FFT is selected, the cursors change from volts and time to decibels (dB) and frequency (Hz).

### Reporting the Setup

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

### Return Format

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

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

## :FUNCTION:CENTER

**N** (see page 658)

**Command Syntax** :FUNCTION:CENTER <frequency>

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

The :FUNCTION:CENTER command sets the center frequency when FFT (Fast Fourier Transform) is selected.

**Query Syntax** :FUNCTION:CENTER?

The :FUNCTION: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:CENTER? and :FUNCTION:SPAN? queries depend on the current :TIMEbase:RANGE value. Once you change either the :FUNCTION:CENTER or :FUNCTION:SPAN value, they no longer track the :TIMEbase:RANGE value.

### See Also

- "[Introduction to :FUNCTION Commands](#)" on page 230
- "[":FUNCTION:SPAN"](#) on page 243
- "[":TIMEbase:RANGE"](#) on page 404
- "[":TIMEbase:SCALe"](#) on page 406

## **:FUNCTION:DISPLAY**

**N** (see page 658)

**Command Syntax**    :FUNCTION:DISPLAY <display>  
                      <display> ::= {{1 | ON} | {0 | OFF}}

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

**Query Syntax**    :FUNCTION:DISPLAY?

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

**Return Format**    <display><NL>  
                      <display> ::= {1 | 0}

**See Also**    • "Introduction to :FUNCTION Commands" on page 230  
                  • ":VIEW" on page 162  
                  • ":BLANK" on page 130  
                  • ":STATus" on page 159

## :FUNCTION:GOFT:OPERation

**N** (see page 658)

**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 the FFT, INTegrate, DIFFerentiate, or SQRT functions:

- ADD – Source1 + source2.
- SUBTract – Source1 - source2.
- MULTiply – Source1 \* source2.

The :FUNCTION:GOFT:SOURce1 and :FUNCTION:GOFT:SOURce2 commands are used to select source1 and source2.

**Query Syntax** :FUNCTION:GOFT:OPERation?

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

**Return Format** <operation><NL>

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

- See Also**
- "Introduction to :FUNCTION Commands" on page 230
  - ":FUNCTION:GOFT:SOURce1" on page 234
  - ":FUNCTION:GOFT:SOURce2" on page 235
  - ":FUNCTION:SOURce1" on page 241

## **:FUNCTION:GOFT:SOURce1**

**N** (see page 658)

**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 the FFT, INTegrate, DIFFerentiate, or SQRT functions.

**Query Syntax**    :FUNCTION:GOFT:SOURce1?

The :FUNCTION:GOFT:SOURce1? query returns the current selection for the first input channel for the g(t) source.

**Return Format**    <value><NL>  
  
<value> ::= CHAN<n>  
  
<n> ::= {1 | 2 | 3 | 4} for the 4ch models  
  
<n> ::= {1 | 2} for the 2ch models

**See Also**    • "Introduction to :FUNCTION Commands" on page 230  
• ":FUNCTION:GOFT:SOURce2" on page 235  
• ":FUNCTION:GOFT:OPERation" on page 233

**:FUNCTION:GOFT:SOURce2**

**N** (see page 658)

**Command Syntax**

```
:FUNCTION:GOFT:SOURce2 <value>
<value> ::= CHANnel<n>
<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
      selection
<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 the FFT, INTegrate, DIFFerentiate, or SQRT functions.

If CHANnel1 or CHANnel2 is selected for :FUNCTION:GOFT:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCTION:GOFT:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

**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, depending on SOURce1
      selection
<n> ::= {1 | 2} for 2ch models
```

**See Also**

- "Introduction to :FUNCTION Commands" on page 230
- ":FUNCTION:GOFT:SOURce1" on page 234
- ":FUNCTION:GOFT:OPERation" on page 233

### :FUNCTION:OFFSet

**N** (see page 658)

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

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

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

**NOTE**

The :FUNCTION:OFFset command is equivalent to the :FUNCTION:REFERENCE command.

**Query Syntax** :FUNCTION:OFFSet?

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

**Return Format** <offset><NL>

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

- See Also**
- "[Introduction to :FUNCTION Commands](#)" on page 230
  - "[":FUNCTION:RANGE](#)" on page 238
  - "[":FUNCTION:REFERENCE](#)" on page 239
  - "[":FUNCTION:SCALE](#)" on page 240

## :FUNCTION:OPERation

**N** (see page 658)

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

```
<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFFerentiate
                  | FFT | SQRT}
```

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

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

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

**Query Syntax** :FUNCTION:OPERation?

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

**Return Format** <operation><NL>

```
<operation> ::= {ADD | SUBT | MULT | INT | DIFF | FFT | SQRT}
```

**See Also** • "Introduction to :FUNCTION Commands" on page 230

• ":FUNCTION:SOURce1" on page 241

• ":FUNCTION:SOURce2" on page 242

## **:FUNCTION:RANGE**

**N** (see page 658)

**Command Syntax**    `:FUNCTION:RANGE <range>`

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

The :FUNCTION:RANGE command defines the full-scale vertical axis for the selected function.

**Query Syntax**    `:FUNCTION:RANGE?`

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

**Return Format**    `<range><NL>`

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

The range for ADD, SUBT, MULT is 8E-6 to 800E+3.

The range for the INTegrate function is 8E-9 to 400E+3 (depends on sweep speed).

The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on sweep speed).

The range for the FFT (Fast Fourier Transform) function is 8 to 800 dBV.

**See Also**

- "[Introduction to :FUNCTION Commands](#)" on page 230
- "[":FUNCTION:SCALE"](#) on page 240

**:FUNCTION:REFERENCE**

**N** (see page 658)

**Command Syntax** :FUNCTION:REFERENCE <level>

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

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

**NOTE**

The FUNCTION:REFERENCE command is equivalent to the :FUNCTION:OFFSET command.

**Query Syntax** :FUNCTION:REFERENCE?

The :FUNCTION:REFERENCE? query outputs the current reference level value for the selected function.

**Return Format** <level><NL>

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

- See Also**
- "[Introduction to :FUNCTION Commands](#)" on page 230
  - "[":FUNCTION:OFFSET](#)" on page 236
  - "[":FUNCTION:RANGE](#)" on page 238
  - "[":FUNCTION:SCALE](#)" on page 240

## **:FUNCTION:SCALE**

**N** (see page 658)

**Command Syntax**    :FUNCTION:SCALE <scale value>[<suffix>]  
                    <scale value> ::= integer in NR1 format  
                    <suffix> ::= {V | dB}

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

**Query Syntax**    :FUNCTION:SCALE?

The :FUNCTION:SCALE? query returns the current scale value for the selected function.

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

**See Also**    • "Introduction to :FUNCTION Commands" on page 230  
                    • ":FUNCTION:RANGE" on page 238

## :FUNCTION:SOURce1

**N** (see page 658)

### Command Syntax

```
:FUNCTION:SOURce1 <value>
<value> ::= {CHANnel<n> | GOFT}
<n> ::= {1 | 2 | 3 | 4} for 4ch models
<n> ::= {1 | 2} for 2ch models
```

The :FUNCTION:SOURce1 command is used for any :FUNCTION:OPERation selection (including the ADD, SUBTract, or MULTIply channel math operations and the FFT, INTegrate, DIFFerentiate, or SQRT transforms). This command selects the first source for channel math operations or the single source for the transforms.

The GOFT parameter is only available for the FFT, INTegrate, DIFFerentiate, or SQRT functions. It lets you specify, as the function input source, the addition, subtraction, or multiplication of two channels. When GOFT is used, the g(t) source is specified by the :FUNCTION:GOFT:OPERation, :FUNCTION:GOFT:SOURce1, and :FUNCTION:GOFT:SOURce2 commands.

### NOTE

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

### Query Syntax

```
:FUNCTION:SOURce1?
```

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

### Return Format

```
<value><NL>
<value> ::= {CHAN<n> | GOFT}
<n> ::= {1 | 2 | 3 | 4} for 4ch models
<n> ::= {1 | 2} for 2ch models
```

### See Also

- "Introduction to :FUNCTION Commands" on page 230
- ":FUNCTION:OPERation" on page 237
- ":FUNCTION:GOFT:OPERation" on page 233
- ":FUNCTION:GOFT:SOURce1" on page 234
- ":FUNCTION:GOFT:SOURce2" on page 235

## :FUNCTION:SOURce2

**N** (see page 658)

**Command Syntax** :FUNCTION:SOURce2 <value>

```
<value> ::= {CHANnel<n> | NONE}
<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
      selection
<n> ::= {1 | 2} for 2ch models
```

The :FUNCTION:SOURce2 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:SOURce2 command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCTION:OPERation command.

If CHANnel1 or CHANnel2 is selected for :FUNCTION:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCTION:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

**Query Syntax** :FUNCTION:SOURce2?

The :FUNCTION:SOURce2? query returns the second source for function operations on two waveforms.

**Return Format** <value><NL>

```
<value> ::= {CHAN<n> | NONE}
<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
      selection
<n> ::= {1 | 2} for 2ch models
```

**See Also**

- "Introduction to :FUNCTION Commands" on page 230
- ":FUNCTION:OPERation" on page 237

## :FUNCTION:SPAN

**N** (see page 658)

### Command Syntax

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

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

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

### Query Syntax

```
:FUNCTION:SPAN?
```

The :FUNCTION:SPAN? query returns the current frequency span in Hertz.

### NOTE

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

### Return Format

```
<span><NL>
```

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

### See Also

- "[Introduction to :FUNCTION Commands](#)" on page 230
- "[":FUNCTION:CENTER"](#) on page 231
- "[":TIMEbase:RANGE"](#) on page 404
- "[":TIMEbase:SCALe"](#) on page 406

### :FUNCTION:WINDOW

**N** (see page 658)

**Command Syntax** :FUNCTION:WINDOW <window>  
<window> ::= {RECTangular | HANNing | FLATtop | BHARris}

The :FUNCTION: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:WINDOW?

The :FUNCTION:WINDOW? query returns the value of the window selected for the FFT function.

**Return Format** <window><NL>  
<window> ::= {RECT | HANN | FLAT | BHAR}

**See Also** • "Introduction to :FUNCTION Commands" on page 230

## :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "Introduction to :HARDcopy Commands" on page 246.

**Table 51** :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see <a href="#">page 247</a> )	:HARDcopy:AREA? (see <a href="#">page 247</a> )	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see <a href="#">page 248</a> )	:HARDcopy:APRinter? (see <a href="#">page 248</a> )	<active_printer> ::= {<index>   <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 249</a> )	:HARDcopy:FACTors? (see <a href="#">page 249</a> )	{0   1}
:HARDcopy:FFEed {{0   OFF}   {1   ON}} (see <a href="#">page 250</a> )	:HARDcopy:FFEed? (see <a href="#">page 250</a> )	{0   1}
:HARDcopy:INKSaver { {0   OFF}   {1   ON}} (see <a href="#">page 251</a> )	:HARDcopy:INKSaver? (see <a href="#">page 251</a> )	{0   1}
:HARDcopy:LAYout <layout> (see <a href="#">page 252</a> )	:HARDcopy:LAYout? (see <a href="#">page 252</a> )	<layout> ::= {LANDscape   PORTRait}
:HARDcopy:PAlette <palette> (see <a href="#">page 253</a> )	:HARDcopy:PAlette? (see <a href="#">page 253</a> )	<palette> ::= {COLor   GRAYscale   NONE}
n/a	:HARDcopy:PRINTER:LIS T? (see <a href="#">page 254</a> )	<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 <a href="#">page 255</a> )	n/a	n/a

## 5 Commands by Subsystem

**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 658)

**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 246
  - "[":HARDcopy:STARt](#)" on page 255
  - "[":HARDcopy:APRinter](#)" on page 248
  - "[":HARDcopy:PRINter:LIST](#)" on page 254
  - "[":HARDcopy:FACTors](#)" on page 249
  - "[":HARDcopy:FFEed](#)" on page 250
  - "[":HARDcopy:INKSaver](#)" on page 251
  - "[":HARDcopy:LAYout](#)" on page 252
  - "[":HARDcopy:PALETTE](#)" on page 253

## **:HARDcopy:APRinter**

**N** (see page 658)

**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 246

• ":HARDcopy:PRINter:LIST" on page 254

• ":HARDcopy:STARt" on page 255

## :HARDcopy:FACTors

**N** (see page 658)

**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 246
  - "[":HARDcopy:STARt](#)" on page 255
  - "[":HARDcopy:FFEed](#)" on page 250
  - "[":HARDcopy:INKSaver](#)" on page 251
  - "[":HARDcopy:LAYOUT](#)" on page 252
  - "[":HARDcopy:PALETTE](#)" on page 253

### :HARDcopy:FFEed

**N** (see page 658)

**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.

ON (or 1) is only valid when PRINter0 or PRINter1 is set as the :HARDcopy:FORMAT type.

**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 246

- "[":HARDcopy:START](#)" on page 255
- "[":HARDcopy:FACTors](#)" on page 249
- "[":HARDcopy:INKSaver](#)" on page 251
- "[":HARDcopy:LAyout](#)" on page 252
- "[":HARDcopy:PAlette](#)" on page 253

**:HARDcopy:INKSaver****N** (see page 658)**Command Syntax** :HARDcopy:INKSaver <value>

&lt;value&gt; ::= {{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>

&lt;value&gt; ::= {0 | 1}

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 246
  - "[":HARDcopy:STARt](#)" on page 255
  - "[":HARDcopy:FACTors](#)" on page 249
  - "[":HARDcopy:FFEed](#)" on page 250
  - "[":HARDcopy:LAYOUT](#)" on page 252
  - "[":HARDcopy:PALETTE](#)" on page 253

## **:HARDcopy:LAYout**

**N** (see page 658)

**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 246

- "[":HARDcopy:STARt](#)" on page 255
- "[":HARDcopy:FACTors](#)" on page 249
- "[":HARDcopy:PALETTE](#)" on page 253
- "[":HARDcopy:FFEEd](#)" on page 250
- "[":HARDcopy:INKSaver](#)" on page 251

## :HARDcopy:PALETTE

**N** (see page 658)

**Command Syntax** :HARDcopy:PALETTE <palette>

<palette> ::= {COLOR | GRAYSCALE | NONE}

The :HARDcopy:PALETTE command sets the hardcopy palette color.

**NOTE**

If no printer is connected, NONE is the only valid parameter.

---

**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 246
  - "[":HARDcopy:STARt](#)" on page 255
  - "[":HARDcopy:FACTors](#)" on page 249
  - "[":HARDcopy:LAYout](#)" on page 252
  - "[":HARDcopy:FFEed](#)" on page 250
  - "[":HARDcopy:INKSaver](#)" on page 251

## **:HARDcopy:PRINter:LIST**

**N** (see page 658)

**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 246
- "[":HARDcopy:APRinter](#)" on page 248
- "[":HARDcopy:STARt](#)" on page 255

**:HARDcopy:STARt**

**N** (see page 658)

**Command Syntax** :HARDcopy:STARt

The :HARDcopy:STARt command starts a print job.

**See Also**

- "[Introduction to :HARDcopy Commands](#)" on page 246
- "[":HARDcopy:APRinter](#)" on page 248
- "[":HARDcopy:PRINTER:LIST](#)" on page 254
- "[":HARDcopy:FACTors](#)" on page 249
- "[":HARDcopy:FFEed](#)" on page 250
- "[":HARDcopy:INKSaver](#)" on page 251
- "[":HARDcopy:LAYOUT](#)" on page 252
- "[":HARDcopy:PALETTE](#)" on page 253

## :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 257.

**Table 52** :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 258)	:MARKer:MODE? (see page 258)	<mode> ::= {OFF   MEASurement   MANual   WAVeform}
:MARKer:X1Position <position>[suffix] (see page 259)	:MARKer:X1Position? (see page 259)	<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 260)	:MARKer:X1Y1source? (see page 260)	<source> ::= {CHANnel<n>   FUNCtion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 261)	:MARKer:X2Position? (see page 261)	<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 262)	:MARKer:X2Y2source? (see page 262)	<source> ::= {CHANnel<n>   FUNCtion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 263)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see page 264)	:MARKer:Y1Position? (see page 264)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V   mV   dB} <return_value> ::= Y1 cursor position value in NR3 format

**Table 52** :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:Y2Position <position>[suffix] (see <a href="#">page 265</a> )	:MARKer:Y2Position? (see <a href="#">page 265</a> )	<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 <a href="#">page 266</a> )	<return_value> ::= Y cursors delta value in NR3 format

**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 NONE;X2Y2 NONE;MODE OFF
```

**:MARKer:MODE**

**N** (see page 658)

**Command Syntax**

```
:MARKer:MODE <mode>
<mode> ::= {OFF | MEASurement | MANual | WAVEform}
```

The :MARKer:MODE command sets the cursors mode:

- OFF – removes the cursor information from the display.
- MANual – enables manual placement of the X and Y cursors.

If the front-panel cursors are off, or are set to the front-panel Hex or Binary mode, setting :MARKer:MODE MANual will put the cursors in the front-panel Normal mode.

- MEASurement – cursors track the most recent measurement.

Setting the mode to MEASurement sets the marker sources (:MARKer:X1Y1source and :MARKer:X2Y2source) to the measurement source (:MEASure:SOURce). Setting the measurement source remotely always sets the marker sources.

- WAVEform – the Y1 cursor tracks the voltage value at the X1 cursor of the waveform specified by the X1Y1source, and the Y2 cursor does the same for the X2 cursor and its X2Y2source.

**Query Syntax**

```
:MARKer:MODE?
```

The :MARKer:MODE? query returns the current cursors mode.

**Return Format**

```
<mode><NL>
<mode> ::= {OFF | MEAS | MAN | WAV}
```

**See Also**

- "Introduction to :MARKer Commands" on page 257
- ":MARKer:X1Y1source" on page 260
- ":MARKer:X2Y2source" on page 262
- ":MEASure:SOURce" on page 297
- ":MARKer:X1Position" on page 259
- ":MARKer:X2Position" on page 261
- ":MARKer:Y1Position" on page 264
- ":MARKer:Y2Position" on page 265

## :MARKer:X1Position

**N** (see page 658)

### 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 258).
- Sets the X1 cursor position to the specified value.

### 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 257
- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X2Position](#)" on page 261
- "[:MARKer:X1Y1source](#)" on page 260
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MEASure:TSTArt](#)" on page 591

**:MARKer:X1Y1source**

**N** (see page 658)

**Command Syntax**

```
:MARKer:X1Y1source <source>
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 258):

- 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, or MATH 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 | NONE}
```

**See Also**

- "[Introduction to :MARKer Commands](#)" on page 257
- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MEASure:SOURce](#)" on page 297

## :MARKer:X2Position

**N** (see page 658)

**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 258).
- Sets the X2 cursor position to the specified value.

**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 257

- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X1Position](#)" on page 259
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MEASure:TSTOp](#)" on page 592

**:MARKer:X2Y2source**

**N** (see page 658)

**Command Syntax**

```
:MARKer:X2Y2source <source>
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 258):

- 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, or MATH 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 | NONE}
```

**See Also**

- "[Introduction to :MARKer Commands](#)" on page 257
- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X1Y1source](#)" on page 260
- "[:MEASure:SOURce](#)" on page 297

**:MARKer:XDELta****N** (see page 658)**Query Syntax** :MARKer:XDELta?

The MARKer:XDELta? query returns the value difference between the current X1 and X2 cursor positions.

$$\text{Xdelta} = (\text{Value at X2 cursor}) - (\text{Value at X1 cursor})$$

**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 257
- "[":MARKer:MODE](#)" on page 258
- "[":MARKer:X1Position](#)" on page 259
- "[":MARKer:X2Position](#)" on page 261
- "[":MARKer:X1Y1source](#)" on page 260
- "[":MARKer:X2Y2source](#)" on page 262

## :MARKer:Y1Position

**N** (see page 658)

**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 258), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y1 cursor position to the specified value.

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 257
- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X1Y1source](#)" on page 260
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MARKer:Y2Position](#)" on page 265
- "[:MEASure:VSTArt](#)" on page 597

## :MARKer:Y2Position

**N** (see page 658)

**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 258), the :MARKer:Y1Position command:

- Sets :MARKer:MODE to MANual.
- Sets the Y2 cursor position to the specified value.

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 257

- "[:MARKer:MODE](#)" on page 258
- "[:MARKer:X1Y1source](#)" on page 260
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MARKer:Y1Position](#)" on page 264
- "[:MEASure:VSTOp](#)" on page 598

## :MARKer:YDELta

**N** (see page 658)

**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.

**Return Format** <value><NL>

<value> ::= difference value in NR3 format

**See Also**

- "[Introduction to :MARKer Commands](#)" on page 257
- "[":MARKer:MODE"](#) on page 258
- "[":MARKer:X1Y1source"](#) on page 260
- "[":MARKer:X2Y2source"](#) on page 262
- "[":MARKer:Y1Position"](#) on page 264
- "[":MARKer:Y2Position"](#) on page 265

## :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 272.

**Table 53** :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 274)	n/a	n/a
:MEASure:COUNTER [ <source/> ] (see page 275)	:MEASure:COUNTER? [<source>] (see page 275)	<source> ::= {CHANnel<n>   EXTernal} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 276)	:MEASure:DEFine? DELay (see page 277)	<delay spec> ::= <edge_spec1>, <edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+   -} <occurrence> ::= integer
:MEASure:DEFine THresholds, <threshold spec> (see page 276)	:MEASure:DEFine? THresholds (see page 277)	<threshold spec> ::= {STANDARD}   {<threshold mode>, <upper>, <middle>, <lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DELay [ <source1>] [, &lt;source2&gt;] (see page 279)</source1>	:MEASure:DELay? [<source1>] [, <source2>] (see page 279)	<source1,2> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 281)	:MEASure:DUTYcycle? [<source>] (see page 281)	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format

## 5 Commands by Subsystem

**Table 53** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 282)	:MEASure:FALLtime? [<source>] (see page 282)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 283)	:MEASure:FREQuency? [<source>] (see page 283)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 284)	:MEASure:NWIDth? [<source>] (see page 284)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 285)	:MEASure:OVERshoot? [<source>] (see page 285)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 287)	:MEASure:PERiod? [<source>] (see page 287)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASE [<source1>] [,<source2>] (see page 288)	:MEASure:PHASE? [<source1>] [,<source2>] (see page 288)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 289)	:MEASure:PREShoot? [<source>] (see page 289)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format

**Table 53** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:PWIDth [ <source/> ] (see page 290)	:MEASure:PWIDth? [ <source/> ] (see page 290)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
n/a	:MEASure:RESults? <result_list> (see page 291)	<result_list> ::= comma-separated list of measurement results
:MEASure:RISEtime [ <source/> ] (see page 294)	:MEASure:RISEtime? [ <source/> ] (see page 294)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [ <source/> ] (see page 295)	:MEASure:SDEViation? [ <source/> ] (see page 295)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1   ON} (see page 296)	:MEASure:SHOW? (see page 296)	{1}
:MEASure:SOURce <source1> [,<source2>] (see page 297)	:MEASure:SOURce? (see page 297)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH   EXTernal} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= {<source>   NONE}
:MEASure:STATistics <type> (see page 299)	:MEASure:STATistics? (see page 299)	<type> ::= {{ON   1}   CURRENT   MEAN   MINimum   MAXimum   STDDev   COUNT} ON ::= all statistics returned
:MEASure:STATistics:INCrement (see page 300)	n/a	n/a
:MEASure:STATistics:RESet (see page 301)	n/a	n/a

## 5 Commands by Subsystem

**Table 53 :MEASure Commands Summary (continued)**

Command	Query	Options and Query Returns
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see <a href="#">page 302</a> )	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see <a href="#">page 304</a> )	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPplitude [<source>] (see <a href="#">page 306</a> )	:MEASure:VAMPplitude? [<source>] (see <a href="#">page 306</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<source>] (see <a href="#">page 307</a> )	:MEASure:VAverage? [<source>] (see <a href="#">page 307</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASE [<source>] (see <a href="#">page 308</a> )	:MEASure:VBASE? [<source>] (see <a href="#">page 308</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

**Table 53** :MEASure Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MEASure:VMAX [<source>] (see page 309)	:MEASure:VMAX? [<source>] (see page 309)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 310)	:MEASure:VMIN? [<source>] (see page 310)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 311)	:MEASure:VPP? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 288)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 312)	<source1,2> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 313)	:MEASure:VRMS? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtimer>[,<source>] (see page 314)	<vtimer> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see page 315)	:MEASure:VTOP? [<source>] (see page 315)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format

## 5 Commands by Subsystem

**Table 53** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:XMAX [<source>] (see page 316)	:MEASure:XMAX? [<source>] (see page 316)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 317)	:MEASure:XMIN? [<source>] (see page 317)	<source> ::= {CHANnel<n>   FUNCTion   MATH} <n> ::= 1-2 or 1-4 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 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:CLEar**

**N** (see page 658)

**Command Syntax** :MEASure:CLEar

This command clears all selected measurements and markers from the screen.

**See Also** • "Introduction to :MEASure Commands" on page 272

## :MEASure:COUNter

**N** (see page 658)

**Command Syntax** :MEASure:COUNter [<source>]  
 <source> ::= {CHANnel<n> | EXTERNAL}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math may be selected for the source.

The counter measurement counts trigger level crossings at the selected trigger slope and displays the results in Hz. The gate time for the measurement is automatically adjusted to be 100 ms or twice the current time window, whichever is longer, up to 1 second. The counter measurement can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 X gate time).

The Y cursor shows the 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 272
- ":MEASure:SOURce" on page 297
- ":MEASure:FREQuency" on page 283
- ":MEASure:CLEar" on page 274

**:MEASure:DEFine**

**N** (see page 658)

**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 205: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.

## 5 Commands by Subsystem

**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 272
- "[":MEASure:DELay](#)" on page 279
- "[":MEASure:SOURce](#)" on page 297
- "[":CHANnel<n>:RANGE](#)" on page 204
- "[":CHANnel<n>:SCALE](#)" on page 205
- "[":CHANnel<n>:PROBe](#)" on page 199
- "[":CHANnel<n>:UNITs](#)" on page 206

## :MEASure:DElay

**N** (see page 658)

**Command Syntax**

```
:MEASure:DElay [<source1>[,<source2>]
<source1>, <source2> ::= {CHANnel<n> | FUNCTion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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

## 5 Commands by Subsystem

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.

<b>Return Format</b>	<value><NL> <value> ::= floating-point number delay time in seconds in NR3 format
<b>See Also</b>	<ul style="list-style-type: none"><li>• "<a href="#">Introduction to :MEASure Commands</a>" on page 272</li><li>• "<a href="#">":MEASure:DEFIne</a>" on page 276</li><li>• "<a href="#">":MEASure:PHASe</a>" on page 288</li></ul>

## :MEASure:DUTYcycle



(see page 658)

### Command Syntax

```
:MEASure:DUTYcycle [<source>]
<source> ::= {CHANnel<n> | FUNCTion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:DUTYcycle command installs a screen measurement and starts a duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

### NOTE

The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

### Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

$$\text{duty cycle} = (\text{pulse width}/\text{period}) * 100$$

### Return Format

```
<value><NL>
<value> ::= ratio of positive pulse width to period in NR3 format
```

### See Also

- "[Introduction to :MEASure Commands](#)" on page 272
- "[:MEASure:PERiod](#)" on page 287
- "[:MEASure:PWIDth](#)" on page 290
- "[:MEASure:SOURce](#)" on page 297

### Example Code

- "[Example Code](#)" on page 297

### :MEASure:FALLtime



(see page 658)

#### Command Syntax

```
:MEASure:FALLtime [<source>]  
<source> ::= {CHANnel<n> | FUNCTion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- ":MEASure:RISetime" on page 294
- ":MEASure:SOURce" on page 297

**:MEASure:FREQuency**

(see page 658)

**Command Syntax** :MEASure:FREQuency [<source>]

&lt;source&gt; ::= {CHANnel&lt;n&gt; | FUNCTION | MATH}

&lt;n&gt; ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

&lt;n&gt; ::= {1 | 2} for the two channel oscilloscope models

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 272

- ":MEASure:SOURce" on page 297

- ":MEASure:PERiod" on page 287

**Example Code** • "Example Code" on page 297

### :MEASure:NWIDth



(see page 658)

#### Command Syntax

```
:MEASure:NWIDth [<source>]  
<source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:NWIDth command installs a screen measurement and starts a negative 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: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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:PWIDth](#)" on page 290
- "[":MEASure:PERiod](#)" on page 287

**:MEASure:OVERshoot**

(see page 658)

**Command Syntax**

```
:MEASure:OVERshoot [<source>]
<source> ::= {CHANnel<n> | FUNCTION | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[:MEASure:PREShoot](#)" on page 289
- "[:MEASure:SOURce](#)" on page 297
- "[:MEASure:VMAX](#)" on page 309

## **5 Commands by Subsystem**

- "[:MEASure:VTOP](#)" on page 315
- "[:MEASure:VBASe](#)" on page 308
- "[:MEASure:VMIN](#)" on page 310

## :MEASure:PERiod



(see page 658)

### Command Syntax

```
:MEASure:PERiod [<source>]
<source> ::= {CHANnel<n> | FUNCTION | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:NWIDth](#)" on page 284
- "[":MEASure:PVIDth](#)" on page 290
- "[":MEASure:FREQuency](#)" on page 283

### Example Code

- "[Example Code](#)" on page 297

### :MEASure:PHASe

**N** (see page 658)

**Command Syntax** :MEASure:PHASe [<source1>[,<source2>]  
<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

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 272
- "[:MEASure:DELay](#)" on page 279
- "[:MEASure:PERiod](#)" on page 287
- "[:MEASure:SOURce](#)" on page 297

## :MEASure:PREShoot



(see page 658)

### Command Syntax

```
:MEASure:PREShoot [<source>]
<source> ::= {CHANnel<n> | FUNCTION | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:VMIN](#)" on page 310
- "[":MEASure:VMAX](#)" on page 309
- "[":MEASure:VTOP](#)" on page 315
- "[":MEASure:VBASE](#)" on page 308

### :MEASure:PWIDth



(see page 658)

#### Command Syntax

```
:MEASure:PWIDth [<source>]  
<source> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:NWIDth](#)" on page 284
- "[":MEASure:PERiod](#)" on page 287

## :MEASure:RESults

**N** (see page 658)

### Query Syntax

:MEASure:RESults?

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values.

If more than one measurement is running continuously, the :MEASure:RESults return values are duplicated for each continuous measurement from the first to last (left to right) result displayed. Each result returned is separated from the previous result by a comma. There is a maximum of four continuous measurements that can be continuously displayed at a time.

When no quick measurements are installed, the :MEASure:RESults? query returns nothing (empty string). When the count for any of the measurements is 0, the value of infinity (9.9E+37) is returned for the min, max, mean, and standard deviation.

### Return Format

<result\_list><NL>

<result\_list> ::= comma-separated list of measurement results

The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measure ment 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 272
- ":MEASure:STATistics" on page 299

### 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
```

## 5 Commands by Subsystem

```
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

        Exit Sub

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

    End Sub

```

### :MEASure:RISetime

 (see page 658)

#### Command Syntax

```
:MEASure: RISetime [<source>]  
<source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:FALLtime](#)" on page 282

## :MEASure:SDEViation

**N** (see page 658)

**Command Syntax** :MEASure:SDEViation [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- ":MEASure:SOURce" on page 297

## **:MEASure:SHOW**

**N** (see page 658)

**Command Syntax**    :MEASure:SHOW <show>  
                    <show> ::= {1 | ON}

The :MEASure:SHOW command enables markers for tracking measurements on the display. This feature is always on.

**Query Syntax**    :MEASure:SHOW?

The :MEASure:SHOW? query returns the current state of the markers.

**Return Format**    <show><NL>  
                    <show> ::= 1

**See Also**    • "Introduction to :MEASure Commands" on page 272

## :MEASure:SOURce

**C** (see page 658)

**Command Syntax**

```
:MEASure:SOURce <source1>[,<source2>]
<source1>,<source2> ::= {CHANnel<n> | FUNCtion | MATH | EXTernal}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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**

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> ::= {CHAN<n> | FUNC | EXT | NONE}
```

### See Also:

- "[Introduction to :MEASure Commands](#)" on page 272
- "[":MARKer:MODE](#)" on page 258
- "[":MARKer:X1Y1source](#)" on page 260
- "[":MARKer:X2Y2source](#)" on page 262
- "[":MEASure:DELay](#)" on page 279
- "[":MEASure:PHASe](#)" on page 288

### 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.
```

## 5 Commands by Subsystem

```
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"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

## :MEASure:STATistics

**N** (see page 658)

**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 272
  - "[:MEASure:RESults](#)" on page 291
  - "[:MEASure:STATistics:RESet](#)" on page 301
  - "[:MEASure:STATistics:INCReement](#)" on page 300

**Example Code**

- "["Example Code"](#) on page 291

## :MEASure:STATistics:INCRement

**N** (see page 658)

### 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 272
- "[":MEASure:STATistics](#)" on page 299
- "[":MEASure:STATistics:RESet](#)" on page 301
- "[":MEASure:RESults](#)" on page 291

**:MEASure:STATistics:RESet**

**N** (see page 658)

**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 272
  - "[:MEASure:STATistics](#)" on page 299
  - "[:MEASure:RESults](#)" on page 291
  - "[:MEASure:STATistics:INCRelement](#)" on page 300

**Example Code**

- "[Example Code](#)" on page 291

**:MEASure:TEDGE**

**N** (see page 658)

**Query Syntax** :MEASure:TEDGE? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a space or plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing from the left screen edge is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {CHANnel<n> | FUNCTion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 303.

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)

```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 744

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 272
  - [":MEASure:TVALUE"](#) on page 304
  - [":MEASure:VTIME"](#) on page 314

**:MEASure:TVALue**

(see page 658)

**Query Syntax**

```
:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]
```

<value> ::= the vertical value that the waveform must cross. The value can be volts or a math function value such as dB, Vs, or V/s.

<slope> ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {CHANnel<n> | FUNCTion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 272
  - "[":MEASure:TEDGE](#)" on page 302
  - "[":MEASure:VTIME](#)" on page 314

### :MEASure:VAMPlitude

 (see page 658)

**Command Syntax** :MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 272
  - "[:MEASure:SOURce](#)" on page 297
  - "[:MEASure:VBASe](#)" on page 308
  - "[:MEASure:VTOP](#)" on page 315
  - "[:MEASure:VPP](#)" on page 311

**:MEASure:VAverage**

**C** (see page 658)

**Command Syntax** :MEASure:VAverage [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:VAverage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VAverage? [<source>]

The :MEASure:VAverage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

**Return Format** <value><NL>

<value> ::= calculated average value in NR3 format

**See Also**

- "Introduction to :MEASure Commands" on page 272
- ":MEASure:SOURce" on page 297

### :MEASure:VBASe



(see page 658)

#### Command Syntax

```
:MEASure:VBASe [<source>]  
<source> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[:MEASure:SOURce](#)" on page 297
- "[:MEASure:VTOP](#)" on page 315
- "[:MEASure:VAMPLitude](#)" on page 306
- "[:MEASure:VMIN](#)" on page 310

**:MEASure:VMAX**

(see page 658)

**Command Syntax** :MEASure:VMAX [<source>]

&lt;source&gt; ::= {CHANnel&lt;n&gt; | FUNCtion | MATH}

&lt;n&gt; ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

&lt;n&gt; ::= {1 | 2} for the two channel oscilloscope models

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 272
  - "[":MEASure:SOURce](#)" on page 297
  - "[":MEASure:VMIN](#)" on page 310
  - "[":MEASure:VPP](#)" on page 311
  - "[":MEASure:VTOP](#)" on page 315

### :MEASure:VMIN



(see page 658)

#### Command Syntax

```
:MEASure:VMIN [<source>]  
<source> ::= {CHANnel<n> | FUNCTion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce"](#) on page 297
- "[":MEASure:VBASe"](#) on page 308
- "[":MEASure:VMAX"](#) on page 309
- "[":MEASure:VPP"](#) on page 311

**:MEASure:VPP**

(see page 658)

**Command Syntax** :MEASure:VPP [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
  - "[:MEASure:SOURce](#)" on page 297
  - "[:MEASure:VMAX](#)" on page 309
  - "[:MEASure:VMIN](#)" on page 310
  - "[:MEASure:VAMPLitude](#)" on page 306

### :MEASure:VRATio

**N** (see page 658)

**Command Syntax** :MEASure:VRATio [<source1>[,<source2>]  
<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:VRATio command places the instrument in the continuous measurement mode and starts a ratio measurement.

**Query Syntax** :MEASure:VRATio? [<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 272
- ":MEASure:VRMS" on page 313
- ":MEASure:SOURce" on page 297

**:MEASure:VRMS**

(see page 658)

**Command Syntax**

```
:MEASure:VRMS [<source>]
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VRMS command installs a screen measurement and starts a dc RMS 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:VRMS? [<source>]
```

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

**Return Format**

```
<value><NL>
<value> ::= calculated dc RMS value in NR3 format
```

**See Also**

- "Introduction to :MEASure Commands" on page 272
- ":MEASure:SOURce" on page 297

## :MEASure:VTIMe

**N** (see page 658)

**Query Syntax** :MEASure:VTIMe? <vtimetime\_argument>[,<source>]

<vtimetime\_argument> ::= time from trigger in seconds

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 272
- "[:MEASure:SOURce](#)" on page 297
- "[:MEASure:TEDGE](#)" on page 302
- "[:MEASure:TVALue](#)" on page 304

**:MEASure:VTOP**

(see page 658)

**Command Syntax**

```
:MEASure:VTOP [⟨source⟩]
⟨source⟩ ::= {CHANnel⟨n⟩ | FUNCtion | MATH}
⟨n⟩ ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
⟨n⟩ ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:VMAX](#)" on page 309
- "[":MEASure:VAMPplitude](#)" on page 306
- "[":MEASure:VBASe](#)" on page 308

### :MEASure:XMAX

**N** (see page 658)

**Command Syntax** :MEASure:XMAX [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
- "[:MEASure:XMIN](#)" on page 317
- "[:MEASure:TMAX](#)" on page 589

## :MEASure:XMIN

**N** (see page 658)

**Command Syntax** :MEASure:XMIN [<source>]

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 272
  - "[":MEASure:XMAX"](#) on page 316
  - "[":MEASure:TMIN"](#) on page 590

## :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "Introduction to :MTESt Commands" on page 320.

**Table 54** :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:AMASK:CREAtE (see <a href="#">page 323</a> )	n/a	n/a
:MTESt:AMASK:SOURce <source> (see <a href="#">page 324</a> )	:MTESt:AMASK:SOURce? (see <a href="#">page 324</a> )	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:MTESt:AMASK:UNITS <units> (see <a href="#">page 325</a> )	:MTESt:AMASK:UNITS? (see <a href="#">page 325</a> )	<units> ::= {CURREnt   DIVisions}
:MTESt:AMASK:XDELta <value> (see <a href="#">page 326</a> )	:MTESt:AMASK:XDELta? (see <a href="#">page 326</a> )	<value> ::= X delta value in NR3 format
:MTESt:AMASK:YDELta <value> (see <a href="#">page 327</a> )	:MTESt:AMASK:YDELta? (see <a href="#">page 327</a> )	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNT:FWAvefor ms? [CHANnel<n>] (see <a href="#">page 328</a> )	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNT:RESet (see <a href="#">page 329</a> )	n/a	n/a
n/a	:MTESt:COUNT:TIME? (see <a href="#">page 330</a> )	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNT:WAVeform s? (see <a href="#">page 331</a> )	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see <a href="#">page 332</a> )	:MTESt:DATA? (see <a href="#">page 332</a> )	<mask> ::= data in IEEE 488.2 # format.
:MTESt:DELETE (see <a href="#">page 333</a> )	n/a	n/a
:MTESt:ENABle {{0   OFF}   {1   ON}} (see <a href="#">page 334</a> )	:MTESt:ENABle? (see <a href="#">page 334</a> )	{0   1}
:MTESt:LOCK {{0   OFF}   {1   ON}} (see <a href="#">page 335</a> )	:MTESt:LOCK? (see <a href="#">page 335</a> )	{0   1}

**Table 54** :MTEST Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:MTEST:OUTPut <signal> (see page 336)	:MTEST:OUTPut? (see page 336)	<signal> ::= {FAIL   PASS}
:MTEST:RMODE <rmode> (see page 337)	:MTEST:RMODE? (see page 337)	<rmode> ::= {FORever   TIME   SIGMa   WAVEforms}
:MTEST:RMODE:FACTion:PRINT {{0   OFF}   {1   ON}} (see page 338)	:MTEST:RMODE:FACTion:PRINT? (see page 338)	{0   1}
:MTEST:RMODE:FACTion:SAVE {{0   OFF}   {1   ON}} (see page 339)	:MTEST:RMODE:FACTion:SAVE? (see page 339)	{0   1}
:MTEST:RMODE:FACTion:STOP {{0   OFF}   {1   ON}} (see page 340)	:MTEST:RMODE:FACTion:STOP? (see page 340)	{0   1}
:MTEST:RMODE:SIGMa <level> (see page 341)	:MTEST:RMODE:SIGMa? (see page 341)	<level> ::= from 0.1 to 9.3 in NR3 format
:MTEST:RMODE:TIME <seconds> (see page 342)	:MTEST:RMODE:TIME? (see page 342)	<seconds> ::= from 1 to 86400 in NR3 format
:MTEST:RMODE:WAVEforms <count> (see page 343)	:MTEST:RMODE:WAVEforms? (see page 343)	<count> ::= number of waveforms in NR1 format
:MTEST:SCALE:BIND {{0   OFF}   {1   ON}} (see page 344)	:MTEST:SCALE:BIND? (see page 344)	{0   1}
:MTEST:SCALE:X1 <x1_value> (see page 345)	:MTEST:SCALE:X1? (see page 345)	<x1_value> ::= X1 value in NR3 format
:MTEST:SCALE:XDELta <xdelta_value> (see page 346)	:MTEST:SCALE:XDELta? (see page 346)	<xdelta_value> ::= X delta value in NR3 format
:MTEST:SCALE:Y1 <y1_value> (see page 347)	:MTEST:SCALE:Y1? (see page 347)	<y1_value> ::= Y1 value in NR3 format
:MTEST:SCALE:Y2 <y2_value> (see page 348)	:MTEST:SCALE:Y2? (see page 348)	<y2_value> ::= Y2 value in NR3 format

**Table 54 :MTEST Commands Summary (continued)**

Command	Query	Options and Query Returns
:MTEST:SOURce <source> (see page 349)	:MTEST:SOURce? (see page 349)	<source> ::= {CHANnel<n>   NONE} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
n/a	:MTEST:TITLe? (see page 350)	<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:BIND 0;X1
+200.000E-06;XDEL +400.000E-06;Y1 -3.00000E+00;Y2
+3.00000E+00;:MTES:RMOD FOR;RMOD:TIME +1E+00;WAV 1000;SIGM
+6.0E+00;:MTES:RMOD:FACT:STOP 0;PRIN 0;SAVE 0
```

### Example Code

```
' Mask testing commands example.
' -----
Option Explicit

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

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = myMgr.Open("TCPIP0::130.29.70.228::inst0::INSTR")
    myScope.IO.Clear      ' Clear the interface.
```

```

' Make sure oscilloscope is running.
myScope.WriteString ":RUN"

' Set mask test termination conditions.
myScope.WriteString ":MTEST:RMODe SIGMa"
myScope.WriteString ":MTEST:RMODe?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test termination mode: " + strQueryResult

myScope.WriteString ":MTEST:RMODe:SIGMa 4.2"
myScope.WriteString ":MTEST:RMODe:SIGMa?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test termination 'test sigma': " + _
    FormatNumber(varQueryResult)

' Use auto-mask to create mask.
myScope.WriteString ":MTEST:AMASK:SOURce CHANnel1"
myScope.WriteString ":MTEST:AMASK:SOURce?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask source: " + strQueryResult

myScope.WriteString ":MTEST:AMASK:UNITS DIVisions"
myScope.WriteString ":MTEST:AMASK:UNITS?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test auto-mask units: " + strQueryResult

myScope.WriteString ":MTEST:AMASK:XDELta 0.1"
myScope.WriteString ":MTEST:AMASK:XDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask X delta: " + _
    FormatNumber(varQueryResult)

myScope.WriteString ":MTEST:AMASK:YDELta 0.1"
myScope.WriteString ":MTEST:AMASK:YDELta?"
varQueryResult = myScope.ReadNumber
Debug.Print "Mask test auto-mask Y delta: " + _
    FormatNumber(varQueryResult)

' Enable "Auto Mask Created" event (bit 10, &H400)
myScope.WriteString "*CLS"
myScope.WriteString ":MTEenable " + CStr(CInt("&H400"))

' Create mask.
myScope.WriteString ":MTEST:AMASK:CREate"
Debug.Print "Auto-mask created, mask test automatically enabled."

' Set up timeout variables.
Dim lngTimeout As Long      ' Max millisecs to wait.
Dim lngElapsed As Long
lngTimeout = 60000           ' 60 seconds.

' Wait until mask is created.
lngElapsed = 0
Do While lngElapsed <= lngTimeout
    myScope.WriteString ":OPERegister:CONDITION?"
    varQueryResult = myScope.ReadNumber

```

## 5 Commands by Subsystem

```
' 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:AMASK:CREate**

**N** (see page 658)

**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 320
- "[":MTEST:AMASK:XDELta](#)" on page 326
- "[":MTEST:AMASK:YDELta](#)" on page 327
- "[":MTEST:AMASK:UNITS](#)" on page 325
- "[":MTEST:AMASK:SOURce](#)" on page 324
- "[":MTEST:SOURce](#)" on page 349

**Example Code**

- "[Example Code](#)" on page 320

### :MTEST:AMASK:SOURce

**N** (see page 658)

#### Command Syntax

```
:MTEST:AMASK:SOURce <source>  
<source> ::= CHANnel<n>  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

#### See Also

- "[Introduction to :MTEST Commands](#)" on page 320
- "[":MTEST:AMASK:XDELta](#)" on page 326
- "[":MTEST:AMASK:YDELta](#)" on page 327
- "[":MTEST:AMASK:UNITS](#)" on page 325
- "[":MTEST:SOURce](#)" on page 349

#### Example Code

- "[Example Code](#)" on page 320

**:MTEST:AMASK:UNITS**

**N** (see page 658)

**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  $\Delta X$  and voltage for  $\Delta Y$ .
- 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 320
  - ":MTEST:AMASK:XDELta" on page 326
  - ":MTEST:AMASK:YDELta" on page 327
  - ":CHANnel<n>:UNITS" on page 206
  - ":MTEST:AMASK:SOURce" on page 324
  - ":MTEST:SOURce" on page 349

**Example Code**

- "Example Code" on page 320

### :MTEST:AMASK:XDELta

**N** (see page 658)

**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  $\pm 250$  ms. If the setting for :MTEST:AMASK:UNITS is DIVisions, the same X delta value will set the tolerance to  $\pm 250$  millidivisions, or 1/4 of a division.

**Query Syntax** :MTEST:AMASK:XDELta?

The :MTEST:AMASK:XDELta? query returns the current setting of the  $\Delta X$  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 320

- "[:MTEST:AMASK:UNITS](#)" on page 325
- "[:MTEST:AMASK:YDELta](#)" on page 327
- "[:MTEST:AMASK:SOURce](#)" on page 324
- "[:MTEST:SOURce](#)" on page 349

**Example Code** • "[Example Code](#)" on page 320

**:MTEST:AMASK:YDELta**

**N** (see page 658)

**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  $\pm 250$  mV. If the setting for :MTEST:AMASK:UNITS is DIVisions, the same Y delta value will set the tolerance to  $\pm 250$  millidivisions, or 1/4 of a division.

**Query Syntax**

```
:MTEST:AMASK:YDELta?
```

The :MTEST:AMASK:YDELta? query returns the current setting of the  $\Delta Y$  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 320
- "[:MTEST:AMASK:UNITS](#)" on page 325
- "[:MTEST:AMASK:XDELta](#)" on page 326
- "[:MTEST:AMASK:SOURce](#)" on page 324
- "[:MTEST:SOURce](#)" on page 349

**Example Code**

- "[Example Code](#)" on page 320

## :MTEST:COUNt:FWAVeforms

**N** (see page 658)

**Query Syntax** :MTEST:COUNT:FWAVEforms? [CHANnel<n>]

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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.

**Return Format** <failed><NL>

<failed> ::= number of failed waveforms in NR1 format.

**See Also** • "[Introduction to :MTEST Commands](#)" on page 320

• "[:MTEST:COUNt:WAVEforms](#)" on page 331

• "[:MTEST:COUNt:TIME](#)" on page 330

• "[:MTEST:COUNt:RESet](#)" on page 329

• "[:MTEST:SOURce](#)" on page 349

**Example Code** • "[Example Code](#)" on page 320

**:MTEST:COUNt:RESet**

**N** (see page 658)

**Command Syntax** :MTEST:COUNT:RESET

The :MTEST:COUNt:RESet command resets the mask statistics.

**See Also**

- "[Introduction to :MTEST Commands](#)" on page 320
- "[:MTEST:COUNT:WAVeforms](#)" on page 331
- "[:MTEST:COUNT:FWAVeforms](#)" on page 328
- "[:MTEST:COUNT:TIME](#)" on page 330

## **:MTEST:COUNt:TIME**

**N** (see page 658)

**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 320
  - ":MTEST:COUNT:WAVEforms" on page 331
  - ":MTEST:COUNT:FWAVEforms" on page 328
  - ":MTEST:COUNt:RESet" on page 329

**Example Code**

- "Example Code" on page 320

**:MTEST:COUNt:WAVeforms**

**N** (see page 658)

**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 320
  - "[:MTEST:COUNt:FWAVeforms](#)" on page 328
  - "[:MTEST:COUNt:TIME](#)" on page 330
  - "[:MTEST:COUNt:RESet](#)" on page 329

**Example Code**

- "[Example Code](#)" on page 320

### :MTEST:DATA

**N** (see page 658)

**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 366
- "[:RECall:MASK\[:STARt\]](#)" on page 354

**:MTEST:DElete**

**N** (see page 658)

**Command Syntax** :MTEST:DElete

The :MTEST:DElete command clears the currently loaded mask.

- See Also**
- "Introduction to :MTEST Commands" on page 320
  - ":MTEST:AMASK:CREATE" on page 323

## **:MTEST:ENABLE**

**N** (see page 658)

**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 320

## :MTEST:LOCK

**N** (see page 658)

**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 320
  - "[":MTEST:SOURce](#)" on page 349

### :MTEST:OUTPut

**N** (see page 658)

**Command Syntax** :MTEST:OUTPut <signal>  
<signal> ::= {FAIL | PASS}

The :MTEST:OUTPut command selects the mask test output condition:

- FAIL – the output occurs when there are mask test failures.
- PASS – the output occurs when the mask test passes.

You can place the mask test signal on the rear panel TRIG OUT BNC using the "[:CALibrate:OUTPut](#)" on page 183 command.

**Query Syntax** :MTEST:OUTPut?

The :MTEST:OUTPut? query returns the currently set output signal.

**Return Format** <signal><NL>  
<signal> ::= {FAIL | PASS}

**See Also** • "Introduction to :MTEST Commands" on page 320  
• "[:CALibrate:OUTPut](#)" on page 183

**:MTEST:RMODE**

**N** (see page 658)

**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 341 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 342 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 343 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 320
- "[:MTEST:RMODE:SIGMA](#)" on page 341
- "[:MTEST:RMODE:TIME](#)" on page 342
- "[:MTEST:RMODE:WAVEforms](#)" on page 343

**Example Code**

- "[Example Code](#)" on page 320

### :MTEST:RMODE:FACTion:PRINt

**N** (see page 658)

**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 "[:HARDcopy Commands](#)" on page 245 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 320
- "[:MTEST:RMODE:FACTion:SAVE](#)" on page 339
- "[:MTEST:RMODE:FACTion:STOP](#)" on page 340

**:MTEST:RMODE:FACTion:SAVE**

**N** (see page 658)

**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 "[:SAVE Commands](#)" on page 357 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 320  
 • "[:MTEST:RMODE:FACTion:PRINT](#)" on page 338  
 • "[:MTEST:RMODE:FACTion:STOP](#)" on page 340

## **:MTEST:RMODE:FACTion:STOP**

**N** (see page 658)

**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 320  
• ":MTEST:RMODE:FACTion:PRINT" on page 338  
• ":MTEST:RMODE:FACTion:SAVE" on page 339

## :MTEST:RMODE:SIGMA

**N** (see page 658)

**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 320
  - "[:MTEST:RMODE](#)" on page 337

**Example Code**

- "[Example Code](#)" on page 320

### :MTEST:RMODE:TIME

**N** (see page 658)

**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 320
- "[":MTEST:RMODE](#)" on page 337

## :MTEST:RMODE:WAveforms

**N** (see page 658)

### 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 320
- ":MTEST:RMODE" on page 337

## :MTEST:SCALe:BIND

**N** (see page 658)

**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 320
- "[":MTEST:SCALe:X1"](#) on page 345
- "[":MTEST:SCALe:XDELta"](#) on page 346
- "[":MTEST:SCALe:Y1"](#) on page 347
- "[":MTEST:SCALe:Y2"](#) on page 348

**:MTEST:SCALe:X1**

**N** (see page 658)

**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 320
- "[:MTEST:SCALe:BIND](#)" on page 344
- "[:MTEST:SCALe:XDELta](#)" on page 346
- "[:MTEST:SCALe:Y1](#)" on page 347
- "[:MTEST:SCALe:Y2](#)" on page 348

### :MTEST:SCALE:XDELta

**N** (see page 658)

**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  $\Delta X$ , redefining  $\Delta X$  also moves those vertices to stay in the same locations with respect to X1 and  $\Delta X$ . 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  $\Delta X$ .

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  $\Delta X$  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  $\Delta X$ .

**Return Format** <xdelta\_value><NL>

<xdelta\_value> ::= X delta value in NR3 format

- See Also**
- "[Introduction to :MTEST Commands](#)" on page 320
  - "[:MTEST:SCALE:BIND](#)" on page 344
  - "[:MTEST:SCALE:X1](#)" on page 345
  - "[:MTEST:SCALE:Y1](#)" on page 347
  - "[:MTEST:SCALE:Y2](#)" on page 348

**:MTEST:SCALe:Y1**

**N** (see page 658)

**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 320
  - "[:MTEST:SCALe:BIND](#)" on page 344
  - "[:MTEST:SCALe:X1](#)" on page 345
  - "[:MTEST:SCALe:XDELta](#)" on page 346
  - "[:MTEST:SCALe:Y2](#)" on page 348

### :MTEST:SCALe:Y2

**N** (see page 658)

**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 320
  - "[:MTEST:SCALe:BIND](#)" on page 344
  - "[:MTEST:SCALe:X1](#)" on page 345
  - "[:MTEST:SCALe:XDELta](#)" on page 346
  - "[:MTEST:SCALe:Y1](#)" on page 347

**:MTEST:SOURce**

**N** (see page 658)

**Command Syntax** :MTEST:SOURce <source>

```
<source> ::= CHANnel<n>
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

- See Also**
- "Introduction to :MTEST Commands" on page 320
  - ":MTEST:AMASK:SOURce" on page 324

## **:MTEST:TITLE**



(see page 658)

**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 320

## :RECall Commands

Recall previously saved oscilloscope setups and traces. See "Introduction to :RECall Commands" on page 351.

**Table 55** :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILEname <base_name> (see page 352)	:RECall:FILEname? (see <a href="#">page 352</a> )	<base_name> ::= quoted ASCII string
:RECall:IMAGE[:START] [<file_spec>] (see page 353)	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see page 354)	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 355)	:RECall:PWD? (see <a href="#">page 355</a> )	<path_name> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 356)	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string

**Introduction to :RECall Commands** The :RECall subsystem provides commands to recall previously saved oscilloscope setups and traces.

### 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:FILEname

**N** (see page 658)

**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 351
  - "[":RECall:IMAGe\[:STARt\]](#)" on page 353
  - "[":RECall:SETup\[:STARt\]](#)" on page 356
  - "[":SAVE:FILEname](#)" on page 359

**:RECall:IMAGe[:STARt]**

**N** (see page 658)

**Command Syntax** :RECall:IMAGe[: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:IMAGe[:STARt] command recalls a trace (TIFF) image.

**NOTE**

If a file extension is provided as part of a specified <file\_name>, it must be ".tif".

**See Also**

- "[Introduction to :RECall Commands](#)" on page 351
- "[":RECall:FILEname](#)" on page 352
- "[":SAVE:IMAGe\[:STARt\]](#)" on page 360

### :RECall:MASK[:STARt]

**N** (see page 658)

**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 351
- "[:RECall:FILEname](#)" on page 352
- "[:SAVE:MASK\[:STARt\]](#)" on page 366
- "[:MTESt:DATA](#)" on page 332

**:RECall:PWD**

(see page 658)

**Command Syntax** :RECall:PWD <path\_name>

&lt;path\_name&gt; ::= 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>

&lt;path\_name&gt; ::= quoted ASCII string

**See Also** • "[Introduction to :RECall Commands](#)" on page 351  
• "[":SAVE:PWD"](#) on page 367

### :RECall:SETup[:STARt]

**N** (see page 658)

**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 351
- "[":RECall:FILEname](#)" on page 352
- "[":SAVE:SETup\[:STARt\]](#)" on page 368

## :SAVE Commands

Save oscilloscope setups and traces, screen images, and data. See "Introduction to :SAVE Commands" on page 358.

**Table 56** :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILEname <base_name> (see page 359)	:SAVE:FILEname? (see page 359)	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_spec>] (see page 360)	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
n/a	:SAVE:IMAGE:AREA? (see page 361)	<area> ::= {GRAT   SCR}
:SAVE:IMAGE:FACTors {0   OFF}   {1   ON} (see page 362)	:SAVE:IMAGE:FACTors? (see page 362)	{0   1}
:SAVE:IMAGE:FORMAT <format> (see page 363)	:SAVE:IMAGE:FORMAT? (see page 363)	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGE:INKSaver {0   OFF}   {1   ON} (see page 364)	:SAVE:IMAGE:INKSaver? (see page 364)	{0   1}
:SAVE:IMAGE:PALETTE <palette> (see page 365)	:SAVE:IMAGE:PALETTE? (see page 365)	<palette> ::= {COLOR   GRAYscale   MONochrome}
:SAVE:MASK[:START] [<file_spec>] (see page 366)	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-3; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:PWD <path_name> (see page 367)	:SAVE:PWD? (see page 367)	<path_name> ::= quoted ASCII string

**Table 56** :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETUp[:START] [<file_spec>] (see page 368)	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 369)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMAT <format> (see page 370)	:SAVE:WAVeform:FORMAT ? (see page 370)	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVeform:LENGTH <length> (see page 371)	:SAVE:WAVeform:LENGTH ? (see page 371)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVeform:SEGMen ted <option> (see page 372)	:SAVE:WAVeform:SEGMen ted? (see page 372)	<option> ::= {ALL   CURRent}

**Introduction to :SAVE Commands** The :SAVE subsystem provides commands to save oscilloscope setups and traces, 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:FILEname

**N** (see page 658)

**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 358
  - "[":SAVE:IMAGe\[:STARt\]](#)" on page 360
  - "[":SAVE:SETup\[:STARt\]](#)" on page 368
  - "[":SAVE:WAveform\[:STARt\]](#)" on page 369
  - "[":SAVE:PWD](#)" on page 367
  - "[":RECall:FILEname](#)" on page 352

### :SAVE:IMAGe[:STARt]

**N** (see page 658)

**Command Syntax** :SAVE:IMAGe[: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:IMAGe[:STARt] command saves an image.

**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.

**NOTE**

When the <internal\_loc> option is used, the :SAVE:IMAGe:FORMAT will be changed to TIFF.

**See Also**

- "[Introduction to :SAVE Commands](#)" on page 358
- "[":SAVE:IMAGe:AREA](#)" on page 361
- "[":SAVE:IMAGe:FACTors](#)" on page 362
- "[":SAVE:IMAGe:FORMAT](#)" on page 363
- "[":SAVE:IMAGe:INKSaver](#)" on page 364
- "[":SAVE:IMAGe:PAlette](#)" on page 365
- "[":SAVE:FILEname](#)" on page 359
- "[":RECall:IMAGe\[:STARt\]](#)" on page 353

## :SAVE:IMAGe:AREA

**N** (see page 658)

**Query Syntax** :SAVE:IMAGe:AREA?

The :SAVE:IMAGe:AREA? query returns the selected image area. If the :SAVE:IMAGe:FORMAT is TIFF, the area is GRAT (graticule). Otherwise, it is SCR (screen).

**Return Format** <area><NL>

<area> ::= {GRAT | SCR}

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 358
  - "[":SAVE:IMAGe\[:START\]](#)" on page 360
  - "[":SAVE:IMAGe:FACTors](#)" on page 362
  - "[":SAVE:IMAGe:FORMAT](#)" on page 363
  - "[":SAVE:IMAGe:INKSaver](#)" on page 364
  - "[":SAVE:IMAGe:PAlette](#)" on page 365

## :SAVE:IMAGe:FACTors

**N** (see page 658)

**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 358

- "[":SAVE:IMAGe\[:STARt\]](#)" on page 360
- "[":SAVE:IMAGe:AREA](#)" on page 361
- "[":SAVE:IMAGe:FORMAT](#)" on page 363
- "[":SAVE:IMAGe:INKSaver](#)" on page 364
- "[":SAVE:IMAGe:PALETTE](#)" on page 365

## :SAVE:IMAGe:FORMat

**N** (see page 658)

- Command Syntax** :SAVE:IMAGe:FORMat <format>  
 <format> ::= {TIFF | {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> ::= {TIFF | 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 358
  - "[":SAVE:IMAGe\[:STARt\]](#)" on page 360
  - "[":SAVE:IMAGe:AREA](#)" on page 361
  - "[":SAVE:IMAGe:FACTors](#)" on page 362
  - "[":SAVE:IMAGe:INKSaver](#)" on page 364
  - "[":SAVE:IMAGe:PAlette](#)" on page 365
  - "[":SAVE:WAveform:FORMat](#)" on page 370

### :SAVE:IMAGe:INKSaver

**N** (see page 658)

**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 358  
                  • "[":SAVE:IMAGe\[:STARt\]](#)" on page 360  
                  • "[":SAVE:IMAGe:AREA](#)" on page 361  
                  • "[":SAVE:IMAGe:FACTors](#)" on page 362  
                  • "[":SAVE:IMAGe:FORMAT](#)" on page 363  
                  • "[":SAVE:IMAGe:PAlette](#)" on page 365

## :SAVE:IMAGe:PAlette

**N** (see page 658)

**Command Syntax** :SAVE:IMAGe:PAlette <palette>

<palette> ::= {COLOR | GRAYscale | MONochrome}

The :SAVE:IMAGe:PAlette command sets the image palette color.

**NOTE**

MONochrome is the only valid choice when the :SAVE:IMAGe:FORMAT is TIFF. COLOR and GRAYscale are the only valid choices when the format is not TIFF.

**Query Syntax** :SAVE:IMAGe:PAlette?

The :SAVE:IMAGe:PAlette? query returns the selected image palette color.

**Return Format** <palette><NL>

<palette> ::= {COL | GRAY | MON}

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 358
  - "[":SAVE:IMAGe\[:STARt\]](#)" on page 360
  - "[":SAVE:IMAGe:AREA](#)" on page 361
  - "[":SAVE:IMAGe:FACTors](#)" on page 362
  - "[":SAVE:IMAGe:FORMAT](#)" on page 363
  - "[":SAVE:IMAGe:INKSaver](#)" on page 364

### :SAVE:MASK[:STARt]

**N** (see page 658)

**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 358
- "[:SAVE:FILEname](#)" on page 359
- "[:RECall:MASK\[:STARt\]](#)" on page 354
- "[:MTESt:DATA](#)" on page 332

**:SAVE:PWD**

**N** (see page 658)

**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 358
- "[":SAVE:FILENAME](#)" on page 359
- "[":RECALL:PWD](#)" on page 355

### :SAVE:SETUp[:STARt]

**N** (see page 658)

**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 358
- "[":SAVE:FILEname](#)" on page 359
- "[":RECall:SETUp\[:STARt\]](#)" on page 356

**:SAVE:WAVeform[:STARt]****N** (see page 658)**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**

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 358
- "[":SAVE:WAVeform:FORMAT](#)" on page 370
- "[":SAVE:WAVeform:LENGTH](#)" on page 371
- "[":SAVE:FILEname](#)" on page 359
- "[":RECall:SETup\[:STARt\]](#)" on page 356

## :SAVE:WAVeform:FORMat

**N** (see page 658)

**Command Syntax** :SAVE:WAVeform:FORMat <format>

```
<format> ::= {ALB | ASCiixy | CSV | BINary}
```

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ALB – creates an Agilent module binary format file. These files can be viewed offline by the *Agilent Logic Analyzer* application software. The proper file extension for this format is ".alb".
- ASCiixy – creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV – creates one comma-separated value file that contains information for all analog channels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINary – creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

**Query Syntax** :SAVE:WAVeform:FORMat?

The :SAVE:WAVeform:FORMat? query returns the selected waveform data format type.

**Return Format** <format><NL>

```
<format> ::= {ALB | ASC | CSV | BIN | NONE}
```

When NONE is returned, it indicates that an image file format is currently selected.

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 358
  - "[":SAVE:WAVeform\[:START\]](#)" on page 369
  - "[":SAVE:WAVeform:LENGTH](#)" on page 371
  - "[":SAVE:IMAGE:FORMAT](#)" on page 363

**:SAVE:WAVeform:LENGth**

**N** (see page 658)

**Command Syntax** :SAVE:WAVeform:LENGth <length>

<length> ::= 100 to max. length; an integer in NR1 format

The :SAVE:WAVeform:LENGth command sets the waveform data length (that is, the number of points saved).

**Query Syntax** :SAVE:WAVeform:LENGth?

The :SAVE:WAVeform:LENGth? query returns the specified waveform data length.

**Return Format** <length><NL>

<length> ::= 100 to max. length; an integer in NR1 format

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 358
  - "[":SAVE:WAVeform\[:START\]](#)" on page 369
  - "[":WAVeform:POINTs](#)" on page 512
  - "[":SAVE:WAVeform:FORMAT](#)" on page 370

## **:SAVE:WAveform:SEGmented**

**N** (see page 658)

**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 358
- "[":SAVE:WAveform\[:START\]" on page 369](#)
- "[":SAVE:WAveform:FORMAT" on page 370](#)
- "[":SAVE:WAveform:LENGTH" on page 371](#)

## :SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See "Introduction to :SBUS Commands" on page 374.

**Table 57** :SBUS Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS:CAN:COUNT:ERRor? (see <a href="#">page 375</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:OVERload? (see <a href="#">page 376</a> )	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see <a href="#">page 377</a> )	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal? (see <a href="#">page 378</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILization? (see <a href="#">page 379</a> )	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 380</a> )	:SBUS:DISPlay? (see <a href="#">page 380</a> )	{0   1}
:SBUS:IIC:ASIZE <size> (see <a href="#">page 381</a> )	:SBUS:IIC:ASIZE? (see <a href="#">page 381</a> )	<size> ::= {BIT7   BIT8}
:SBUS:LIN:PARity {{0   OFF}   {1   ON}} (see <a href="#">page 382</a> )	:SBUS:LIN:PARity? (see <a href="#">page 382</a> )	{0   1}
:SBUS:MODE <mode> (see <a href="#">page 383</a> )	:SBUS:MODE? (see <a href="#">page 383</a> )	<mode> ::= {IIC   SPI   CAN   LIN   FLEXray   UART}
:SBUS:SPI:WIDTh <word_width> (see <a href="#">page 384</a> )	:SBUS:SPI:WIDTh? (see <a href="#">page 384</a> )	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see <a href="#">page 385</a> )	:SBUS:UART:BASE? (see <a href="#">page 385</a> )	<base> ::= {ASCII   BINARY   HEX}
n/a	:SBUS:UART:COUNT:ERRor? (see <a href="#">page 386</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESe t (see <a href="#">page 387</a> )	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFRames? (see <a href="#">page 388</a> )	<frame_count> ::= integer in NR1 format

**Table 57** :SBUS Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:SBUS:UART:COUNT:TXFR ames? (see <a href="#">page 389</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see <a href="#">page 390</a> )	:SBUS:UART:FRAMing? (see <a href="#">page 390</a> )	<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

**Introduction to :SBUS Commands** The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

**NOTE** These commands are only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

### Reporting the Setup

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

### Return Format

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

```
:SBUS:DISP 0;MODE IIC
```

**:SBUS:CAN:COUNt:ERRor**

**N** (see page 658)

**Query Syntax** :SBUS:CAN:COUNt:ERRor?

Returns the error frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on page 617

**See Also** • "[:SBUS:CAN:COUNt:RESET](#)" on page 377  
• "[Introduction to :SBUS Commands](#)" on page 374  
• "[:SBUS:MODE](#)" on page 383  
• "[:TRIGger:CAN Commands](#)" on page 422

**:SBUS:CAN:COUNt:OVERload**

**N** (see page 658)

**Query Syntax** :SBUS: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 617

**See Also** • "[:SBUS:CAN:COUNt:RESET](#)" on page 377  
• "[Introduction to :SBUS Commands](#)" on page 374  
• "[:SBUS:MODE](#)" on page 383  
• "[:TRIGGER:CAN Commands](#)" on page 422

**:SBUS:CAN:COUNt:RESet****N** (see page 658)**Command Syntax** :SBUS:CAN:COUNt:RESet

Resets the frame counters.

**Errors** • "-241, Hardware missing" on page 617**See Also** • ":SBUS:CAN:COUNt:ERRor" on page 375  
• ":SBUS:CAN:COUNt:OVERload" on page 376  
• ":SBUS:CAN:COUNt:TOTal" on page 378  
• ":SBUS:CAN:COUNt:UTILization" on page 379  
• "Introduction to :SBUS Commands" on page 374  
• ":SBUS:MODE" on page 383  
• ":TRIGger:CAN Commands" on page 422

**:SBUS:CAN:COUNt:TOTal**

**N** (see page 658)

**Query Syntax** :SBUS: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 617

**See Also** • ":SBUS:CAN:COUNt:RESET" on page 377  
• "Introduction to :SBUS Commands" on page 374  
• ":SBUS:MODE" on page 383  
• ":TRIGger:CAN Commands" on page 422

**:SBUS:CAN:COUNt:UTILization**

**N** (see page 658)

**Query Syntax** :SBUS:CAN:COUNT:UTILization?

Returns the percent utilization.

**Return Format** <percent><NL>

<percent> ::= floating-point in NR3 format

**Errors** • "-241, Hardware missing" on page 617

**See Also** • "[:SBUS:CAN:COUNt:RESET](#)" on page 377  
• "["Introduction to :SBUS Commands"](#) on page 374  
• "[:SBUS:MODE](#)" on page 383  
• "[:TRIGger:CAN Commands](#)" on page 422

### :SBUS:DISPlay

**N** (see page 658)

**Command Syntax** :SBUS:DISPLAY <display>

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

The :SBUS:DISPLAY command turns displaying of the serial decode bus on or off.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax** :SBUS:DISPLAY?

The :SBUS:DISPLAY? query returns the current display setting of the serial decode bus.

**Return Format** <display><NL>

<display> ::= {0 | 1}

**Errors** • "-241, Hardware missing" on page 617

**See Also** • "Introduction to :SBUS Commands" on page 374  
• ":CHANnel<n>:DISPLAY" on page 194  
• ":VIEW" on page 162  
• ":BLANK" on page 130  
• ":STATus" on page 159

**:SBUS:IIC:ASIZE****N** (see page 658)**Command Syntax** :SBUS:IIC:ASIZE <size>  
<size> ::= {BIT7 | BIT8}

The :SBUS: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.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

**Query Syntax** :SBUS:IIC:ASIZE?

The :SBUS:IIC:ASIZE? query returns the current IIC address width setting.

**Return Format** <mode><NL>

<mode> ::= {BIT7 | BIT8}

**Errors** • "-241, Hardware missing" on page 617**See Also** • "Introduction to :SBUS Commands" on page 374  
• ":TRIGger:IIC Commands" on page 453

## :SBUS:LIN:PARity

**N** (see page 658)

**Command Syntax** :SBUS:LIN:PARity <display>

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

The :SBUS: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.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax**

:SBUS:LIN:PARity?

The :SBUS: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 617

**See Also** • "Introduction to :SBUS Commands" on page 374  
• ":TRIGger:LIN Commands" on page 462

**:SBUS:MODE**

**N** (see page 658)

**Command Syntax**

```
:SBUS:MODE <mode>
<mode> ::= {IIC | SPI | CAN | LIN | UART}
```

The :SBUS:MODE command determines the decode mode for the serial bus.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax**

```
:SBUS:MODE?
```

The :SBUS:MODE? query returns the current serial bus decode mode setting.

**Return Format**

```
<mode><NL>
<mode> ::= {IIC | SPI | CAN | LIN | UART | NONE}
```

- "-241, Hardware missing" on page 617

**See Also**

- "[Introduction to :SBUS Commands](#)" on page 374
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:IIC Commands](#)" on page 453
- "[:TRIGger:SPI Commands](#)" on page 470
- "[:TRIGger:CAN Commands](#)" on page 422
- "[:TRIGger:LIN Commands](#)" on page 462
- "[:TRIGger:UART Commands](#)" on page 485

### :SBUS:SPI:WIDTh

**N** (see page 658)

**Command Syntax** :SBUS:SPI:WIDTh <word\_width>

<word\_width> ::= integer 4-16 in NR1 format

The :SBUS:SPI:WIDTh command determines the number of bits in a word of data for SPI.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

**Query Syntax** :SBUS:SPI:WIDTh?

The :SBUS: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 617

**See Also** • "Introduction to :SBUS Commands" on page 374  
• ":SBUS:MODE" on page 383  
• ":TRIGger:SPI Commands" on page 470

## :SBUS:UART:BASE

**N** (see page 658)

**Command Syntax** :SBUS:UART:BASE <base>

<base> ::= {ASCii | BINary | HEX}

The :SBUS:UART:BASE command determines the base to use for the UART decode display.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Query Syntax** :SBUS:UART:BASE?

The :SBUS:UART:BASE? query returns the current UART decode base setting.

**Return Format** <base><NL>

<base> ::= {ASCii | BINary | HEX}

- Errors**
- "-241, Hardware missing" on page 617

- See Also**
- "Introduction to :SBUS Commands" on page 374
  - ":TRIGger:UART Commands" on page 485

## **:SBUS:UART:COUNt:ERRor**

**N** (see page 658)

**Query Syntax** :SBUS:UART:COUNT:ERRor?

Returns the UART error frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on page 617

**See Also** • ":SBUS:UART:COUNt:RESet" on page 387  
• "Introduction to :SBUS Commands" on page 374  
• ":SBUS:MODE" on page 383  
• ":TRIGger:UART Commands" on page 485

**:SBUS:UART:COUNt:RESet****N** (see page 658)**Command Syntax** :SBUS:UART:COUNT:RESET

Resets the UART frame counters.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

- Errors**
- "
- [-241, Hardware missing](#)
- " on page 617

- See Also**
- "
- [:SBUS:UART:COUNt:ERRor](#)
- " on page 386
- 
- "
- [:SBUS:UART:COUNt:RXFRames](#)
- " on page 388
- 
- "
- [:SBUS:UART:COUNt:TXFRames](#)
- " on page 389
- 
- "
- [Introduction to :SBUS Commands](#)
- " on page 374
- 
- "
- [:SBUS:MODE](#)
- " on page 383
- 
- "
- [:TRIGger:UART Commands](#)
- " on page 485

## :SBUS:UART:COUNt:RXFRAMES

**N** (see page 658)

**Query Syntax** :SBUS:UART:COUNT:RXFRAMES?

Returns the UART Rx frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on page 617

**See Also** • ":SBUS:UART:COUNt:RESet" on page 387  
• "Introduction to :SBUS Commands" on page 374  
• ":SBUS:MODE" on page 383  
• ":TRIGger:UART Commands" on page 485

**:SBUS:UART:COUNt:TXFRames****N** (see page 658)**Query Syntax** :SBUS:UART:COUNt:TXFRames?

Returns the UART Tx frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

&lt;frame\_count&gt; ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on page 617**See Also** • "[:SBUS:UART:COUNt:RESet](#)" on page 387  
• "Introduction to [:SBUS Commands](#)" on page 374  
• "[:SBUS:MODE](#)" on page 383  
• "[:TRIGger:UART Commands](#)" on page 485

## :SBUS:UART:FRAMing

**N** (see page 658)

**Command Syntax**

```
:SBUS: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:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Query Syntax**

```
:SBUS:UART:FRAMing?
```

The :SBUS: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 617

**See Also**

- "Introduction to :SBUS Commands" on page 374
- ":TRIGger:UART Commands" on page 485

## :SYSTem Commands

Control basic system functions of the oscilloscope. See "[Introduction to :SYSTem Commands](#)" on page 391.

**Table 58** :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see <a href="#">page 392</a> )	:SYSTem:DATE? (see <a href="#">page 392</a> )	<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 <a href="#">page 393</a> )	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see <a href="#">page 394</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 615</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 395</a> )	:SYSTem:LOCK? (see <a href="#">page 395</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PROTection:LOCK <value> (see <a href="#">page 396</a> )	:SYSTem:PROTection:LOCK? (see <a href="#">page 396</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 397</a> )	:SYSTem:SETup? (see <a href="#">page 397</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 399</a> )	:SYSTem:TIME? (see <a href="#">page 399</a> )	<time> ::= hours,minutes,seconds in NR1 format

**Introduction to :SYSTem Commands** SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

## **:SYSTem:DATE**

**N** (see page 658)

**Command Syntax** :SYSTem:DATE <date>

```
<date> ::= <year>,<month>,<day>
<year> ::= 4-digit year in NR1 format
<month> ::= {1,...,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNE
              | JULY | AUGust | SEPtember | OCTober | NOVember | DECember}
<day> ::= {1,...,31}
```

The :SYSTem:DATE command sets the date. Validity checking is performed to ensure that the date is valid.

**Query Syntax** :SYSTem:DATE?

The SYSTem:DATE? query returns the date.

**Return Format** <year>,<month>,<day><NL>

- See Also**
- "Introduction to :SYSTem Commands" on page 391
  - ":SYSTem:TIME" on page 399

## :SYSTem:DSP

**N** (see page 658)

**Command Syntax** :SYSTem:DSP <string>

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

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

**See Also** • "Introduction to :SYSTem Commands" on page 391

## **:SYSTem:ERRor**



(see page 658)

### **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 "[Error Messages](#)" on page 615.

### **See Also**

- "[Introduction to :SYSTem Commands](#)" on page 391
- "[\\*ESR \(Standard Event Status Register\)](#)" on page 104
- "[\\*CLS \(Clear Status\)](#)" on page 101

## :SYSTem:LOCK

**N** (see page 658)

**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 391

## **:SYSTem:PROTection:LOCK**

**N** (see page 658)

**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 391

## :SYSTem:SEtup

**C** (see page 658)

**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 391
  - "[\\*LRN \(Learn Device Setup\)](#)" on page 107

**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,
' #800002204<setup string><NL>
' where the setup string is 2204 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.
```

## 5 Commands by Subsystem

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"  
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

**:SYSTem:TIME**

**N** (see page 658)

**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 391
  - ":SYSTem:DATE" on page 392

## :TIMEbase Commands

Control all horizontal sweep functions. See "Introduction to :TIMEbase Commands" on page 400.

**Table 59** :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see page 402)	:TIMEbase:MODE? (see page 402)	<value> ::= {MAIN   WINDOW   XY   ROLL}
:TIMEbase:POSITION <pos> (see page 403)	:TIMEbase:POSITION? (see page 403)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMEbase:RANGE <range_value> (see page 404)	:TIMEbase:RANGE? (see page 404)	<range_value> ::= 10 ns through 500 s in NR3 format
:TIMEbase:REFERENCE {LEFT   CENTER   RIGHT} (see page 405)	:TIMEbase:REFERENCE? (see page 405)	<return_value> ::= {LEFT   CENTER   RIGHT}
:TIMEbase:SCALE <scale_value> (see page 406)	:TIMEbase:SCALE? (see page 406)	<scale_value> ::= scale value in seconds in NR3 format
:TIMEbase:VERNier {{0   OFF}   {1   ON}} (see page 407)	:TIMEbase:VERNier? (see page 407)	{0   1}
:TIMEbase:WINDOW:POSITION <pos> (see page 408)	:TIMEbase:WINDOW:POSITION? (see page 408)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMEbase:WINDOW:RANGE <range_value> (see page 409)	:TIMEbase:WINDOW:RANGE? (see page 409)	<range_value> ::= range value in seconds in NR3 format for the zoomed window
:TIMEbase:WINDOW:SCALE <scale_value> (see page 410)	:TIMEbase:WINDOW:SCALE? (see page 410)	<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 658)

#### Command Syntax

```
:TIMEbase:MODE <value>  
<value> ::= {MAIN | WINDow | XY | ROLL}
```

The :TIMEbase:MODE command sets the current time base. There are four time base modes:

- MAIN – The normal time base mode is the main time base. It is the default time base mode after the \*RST (Reset) command.
- WINDow – In the WINDow (zoomed or delayed) time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.
- XY – In the XY mode, the :TIMEbase:RANGE, :TIMEbase:POSITION, and :TIMEbase:REFERENCE commands are not available. No measurements are available in this mode.
- ROLL – In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMEbase:REFERENCE selection changes to RIGHt.

#### NOTE

If a :DIGItize command is executed when the :TIMEbase:MODE is not MAIN, the :TIMEbase:MODE is set to MAIN.

#### 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 400
- "[\\*RST \(Reset\)](#)" on page 111
- "[:TIMEbase:RANGE](#)" on page 404
- "[:TIMEbase:POSITION](#)" on page 403
- "[:TIMEbase:REFERENCE](#)" on page 405

#### 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"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

**:TIMEbase:POSITION**

**C** (see page 658)

**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 400
  - "[:TIMEbase:REFERENCE](#)" on page 405
  - "[:TIMEbase:RANGE](#)" on page 404
  - "[:TIMEbase:SCALE](#)" on page 406
  - "[:TIMEbase:WINDOW:POSITION](#)" on page 408
  - "[:TIMEbase:DELay](#)" on page 609

## :TIMEbase:RANGE



(see page 658)

**Command Syntax** :TIMEbase:RANGE <range\_value>

<range\_value> ::= 10 ns through 500 s 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> ::= 10 ns through 500 s in NR3 format

- See Also**
- "[Introduction to :TIMEbase Commands](#)" on page 400
  - "[:TIMEbase:MODE](#)" on page 402
  - "[:TIMEbase:SCALe](#)" on page 406
  - "[:TIMEbase:WINDow:RANGE](#)" on page 409

**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.
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

**:TIMEbase:REFerence**

**C** (see page 658)

**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 400
  - ":TIMEbase:MODE" on page 402

**Example Code**

```
' TIME_REFERENCE - Possible values are LEFT and CENTER.
'   - LEFT sets the display reference on time division from the left.
'   - CENTER sets the display reference to the center of the screen.
myScope.WriteString ":TIMEBASE:REFERENCE CENTER" ' Set reference to center.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

### :TIMEbase:SCALe

**N** (see page 658)

**Command Syntax** :TIMEbase:SCALe <scale\_value>

<scale\_value> ::= 1 ns through 50 s in NR3 format

The :TIMEbase:SCALe command sets the horizontal scale or units per division for the main window.

**Query Syntax** :TIMEbase:SCALe?

The :TIMEbase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

**Return Format** <scale\_value><NL>

<scale\_value> ::= 1 ns through 50 s in NR3 format

- See Also**
- "[Introduction to :TIMEbase Commands](#)" on page 400
  - "[:TIMEbase:RANGE](#)" on page 404
  - "[:TIMEbase:WINDOW:SCALe](#)" on page 410
  - "[:TIMEbase:WINDOW:RANGE](#)" on page 409

**:TIMEbase:VERNier**

**N** (see page 658)

**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 400

### :TIMEbase:WINDOW:POSition



(see page 658)

#### 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 400
- "[:TIMEbase:MODE](#)" on page 402
- "[:TIMEbase:POSition](#)" on page 403
- "[:TIMEbase:RANGE](#)" on page 404
- "[:TIMEbase:SCALE](#)" on page 406
- "[:TIMEbase:WINDOW:RANGE](#)" on page 409
- "[:TIMEbase:WINDOW:SCALE](#)" on page 410

**:TIMEbase:WINDOW:RANGE**

(see page 658)

**Command Syntax** :TIMEbase:WINDOW:RANGE <range value>

&lt;range value&gt; ::= 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>

&lt;value&gt; ::= range value in seconds

- See Also**
- "Introduction to :TIMEbase Commands" on page 400
  - ":TIMEbase:RANGE" on page 404
  - ":TIMEbase:POSITION" on page 403
  - ":TIMEbase:SCALE" on page 406

### :TIMEbase:WINDOW:SCALE

**N** (see page 658)

**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 400
  - ":TIMEbase:RANGE" on page 404
  - ":TIMEbase:POSITION" on page 403
  - ":TIMEbase:SCALE" on page 406
  - ":TIMEbase:WINDOW:RANGE" on page 409

## :TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[General :TRIGger Commands](#)" on page 414
- "[:TRIGger:CAN Commands](#)" on page 422
- "[:TRIGger:DURation Commands](#)" on page 433
- "[:TRIGger\[:EDGE\] Commands](#)" on page 439
- "[:TRIGger:GLITch Commands](#)" on page 445 (Pulse Width trigger)
- "[:TRIGger:IIC Commands](#)" on page 453
- "[:TRIGger:LIN Commands](#)" on page 462
- "[:TRIGger:SPI Commands](#)" on page 470
- "[:TRIGger:TV Commands](#)" on page 479
- "[:TRIGger:UART Commands](#)" on page 485

**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 421) 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 417).

- **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. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on CAN data and identifier patterns, set the bit sample point, and have the module send an acknowledge to the bus when it receives a valid message.

### NOTE

The CAN and LIN serial decode option (Option ASM) replaces the functionality that was available with the N2758A CAN trigger module for the 54620/54640 Series oscilloscopes.

- **Edge triggering** identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Pulse width triggering** (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering** identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels.
- **Duration triggering** lets you define a pattern, then trigger on a specified time duration.
- **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. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on Frame IDs.
- **SPI (Serial Peripheral Interface) triggering** consists of connecting the oscilloscope to a clock, data, 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 32 bits long.
- **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  $\frac{1}{2}$  division of sync amplitude with any analog channel as the trigger source.
- **UART/RS-232 triggering** (with Option 232) lets you trigger on RS-232 serial data.

### 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
```

## General :TRIGger Commands

**Table 60** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}   {1   ON}} (see <a href="#">page 415</a> )	:TRIGger:HFReject? (see <a href="#">page 415</a> )	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see <a href="#">page 416</a> )	:TRIGger:HOLDoff? (see <a href="#">page 416</a> )	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:MODE <mode> (see <a href="#">page 417</a> )	:TRIGger:MODE? (see <a href="#">page 417</a> )	<mode> ::= {EDGE   GLITch   PATtern   DURation   TV} <return_value> ::= {<mode>   <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREJect {{0   OFF}   {1   ON}} (see <a href="#">page 418</a> )	:TRIGger:NREJect? (see <a href="#">page 418</a> )	{0   1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>] (see <a href="#">page 419</a> )	:TRIGger:PATTern? (see <a href="#">page 419</a> )	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnn"; n ::= {0,...,9   A,...,F} (# bits = # channels) <edge source> ::= {CHANnel<n>   EXTernal   NONE} <edge> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see <a href="#">page 421</a> )	:TRIGger:SWEep? (see <a href="#">page 421</a> )	<sweep> ::= {AUTO   NORMAL}

**:TRIGger:HFReject**

(see page 658)

**Command Syntax** :TRIGger:HFReject <value>

&lt;value&gt; ::= {{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>

&lt;value&gt; ::= {0 | 1}

- See Also**
- "Introduction to :TRIGger Commands" on page 411
  - ":TRIGGER[:EDGE]:REJect" on page 442

### :TRIGger:HOLDOff



(see page 658)

**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 411

**:TRIGger:MODE**

(see page 658)

**Command Syntax** :TRIGger:MODE <mode>

```
1234567890123456789012345678901234567890123456789012345678901234567890
<mode> ::= {EDGE | GLITch | PATtern | CAN | DURation | IIC | LIN | SPI
           | TV | USB | FLEXray | UART}
```

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> ::= {NONE | EDGE | GLITch | PATtern | CAN | DURation | IIC
           | LIN | SPI | TV | USB | FLEXray | UART}
```

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[":TRIGger:SWEep](#)" on page 421
  - "[":TIMEbase:MODE](#)" on page 402

**Example Code**

```
' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATtern, CAN,
' DURation, IIC, LIN, SPI, TV, USB, FLEXray, or UART.

' Set the trigger mode to EDGE.
myScope.WriteString ":TRIGGER:MODE EDGE"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

## **:TRIGger:NREJect**



(see page 658)

**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 411

## :TRIGger:PATTERn



(see page 658)

### Command Syntax

```
:TRIGger:PATTERn <pattern>

<pattern> ::= <value>, <mask> [, <edge source>, <edge>]

<value> ::= integer in NR1 format or <string>

<mask> ::= integer in NR1 format or <string>

<string> ::= "0xnn"; n ::= {0,...,9 | A,...,F}
            (# bits = # channels, see following table)

<edge source> ::= {CHANnel<n> | EXTERNAL | NONE}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTERn command defines the specified pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog channels</b>	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
<b>2 analog channels</b>	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

### NOTE

The optional source and the optional edge should be sent together or not at all. The edge will be set in the simple pattern if it is included. If the edge source is also specified in the mask, the edge takes precedence.

### Query Syntax

```
:TRIGger:PATTERn?
```

The :TRIGger:PATTERn? query returns the pattern value, the mask, and the edge of interest in the simple pattern.

### Return Format

```
<pattern><NL>
```

### See Also

- "Introduction to :TRIGger Commands" on page 411

## **5 Commands by Subsystem**

- [":TRIGger:MODE"](#) on page 417

## :TRIGger:SWEep



(see page 658)

**Command Syntax** :TRIGger:SWEep <sweep>

&lt;sweep&gt; ::= {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>

&lt;sweep&gt; ::= current trigger sweep mode

**See Also** • ["Introduction to :TRIGger Commands" on page 411](#)

## :TRIGger:CAN Commands

**Table 61** :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATTERn:DATA <value>, <mask> (see <a href="#">page 424</a> )	:TRIGger:CAN:PATTERn:DATA? (see <a href="#">page 424</a> )	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-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
:TRIGger:CAN:PATTERn:DATA:LENGTH <length> (see <a href="#">page 425</a> )	:TRIGger:CAN:PATTERn:DATA:LENGTH? (see <a href="#">page 425</a> )	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATTERn:ID <value>, <mask> (see <a href="#">page 426</a> )	:TRIGger:CAN:PATTERn:ID? (see <a href="#">page 426</a> )	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <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
:TRIGger:CAN:PATTERn:ID:MODE <value> (see <a href="#">page 427</a> )	:TRIGger:CAN:PATTERn:ID:MODE? (see <a href="#">page 427</a> )	<value> ::= {STANDARD   EXTENDED} (with Option AMS)
:TRIGger:CAN:SAMPLEpo int <value> (see <a href="#">page 428</a> )	:TRIGger:CAN:SAMPLEpo int? (see <a href="#">page 428</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:CAN:SIGNAL:B AUDrate <baudrate> (see <a href="#">page 429</a> )	:TRIGger:CAN:SIGNAL:B AUDrate? (see <a href="#">page 429</a> )	<baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

**Table 61** :TRIGger:CAN Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
:TRIGger:CAN:SOURce <source> (see page 430)	:TRIGger:CAN:SOURce? (see <a href="#">page 430</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGItal0,..,DIGItal15   } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 431)	:TRIGger:CAN:TRIGger? (see <a href="#">page 432</a> )	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF   DATA   ERRor   IDData   IDEither   IDRmote   ALLerrors   OVERload   ACKerror} (with Option AMS)

**:TRIGger:CAN:PATTERn:DATA**

**N** (see page 658)

**Command Syntax**

```
:TRIGger:CAN:PATTERn:DATA <value>,<mask>
<value> ::= 64-bit integer in decimal, <nondecimal>, or <string>
<mask> ::= 64-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
```

The :TRIGger:CAN:PATTERn:DATA command defines the CAN data pattern resource according to the value and the mask. This pattern, along with the data length (set by the :TRIGger:CAN:PATTERn:DATA:LENGth command), control the data pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

**NOTE**

If more bytes are sent for <value> or <mask> than specified by the :TRIGger:CAN:PATTERn:DATA:LENGth command, the most significant bytes will be truncated. If the data length is changed after the <value> and <mask> are programmed, the added or deleted bytes will be added to or deleted from the least significant bytes.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax**

```
:TRIGger:CAN:PATTERn:DATA?
```

The :TRIGger:CAN:PATTERn:DATA? query returns the current settings of the specified CAN data pattern resource.

**Return Format**

<value>, <mask><NL> in nondecimal format

- ["-241, Hardware missing" on page 617](#)

**See Also**

- ["Introduction to :TRIGger Commands" on page 411](#)
- [":TRIGger:CAN:PATTERn:DATA:LENGth" on page 425](#)
- [":TRIGger:CAN:PATTERn:ID" on page 426](#)

**:TRIGger:CAN:PATTERn:DATA:LENGTH**

**N** (see page 658)

**Command Syntax** :TRIGger:CAN:PATTERn:DATA:LENGTH <length>  
 <length> ::= integer from 1 to 8 in NR1 format

The :TRIGger:CAN: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 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIGger:CAN:PATTERn:DATA command.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATTERn:DATA:LENGTH?

The :TRIGger:CAN: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 617

**See Also** • "Introduction to :TRIGger Commands" on page 411  
 • ":TRIGger:CAN:PATTERn:DATA" on page 424  
 • ":TRIGger:CAN:SOURce" on page 430

**:TRIGger:CAN:PATTERn:ID**

**N** (see page 658)

**Command Syntax**

```
:TRIGger:CAN:PATTERn:ID <value>, <mask>
<value> ::= 32-bit integer in decimal, <nondecimal>, or <string>
<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
```

The :TRIGger:CAN:PATTERn:ID command defines the CAN identifier pattern resource according to the value and the mask. This pattern, along with the identifier mode (set by the :TRIGger:CAN:PATTERn:ID:MODE command), control the identifier pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the identifier pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the identifier stream. Only bits with a "1" set on the mask are used.

**NOTE**

If more bits are sent than allowed (11 bits in standard mode, 29 bits in extended mode) by the :TRIGger:CAN:PATTERn:ID:MODE command, the most significant bytes will be truncated. If the ID mode is changed after the <value> and <mask> are programmed, the added or deleted bits will be added to or deleted from the most significant bits.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax**

```
:TRIGger:CAN:PATTERn:ID?
```

The :TRIGger:CAN:PATTERn:ID? query returns the current settings of the specified CAN identifier pattern resource.

**Return Format**

<value>, <mask><NL> in nondecimal format

**Errors**

- "-241, Hardware missing" on page 617

**See Also**

- "Introduction to :TRIGger Commands" on page 411
- ":TRIGger:CAN:PATTERn:ID:MODE" on page 427
- ":TRIGger:CAN:PATTERn:DATA" on page 424

**:TRIGger:CAN:PATTERn:ID:MODE**

**N** (see page 658)

**Command Syntax** :TRIGger:CAN:PATTERn:ID:MODE <value>  
 <value> ::= {STANdard | EXTended}

The :TRIGger:CAN: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 :TRIGger:CAN:PATTERn:ID command.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATTERn:ID:MODE?

The :TRIGger:CAN: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 617

**See Also** • "Introduction to :TRIGger Commands" on page 411  
 • ":TRIGger:MODE" on page 417  
 • ":TRIGger:CAN:PATTERn:DATA" on page 424  
 • ":TRIGger:CAN:PATTERn:DATA:LENGth" on page 425  
 • ":TRIGger:CAN:PATTERn:ID" on page 426

### :TRIGger:CAN:SAMPLEpoint

**N** (see page 658)

**Command Syntax** :TRIGger:CAN:SAMPLEpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger: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** :TRIGger:CAN:SAMPLEpoint?

The :TRIGger: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 411

• "[:TRIGger:MODE](#)" on page 417

• "[:TRIGger:CAN:TRIGger](#)" on page 431

**:TRIGger:CAN:SIGNAl:BAUDrate**

**N** (see page 658)

**Command Syntax** :TRIGger:CAN:SIGNAl:BAUDrate <baudrate>  
 <baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

The :TRIGger:CAN:SIGNAl:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 1 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.

If the baud rate you select does not match the system baud rate, false triggers may occur.

**Query Syntax** :TRIGger:CAN:SIGNAl:BAUDrate?

The :TRIGger:CAN:SIGNAl:BAUDrate? query returns the current CAN baud rate setting.

**Return Format** <baudrate><NL>  
 <baudrate> ::= integer from 10000 to 1000000 in 100 b/s increments

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGGER:MODE](#)" on page 417
- "[:TRIGGER:CAN:TRIGGER](#)" on page 431
- "[:TRIGGER:CAN:SIGNAl:DEFinition](#)" on page 611
- "[:TRIGGER:CAN:SOURce](#)" on page 430

### :TRIGger:CAN:SOURce

**N** (see page 658)

**Command Syntax** :TRIGger:CAN:SOURce <source>

```
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:CAN:SOURce command sets the source for the CAN signal. The source setting is only valid when :TRIGger:CAN:TRIGger is set to SOF (start of frame).

**Query Syntax** :TRIGger:CAN:SOURce?

The :TRIGger:CAN:SOURce? query returns the current source for the CAN signal.

**Return Format** <source><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:CAN:TRIGger](#)" on page 431
  - "[:TRIGger:CAN:SIGNAl:DEFinition](#)" on page 611

## :TRIGger:CAN:TRIGger

**N** (see page 658)

### Command Syntax :TRIGger:CAN:TRIGger <condition>

```
<condition> ::= {SOF | DATA | ERROR | IDData | IDEither | IDRremote |
    ALLerrors | OVERload | ACKerror}
```

The :TRIGger: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.
- IDRremote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
SOF	SOF - Start of Frame
DATA	Id & Data - Data Frame Id and Data
ERROR	Error - Error frame
IDData	Id & ~RTR - Data Frame Id (~RTR)
IDEither	Id - Remote or Data Frame Id
IDRremote	Id & RTR - Remote Frame Id (RTR)
ALLerrors	All Errors - All Errors
OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the :TRIGger:CAN:PATTERn:ID and :TRIGger:CAN:PATTERn:ID:MODE commands.

CAN Data specification is set by the :TRIGger:CAN:PATTERn:DATA command.

CAN Data Length Code is set by the :TRIGger:CAN:PATTERn:DATA:LENGth command.

### NOTE

SOF is the only valid selection for analog oscilloscopes. If the automotive CAN and LIN serial decode option (Option AMS) has not been licensed, SOF is the only valid selection.

**Query Syntax** :TRIGger:CAN:TRIGger?

The :TRIGger:CAN:TRIGger? query returns the current CAN trigger on condition.

**Return Format** <condition><NL>

<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}

**Errors** • ["-241, Hardware missing" on page 617](#)

**See Also** • ["Introduction to :TRIGger Commands" on page 411](#)  
• [":TRIGger:MODE" on page 417](#)  
• [":TRIGger:CAN:PATTERn:DATA" on page 424](#)  
• [":TRIGger:CAN:PATTERn:DATA:LENGth" on page 425](#)  
• [":TRIGger:CAN:PATTERn:ID" on page 426](#)  
• [":TRIGger:CAN:PATTERn:ID:MODE" on page 427](#)  
• [":TRIGger:CAN:SIGNAL:DEFinition" on page 611](#)  
• [":TRIGger:CAN:SOURce" on page 430](#)

## :TRIGger:DURation Commands

**Table 62** :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see <a href="#">page 434</a> )	:TRIGger:DURation:GREaterthan? (see <a href="#">page 434</a> )	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see <a href="#">page 435</a> )	:TRIGger:DURation:LESSthan? (see <a href="#">page 435</a> )	<less_than_time> ::= floating-point number from in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:PATtern <value>, <mask> (see <a href="#">page 436</a> )	:TRIGger:DURation:PATtern? (see <a href="#">page 436</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= ""0xnnnnnn" n ::= {0,...,9   A,...,F}
:TRIGger:DURation:QUALifier <qualifier> (see <a href="#">page 437</a> )	:TRIGger:DURation:QUALifier? (see <a href="#">page 437</a> )	<qualifier> ::= {GREaterthan   LESSthan   INRange   OUTRange   TIMEOUT}
:TRIGger:DURation:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see <a href="#">page 438</a> )	:TRIGger:DURation:RANGE? (see <a href="#">page 438</a> )	<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:DURation:GREaterthan

**N** (see page 658)

**Command Syntax** :TRIGger:DURation:GREaterthan <greater\_than\_time>[<suffix>]

<greater\_than\_time> ::= minimum trigger duration in seconds  
in NR3 format

<suffix> ::= {s | ms | us | ns | ps}

The :TRIGger:DURation:GREaterthan command sets the minimum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIGger:DURation:QUALifier is set to TIMEOUT.

**Query Syntax** :TRIGger:DURation:GREaterthan?

The :TRIGger:DURation: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 411
  - "[:TRIGger:DURation:PATTern](#)" on page 436
  - "[:TRIGger:DURation:QUALifier](#)" on page 437
  - "[:TRIGger:MODE](#)" on page 417

**:TRIGger:DURation:LESSthan**

**N** (see page 658)

**Command Syntax** :TRIGger:DURation:LESSthan <less\_than\_time>[<suffix>]  
 <less\_than\_time> ::= maximum trigger duration in seconds  
     in NR3 format  
 <suffix> ::= {s | ms | us | ns | ps}

The :TRIGger:DURation:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to LESSthan.

**Query Syntax** :TRIGger:DURation:LESSthan?

The :TRIGger:DURation:LESSthan? query returns the duration time for the defined pattern.

**Return Format** <less\_than\_time><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:DURation:PATTern](#)" on page 436
  - "[:TRIGger:DURation:QUALifier](#)" on page 437
  - "[:TRIGger:MODE](#)" on page 417

**:TRIGger:DURation:PATTERn**

**N** (see page 658)

**Command Syntax** :TRIGger:DURation:PATTERn <value>, <mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnn"; n ::= {0,...,9 | A,...,F}

The :TRIGger:DURation:PATTERn command defines the specified duration pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog channels</b>	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
<b>2 analog channels</b>	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**Query Syntax** :TRIGger:DURation:PATTERn?

The :TRIGger:DURation:PATTERn? query returns the pattern value.

**Return Format** <value>, <mask><NL>

<value> ::= a 32-bit integer in NR1 format.

<mask> ::= a 32-bit integer in NR1 format.

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:PATTERn](#)" on page 419

**:TRIGger:DURation:QUALifier**

**N** (see page 658)

<b>Command Syntax</b>	<code>:TRIGger:DURation:QUALifier &lt;qualifier&gt;</code> <code>&lt;qualifier&gt; ::= {GREaterthan   LESSthan   INRange   OUTRange   TIMEout}</code>
	The :TRIGger:DURation:QUALifier command qualifies the trigger duration.
	Set the GREaterthan qualifier value with the :TRIGger:DURation:GREaterthan command.
	Set the LESSthan qualifier value with the :TRIGger:DURation:LESSthan command.
	Set the INRange and OUTRange qualifier values with the :TRIGger:DURation:RANGE command.
	Set the TIMEout qualifier value with the :TRIGger:DURation:GREaterthan command.
<b>Query Syntax</b>	<code>:TRIGger:DURation:QUALifier?</code>
	The :TRIGger:DURation:QUALifier? query returns the trigger duration qualifier.
<b>Return Format</b>	<code>&lt;qualifier&gt;&lt;NL&gt;</code>
<b>See Also</b>	<ul style="list-style-type: none"> <li>• "<a href="#">Introduction to :TRIGger Commands</a>" on page 411</li> <li>• "<a href="#">":TRIGger:DURation:GREaterthan"</a> on page 434</li> <li>• "<a href="#">":TRIGger:DURation:LESSthan"</a> on page 435</li> <li>• "<a href="#">":TRIGger:DURation:RANGE"</a> on page 438</li> </ul>

### :TRIGger:DURation:RANGE

**N** (see page 658)

**Command Syntax** :TRIGger:DURation: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:DURation:RANGE command sets the duration for the defined pattern when the :TRIGger:DURation: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:DURation:RANGE?

The :TRIGger:DURation: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 411
  - "[:TRIGger:DURation:PATTern](#)" on page 436
  - "[:TRIGger:DURation:QUALifier](#)" on page 437
  - "[:TRIGger:MODE](#)" on page 417

## :TRIGger[:EDGE] Commands

**Table 63** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LF} (see page 440)	:TRIGger[:EDGE]:COUPling? (see page 440)	{AC   DC   LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 441)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 441)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LF   HF} (see page 442)	:TRIGger[:EDGE]:REJect? (see page 442)	{OFF   LF   HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 443)	:TRIGger[:EDGE]:SLOPe? (see page 443)	<polarity> ::= {POSitive   NEGative   EITHer   ALternate}
:TRIGger[:EDGE]:SOURce <source> (see page 444)	:TRIGger[:EDGE]:SOURce? (see page 444)	<source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format

**:TRIGger[:EDGE]:COUPLing**

(see page 658)

**Command Syntax** :TRIGger[:EDGE]:COUPLing <coupling>  
 <coupling> ::= {AC | DC | LFR}

The :TRIGger[:EDGE]:COUPLing command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LFR.

- 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.
- LFR 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 411
- ":TRIGger:MODE" on page 417
- ":TRIGger[:EDGE]:REJECT" on page 442

**:TRIGger[:EDGE]:LEVel**

(see page 658)

**Command Syntax** :TRIGger[:EDGE]:LEVel <level>  
 <level> ::= <level>[,<source>]  
 <level> ::= 0.75 x full-scale voltage from center screen in NR3 format  
     for internal triggers  
 <level> ::= ±(external range setting) in NR3 format  
     for external triggers  
 <source> ::= {CHANnel<n> | EXTERNAL}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

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 411
  - "[":TRIGger\[:EDGE\]:SOURce](#)" on page 444
  - "[":EXTERNAL:RANGE](#)" on page 226

### :TRIGger[:EDGE]:REject



(see page 658)

**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 411
- "[:TRIGger:HFReject](#)" on page 415
- "[:TRIGger\[:EDGE\]:COUPLing](#)" on page 440

**:TRIGger[:EDGE]:SLOPe**

(see page 658)

**Command Syntax** :TRIGger[:EDGE]:SLOPe <slope>

&lt;slope&gt; ::= {NEGative | POSitive | 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>

&lt;slope&gt; ::= {NEG | POS | ALT}

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:TV:POLarity](#)" on page 482

**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"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

### :TRIGger[:EDGE]:SOURce



(see page 658)

**Command Syntax** :TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTERNAL | LINE}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger[:EDGE]:SOURce command selects the channel that produces the trigger.

**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 | NONE}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:MODE](#)" on page 417

**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"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

## :TRIGger:GLITch Commands

**Table 64** :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater_than_time>[suffix] (see <a href="#">page 446</a> )	:TRIGger:GLITch:GREaterthan? (see <a href="#">page 446</a> )	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:LESSthan <less_than_time>[suffix] (see <a href="#">page 447</a> )	:TRIGger:GLITch:LESSthan? (see <a href="#">page 447</a> )	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see <a href="#">page 448</a> )	:TRIGger:GLITch:LEVel? (see <a href="#">page 448</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= ±(external range setting) in NR3 format. <source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see <a href="#">page 449</a> )	:TRIGger:GLITch:POLarity? (see <a href="#">page 449</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see <a href="#">page 450</a> )	:TRIGger:GLITch:QUALifier? (see <a href="#">page 450</a> )	<qualifier> ::= {GREaterthan   LESSthan   RANGE}
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see <a href="#">page 451</a> )	:TRIGger:GLITch:RANGE? (see <a href="#">page 451</a> )	<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 <a href="#">page 452</a> )	:TRIGger:GLITch:SOURce? (see <a href="#">page 452</a> )	<source> ::= {CHANnel<n>   EXTERNAL} <n> ::= 1-2 or 1-4 in NR1 format

### :TRIGger:GLITch:GREaterthan

**N** (see page 658)

**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 411  
• "[:TRIGger:GLITch:SOURce](#)" on page 452  
• "[:TRIGger:GLITch:QUALifier](#)" on page 450  
• "[:TRIGger:MODE](#)" on page 417

**:TRIGger:GLITch:LESSthan**

**N** (see page 658)

**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 411  
• "[:TRIGger:GLITch:SOURce](#)" on page 452  
• "[:TRIGger:GLITch:QUALifier](#)" on page 450  
• "[:TRIGger:MODE](#)" on page 417

### :TRIGger:GLITch:LEVel

**N** (see page 658)

**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  
<source> ::= {CHANnel<n> | EXTERNAL}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:GLITch:SOURce](#)" on page 452
  - "[:EXTERNAL:RANGE](#)" on page 226

**:TRIGger:GLITch:POLarity**

**N** (see page 658)

**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 411  
• "[:TRIGger:MODE](#)" on page 417  
• "[:TRIGger:GLITch:SOURce](#)" on page 452

### :TRIGger:GLITch:QUALifier

**N** (see page 658)

**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 411
- "[:TRIGger:GLITch:SOURce](#)" on page 452
- "[:TRIGger:MODE](#)" on page 417

**:TRIGger:GLITch:RANGE**

**N** (see page 658)

**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 411
  - "[:TRIGger:GLITch:SOURce](#)" on page 452
  - "[:TRIGger:GLITch:QUALifier](#)" on page 450
  - "[:TRIGger:MODE](#)" on page 417

### :TRIGger:GLITch:SOURce

**N** (see page 658)

**Command Syntax** :TRIGger:GLITch:SOURce <source>

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

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:GLITch:LEVel](#)" on page 448
  - "[:TRIGger:GLITch:POLarity](#)" on page 449
  - "[:TRIGger:GLITch:QUALifier](#)" on page 450
  - "[:TRIGger:GLITch:RANGE](#)" on page 451

**Example Code**

- "[Example Code](#)" on page 444

## :TRIGger:IIC Commands

**Table 65** :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATTern:ADDReSS <value> (see page 454)	:TRIGger:IIC:PATTern:ADDReSS? (see page 454)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATTern:DATA <value> (see page 455)	:TRIGger:IIC:PATTern:DATA? (see page 455)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATTern:DATA2 <value> (see page 456)	:TRIGger:IIC:PATTern:DATA2? (see page 456)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC[:SOURce]:CLOCk <source> (see page 457)	:TRIGger:IIC[:SOURce]:CLOCk? (see page 457)	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGITAL0,...,DIGITAL15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see page 458)	:TRIGger:IIC[:SOURce]:DATA? (see page 458)	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGITAL0,...,DIGITAL15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see page 459)	:TRIGger:IIC:TRIGger:QUALifier? (see page 459)	<value> ::= {EQUAL   NOTEQUAL   LESSthan   GREATERthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see page 460)	:TRIGger:IIC:TRIGger[:TYPE]? (see page 460)	<type> ::= {START   STOP   READ7   READEprom   WRITE7   WRITE10   NACKnowledge   ANACKnowledge   R7Data2   W7Data2   REStart}

### :TRIGger:IIC:PATTERn:ADDResS

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:PATTERn:ADDResS <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC: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** :TRIGger:IIC:PATTERn:ADDResS?

The :TRIGger:IIC:PATTERn:ADDResS? query returns the current address for IIC data.

**Return Format** <value><NL>

<value> ::= integer

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:IIC:PATTERn:DATA](#)" on page 455
  - "[:TRIGger:IIC:PATTERn:DATa2](#)" on page 456
  - "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 460

**:TRIGger:IIC:PATTERn:DATA**

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:PATTERn:DATA <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC: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** :TRIGger:IIC:PATTERn:DATA?

The :TRIGger:IIC:PATTERn:DATA? query returns the current pattern for IIC data.

**Return Format** <value><NL>

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:IIC:PATTERn:ADDResS](#)" on page 454
- "[:TRIGger:IIC:PATTERn:DATa2](#)" on page 456
- "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 460

### :TRIGger:IIC:PATTERn:DATA2

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:PATTERn:DATA2 <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC: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** :TRIGger:IIC:PATTERn:DATA2?

The :TRIGger:IIC:PATTERn:DATA2? query returns the current pattern for IIC data 2.

**Return Format** <value><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:IIC:PATTERn:ADDRess](#)" on page 454
  - "[:TRIGger:IIC:PATTERn:DATA](#)" on page 455
  - "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 460

**:TRIGger:IIC[:SOURce]:CLOCk**

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:[SOURce:]CLOCk <source>  
 <source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
 <source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:IIC[:SOURce:]CLOCk command sets the source for the IIC serial clock (SCL).

**Query Syntax** :TRIGger:IIC:[SOURce:]CLOCk?

The :TRIGger: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 411
  - ":TRIGger:IIC[:SOURce]:DATA" on page 458

### :TRIGger:IIC[:SOURce]:DATA

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:[SOURce:]DATA <source>  
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:IIC[:SOURce:]DATA command sets the source for IIC serial data (SDA).

**Query Syntax** :TRIGger:IIC:[SOURce:]DATA?

The :TRIGger:IIC[:SOURce:]DATA? query returns the current source for IIC serial data.

**Return Format** <source><NL>

- See Also**
- "Introduction to :TRIGger Commands" on page 411
  - ":TRIGger:IIC[:SOURce]:CLOCK" on page 457

**:TRIGger:IIC:TRIGger:QUALifier**

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:TRIGger:QUALifier <value>

<value> ::= {EQUAL | NOTEQUAL | LESSThan | GREATERthan}

The :TRIGger:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEprom.

**Query Syntax** :TRIGger:IIC:TRIGger:QUALifier?

The :TRIGger: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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 460

**:TRIGger:IIC:TRIGger[:TYPE]**

**N** (see page 658)

**Command Syntax** :TRIGger:IIC:TRIGger[:TYPE] <value>

```
<value> ::= {STAR | STOP | READ7 | READEprom | WRITe7 | WRITe10
| NACKnowledge | ANACKnowledge | R7Data2 | W7Data2 | RESTart}
```

The :TRIGger: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.
- ANACKnowledge – 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 660).

**Query Syntax** :TRIGger:IIC:TRIGger[:TYPE] ?

The :TRIGger: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 411
- ":TRIGger:MODE" on page 417

- "[:TRIGGER:IIC:PATTERN:ADDRess](#)" on page 454
- "[:TRIGGER:IIC:PATTERN:DATA](#)" on page 455
- "[:TRIGGER:IIC:PATTERN:DATa2](#)" on page 456
- "[:TRIGGER:IIC:TRIGGER:QUALifier](#)" on page 459
- "["Long Form to Short Form Truncation Rules"](#) on page 660

## :TRIGger:LIN Commands

**Table 66** :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see page 463)	:TRIGger:LIN:ID? (see page 463)	<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
:TRIGger:LIN:SAMPLEpo int <value> (see page 464)	:TRIGger:LIN:SAMPLEpo int? (see page 464)	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:LIN:SIGNAl:B AUDrate <baudrate> (see page 465)	:TRIGger:LIN:SIGNAl:B AUDrate? (see page 465)	<baudrate> ::= integer from 2400 to 625000 in 100 b/s increments
:TRIGger:LIN:SOURce <source> (see page 466)	:TRIGger:LIN:SOURce? (see page 466)	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGItal0,...,DIGItal15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANDARD <std> (see page 467)	:TRIGger:LIN:STANDARD ? (see page 467)	<std> ::= {LIN13   LIN20}
:TRIGger:LIN:SYNCbreak <value> (see page 468)	:TRIGger:LIN:SYNCbreak? (see page 468)	<value> ::= integer = {11   12   13}
:TRIGger:LIN:TRIGGER <condition> (see page 469)	:TRIGger:LIN:TRIGGER? (see page 469)	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak   ID} (with Option AMS)

**:TRIGger:LIN:ID**

**N** (see page 658)

**Command Syntax** :TRIGger:LIN: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 :TRIGger:LIN:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

**Query Syntax** :TRIGger:LIN:ID?

The :TRIGger:LIN:ID? query returns the current LIN identifier setting.

**Return Format** <value><NL>

<value> ::= integer in decimal

**Errors**

- "-241, Hardware missing" on page 617

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:LIN:TRIGger](#)" on page 469
- "[:TRIGger:LIN:SIGNal:DEFinition](#)" on page 612
- "[:TRIGger:LIN:SOURce](#)" on page 466

## :TRIGger:LIN:SAMPLEpoint

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:SAMPLEpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger: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

:TRIGger:LIN:SAMPLEpoint?

The :TRIGger: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 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:LIN:TRIGger](#)" on page 469

**:TRIGger:LIN:SIGNAl:BAUDrate**

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:SIGNAl:BAUDrate <baudrate>  
 <baudrate> ::= integer from 2400 to 625000 in 100 b/s increments

The :TRIGger: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** :TRIGger:LIN:SIGNAl:BAUDrate?

The :TRIGger: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 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:LIN:TRIGger](#)" on page 469
- "[:TRIGger:LIN:SIGNAl:DEFinition](#)" on page 612
- "[:TRIGger:LIN:SOURce](#)" on page 466

### :TRIGger:LIN:SOURce

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:SOURce <source>

```
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:LIN:SOURce command sets the source for the LIN signal.

**Query Syntax** :TRIGger:LIN:SOURce?

The :TRIGger:LIN:SOURce? query returns the current source for the LIN signal.

**Return Format** <source><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:LIN:TRIGger](#)" on page 469
  - "[:TRIGger:LIN:SIGNAl:DEFinition](#)" on page 612

**:TRIGger:LIN:STANDARD**

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:STANDARD <std>  
<std> ::= {LIN13 | LIN20}

The :TRIGger:LIN:STANDARD command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

**Query Syntax** :TRIGger:LIN:STANDARD?

The :TRIGger:LIN:STANDARD? query returns the current LIN standard setting.

**Return Format** <std><NL>  
<std> ::= {LIN13 | LIN20}

**See Also** • "[Introduction to :TRIGger Commands](#)" on page 411  
• "[:TRIGger:MODE](#)" on page 417  
• "[:TRIGger:LIN:SIGNAl:DEFinition](#)" on page 612  
• "[:TRIGger:LIN:SOURce](#)" on page 466

### :TRIGger:LIN:SYNCbreak

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:SYNCbreak <value>

<value> ::= integer = {11 | 12 | 13}

The :TRIGger: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** :TRIGger:LIN:SYNCbreak?

The :TRIGger:LIN:STANDARD? query returns the current LIN sync break setting.

**Return Format** <value><NL>

<value> ::= {11 | 12 | 13}

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:LIN:SIGNAL:DEFinition](#)" on page 612
- "[:TRIGger:LIN:SOURce](#)" on page 466

**:TRIGger:LIN:TRIGger**

**N** (see page 658)

**Command Syntax** :TRIGger:LIN:TRIGger <condition>  
 <condition> ::= {SYNCbreak | ID}

The :TRIGger:LIN:TRIGger command sets the LIN trigger on condition to be Sync Break (SYNCbreak) or Frame Id (ID).

**NOTE**

The ID option is available when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:LIN:TRIGger?

The :TRIGger:LIN:TRIGger? query returns the current LIN trigger value.

**Return Format** <condition><NL>  
 <condition> ::= {SYNC | ID}

**Errors** • "-241, Hardware missing" on page 617

**See Also** • "Introduction to :TRIGger Commands" on page 411  
 • ":TRIGger:MODE" on page 417  
 • ":TRIGger:LIN:SIGNal:DEFinition" on page 612  
 • ":TRIGger:LIN:SOURce" on page 466

## :TRIGger:SPI Commands

**Table 67** :TRIGger:SPI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCK:SL OPe <slope> (see page 471)	:TRIGger:SPI:CLOCK:SL OPe? (see <a href="#">page 471</a> )	<slope> ::= {NEGative   POSitive}
:TRIGger:SPI:CLOCK:TI Meout <time_value> (see page 472)	:TRIGger:SPI:CLOCK:TI Meout? (see <a href="#">page 472</a> )	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 473)	:TRIGger:SPI:FRAMing? (see <a href="#">page 473</a> )	<value> ::= {CHIPselect   NOTChipselect   TIMeout}
:TRIGger:SPI:PATTern: DATA <value>, <mask> (see page 474)	:TRIGger:SPI:PATTern: DATA? (see <a href="#">page 474</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnnnnnn" where n ::= {0,...,9   A,...,F}
:TRIGger:SPI:PATTern: WIDTh <width> (see page 475)	:TRIGger:SPI:PATTern: WIDTh? (see <a href="#">page 475</a> )	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:C LOCK <source> (see page 476)	:TRIGger:SPI:SOURce:C LOCK? (see <a href="#">page 476</a> )	<value> ::= {CHANnel<n>   EXTernal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:D ATA <source> (see page 477)	:TRIGger:SPI:SOURce:D ATA? (see <a href="#">page 477</a> )	<value> ::= {CHANnel<n>   EXTernal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:F RAME <source> (see page 478)	:TRIGger:SPI:SOURce:F RAME? (see <a href="#">page 478</a> )	<value> ::= {CHANnel<n>   EXTernal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format

**:TRIGger:SPI:CLOCk:SLOPe**

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:CLOCk:SLOPe <slope>  
<slope> ::= {NEGative | POSitive}

The :TRIGger: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** :TRIGger:SPI:CLOCk:SLOPe?

The :TRIGger: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 411  
• "[:TRIGger:SPI:CLOCk:TIMEout](#)" on page 472  
• "[:TRIGger:SPI:SOURce:CLOCk](#)" on page 476

### :TRIGger:SPI:CLOCk:TIMEout

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:CLOCK:TIMEout <time\_value>  
<time\_value> ::= time in seconds in NR1 format

The :TRIGger:SPI:CLOCk:TIMEout command sets the SPI signal clock timeout resource in seconds from 500 ns to 10 s when the :TRIGger:SPI:FRAMing command is set to TIMEout. The timer is used to frame a signal by a clock timeout.

**Query Syntax** :TRIGger:SPI:CLOCK:TIMEout?

The :TRIGger:SPI:CLOCk:TIMEout? query returns current SPI clock timeout setting.

**Return Format** <time value><NL>  
<time\_value> ::= time in seconds in NR1 format

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:SPI:CLOCk:SLOPe](#)" on page 471
- "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
- "[:TRIGger:SPI:FRAMing](#)" on page 473

## :TRIGger:SPI:FRAMing

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:FRAMing <value>

<value> ::= {CHIPselect | NOTChipselect | TIMEout}

The :TRIGger:SPI:FRAMing command sets the SPI trigger framing value. If TIMEout is selected, the timeout value is set by the :TRIGger:SPI:CLOCK:TIMEout command.

**Query Syntax** :TRIGger:SPI:FRAMing?

The :TRIGger:SPI:FRAMing? query returns the current SPI framing value.

**Return Format** <value><NL>

<value> ::= {CHIPselect | NOTChipselect | TIMEout}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:SPI:CLOCK:TIMEout](#)" on page 472
  - "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478

### :TRIGger:SPI:PATTern:DATA

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:PATTern:DATA <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

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

The :TRIGger:SPI:PATTern:DATA command defines the SPI data pattern resource according to the value and the mask. This pattern, along with the data width, control the data pattern searched for in the data stream.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

**Query Syntax** :TRIGger:SPI:PATTern:DATA?

The :TRIGger:SPI:PATTern:DATA? query returns the current settings of the specified SPI data pattern resource.

**Return Format** <value>, <mask><NL>

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:SPI:PATTern:WIDTh](#)" on page 475
- "[:TRIGger:SPI:SOURce:DATA](#)" on page 477

**:TRIGger:SPI:PATTERn:WIDTh**

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:PATTERn:WIDTh <width>

<width> ::= integer from 4 to 32 in NR1 format

The :TRIGger:SPI:PATTERn:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 32 bits.

**Query Syntax** :TRIGger:SPI:PATTERn:WIDTh?

The :TRIGger:SPI:PATTERn:WIDTh? query returns the current SPI data pattern width setting.

**Return Format** <width><NL>

<width> ::= integer from 4 to 32 in NR1 format

**See Also**

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:SPI:PATTERn:DATA](#)" on page 474
- "[:TRIGger:SPI:SOURce:DATA](#)" on page 477

### :TRIGger:SPI:SOURce:CLOCK

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:SOURce:CLOCK <source>

<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models

<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:SPI:SOURce:CLOCK command sets the source for the SPI serial clock.

**Query Syntax** :TRIGger:SPI:SOURce:CLOCK?

The :TRIGger: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 411
  - "[:TRIGger:SPI:CLOCK:SLOPe](#)" on page 471
  - "[:TRIGger:SPI:CLOCK:TIMEout](#)" on page 472
  - "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478
  - "[:TRIGger:SPI:SOURce:DATA](#)" on page 477

**:TRIGger:SPI:SOURce:DATA**

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:SOURce:DATA <source>  
 <source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
 <source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:SPI:SOURce:DATA command sets the source for the SPI serial data.

**Query Syntax** :TRIGger:SPI:SOURce:DATA?

The :TRIGger:SPI:SOURce:DATA? query returns the current source for the SPI serial data.

**Return Format** <source><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
  - "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478
  - "[:TRIGger:SPI:PATTERn:DATA](#)" on page 474
  - "[:TRIGger:SPI:PATTERn:WIDTh](#)" on page 475

### :TRIGger:SPI:SOURce:FRAMe

**N** (see page 658)

**Command Syntax** :TRIGger:SPI:SOURce:FRAMe <source>  
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger:SPI:SOURce:FRAMe command sets the frame source when :TRIGger:SPI:FRAMing is set to CHIPselect or NOTChipselect.

**Query Syntax** :TRIGger:SPI:SOURce:FRAMe?

The :TRIGger: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 411
  - "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
  - "[:TRIGger:SPI:SOURce:DATA](#)" on page 477
  - "[:TRIGger:SPI:FRAMing](#)" on page 473

## :TRIGger:TV Commands

**Table 68** :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 480)	:TRIGger:TV:LINE? (see <a href="#">page 480</a> )	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 481)	:TRIGger:TV:MODE? (see <a href="#">page 481</a> )	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LINE   VERTICAL   LFIELD1   LFIELD2   LATERNATE   LVERTICAL}
:TRIGger:TV:POLarity <polarity> (see page 482)	:TRIGger:TV:POLarity? (see <a href="#">page 482</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:TV:SOURce <source> (see page 483)	:TRIGger:TV:SOURce? (see <a href="#">page 483</a> )	<source> ::= {CHANNEL<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANDARD <standard> (see page 484)	:TRIGger:TV:STANDARD? (see <a href="#">page 484</a> )	<standard> ::= {GENERIC   NTSC   PALM   PAL   SECAM   {P480L60HZ   P480}   {P720L60HZ   P720}   {P1080L24HZ   P1080}   P1080L25HZ   {I1080L50HZ   I1080}   I1080L60HZ}

**:TRIGger:TV:LINE**

**N** (see page 658)

**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 69** TV Trigger Line Number Limits

TV Standard	Mode				
	LINE	LField1	LField2	LALternate	VERTical
<b>NTSC</b>		1 to 263	1 to 262	1 to 262	
<b>PAL</b>		1 to 313	314 to 625	1 to 312	
<b>PAL-M</b>		1 to 263	264 to 525	1 to 262	
<b>SECAM</b>		1 to 313	314 to 625	1 to 312	
<b>GENERIC</b>		1 to 1024	1 to 1024		1 to 1024
<b>P480L60HZ</b>	1 to 525				
<b>P720L60HZ</b>	1 to 750				
<b>P1080L24HZ</b>	1 to 1125				
<b>P1080L25HZ</b>	1 to 1125				
<b>I1080L50HZ</b>	1 to 1125				
<b>I1080L60HZ</b>	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 411  
 • ":TRIGger:TV:STANDARD" on page 484  
 • ":TRIGger:TV:MODE" on page 481

**:TRIGger:TV:MODE**

**N** (see page 658)

**Command Syntax** :TRIGger:TV:MODE <mode>

```
<mode> ::= {FIELD1 | FIELD2 | AFIELDS | ALINES | LINE | VERTical
             | LFIELD1 | LFIELD2 | LALTernate | LVERTical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANDARD is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANDARD is GENeric.

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
LVERTical	LINEVert

**Query Syntax** :TRIGger:TV:MODE?

The :TRIGger:TV:MODE? query returns the TV trigger mode.

**Return Format** <value><NL>

```
<value> ::= {FIELD1 | FIELD2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
              | LALT | LVER}
```

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 411
  - "[:TRIGger:TV:STANDARD](#)" on page 484
  - "[:TRIGger:MODE](#)" on page 417

## **:TRIGger:TV:POLarity**

**N** (see page 658)

**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 411  
                  • ":TRIGger:MODE" on page 417  
                  • ":TRIGger:TV:SOURce" on page 483

**:TRIGger:TV:SOURce****N** (see page 658)**Command Syntax** :TRIGger:TV:SOURce <source>

```
<source> ::= {CHANnel<n>}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 411
  - ":TRIGger:MODE" on page 417
  - ":TRIGger:TV:POLarity" on page 482

- Example Code**
- "Example Code" on page 444

### :TRIGger:TV:STANdard

**N** (see page 658)

**Command Syntax** :TRIGger:TV:STANdard <standard>

```
<standard> ::= {GENeric | NTSC | PALM | PAL | SECam  
| {P480L60HZ | P480} | {P720L60HZ | P720}  
| {P1080L24HZ | P1080} | P1080L25HZ  
| {I1080L50HZ | I1080} | I1080L60HZ}
```

The :TRIGger:TV:STANdard command selects the video standard. GENeric mode is non-interlaced.

**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 | I1080L50HZ | I1080L60HZ}
```

## :TRIGger:UART Commands

**Table 70** :TRIGger:UART Commands Summary

Command	Query	Options and Query Returns
:TRIGger:UART:BASE <base> (see page 487)	:TRIGger:UART:BASE? (see page 487)	<base> ::= {ASCII   HEX}
:TRIGger:UART:BAUDrate <baudrate> (see page 488)	:TRIGger:UART:BAUDrate? (see page 488)	<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments
:TRIGger:UART:BITorder <bitorder> (see page 489)	:TRIGger:UART:BITorder? (see page 489)	<bitorder> ::= {LSBFIRST   MSBFIRST}
:TRIGger:UART:BURSt <value> (see page 490)	:TRIGger:UART:BURSt? (see page 490)	<value> ::= {OFF   1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see page 491)	:TRIGger:UART:DATA? (see page 491)	<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)
:TRIGger:UART:IDLE <time_value> (see page 492)	:TRIGger:UART:IDLE? (see page 492)	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 493)	:TRIGger:UART:PARity? (see page 493)	<parity> ::= {EVEN   ODD   NONE}
:TRIGger:UART:PolaritY <polarity> (see page 494)	:TRIGger:UART:PolaritY? (see page 494)	<polarity> ::= {HIGH   LOW}
:TRIGger:UART:QUALifier <value> (see page 495)	:TRIGger:UART:QUALifier? (see page 495)	<value> ::= {EQUAL   NOTEQUAL   GREATERthan   LESSthan}

## 5 Commands by Subsystem

**Table 70** :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:SOURce: RX <source> (see page 496)	:TRIGger:UART:SOURce: RX? (see <a href="#">page 496</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:SOURce: TX <source> (see page 497)	:TRIGger:UART:SOURce: TX? (see <a href="#">page 497</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see page 498)	:TRIGger:UART:TYPE? (see <a href="#">page 498</a> )	<value> ::= {RSTArt   RSTOP   RDATA   RD1   RD0   RDX   PARityerror   TSTArt   TSTOP   TDATA   TD1   TD0   TDX}
:TRIGger:UART:WIDTh <width> (see page 499)	:TRIGger:UART:WIDTh? (see <a href="#">page 499</a> )	<width> ::= {5   6   7   8   9}

**:TRIGger:UART:BASE**

**N** (see page 658)

**Command Syntax** :TRIGger:UART:BASE <base>  
 <base> ::= {ASCii | HEX}

The :TRIGger:UART: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 :TRIGger:UART:BASE setting does not affect the :TRIGger:UART:DATA command which can always set data values using ASCII or hexadecimal values.

**NOTE**

The :TRIGger:UART:BASE command is independent of the :SBUS:UART:BASE command which affects decode only.

**Query Syntax** :TRIGger:UART:BASE?

The :TRIGger:UART:BASE? query returns the current UART base setting.

**Return Format** <base><NL>  
 <base> ::= {ASC | HEX}

**See Also** • "[Introduction to :TRIGger Commands](#)" on page 411  
 • "[:TRIGger:MODE](#)" on page 417  
 • "[:TRIGger:UART:DATA](#)" on page 491

### :TRIGger:UART:BAUDrate

**N** (see page 658)

**Command Syntax** :TRIGger:UART:BAUDrate <baudrate>

<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments

The :TRIGger: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 1200 b/s to 3 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.

If the baud rate you select does not match the system baud rate, false triggers may occur.

**Query Syntax** :TRIGger:UART:BAUDrate?

The :TRIGger:UART:BAUDrate? query returns the current UART baud rate setting.

**Return Format** <baudrate><NL>

<baudrate> ::= integer from 1200 to 3000000 in 100 b/s increments

- See Also**
- "Introduction to :TRIGger Commands" on page 411
  - ":TRIGger:MODE" on page 417
  - ":TRIGger:UART:TYPE" on page 498

**:TRIGger:UART:BITorder**

**N** (see page 658)

**Command Syntax** :TRIGger:UART:BITorder <bitorder>  
 <bitorder> ::= {LSBFFirst | MSBFFirst}

The :TRIGger: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** :TRIGger:UART:BITorder?

The :TRIGger: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 411
- "[:TRIGGER:MODE](#)" on page 417
- "[:TRIGger:UART:TYPE](#)" on page 498
- "[:TRIGger:UART:SOURce:RX](#)" on page 496
- "[:TRIGger:UART:SOURce:TX](#)" on page 497

### :TRIGger:UART:BURSt

**N** (see page 658)

**Command Syntax** :TRIGger:UART:BURSt <value>

<value> ::= {OFF | 1 to 4096 in NR1 format}

The :TRIGger:UART: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** :TRIGger:UART:BURSt?

The :TRIGger:UART: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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:UART:IDLE](#)" on page 492
  - "[:TRIGger:UART:TYPE](#)" on page 498

## :TRIGger:UART:DATA

**N** (see page 658)

### Command Syntax :TRIGger: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 :TRIGger:UART: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 :TRIGger:UART:DATA?

The :TRIGger:UART: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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:UART:BASE](#)" on page 487
  - "[:TRIGger:UART:TYPE](#)" on page 498

### :TRIGger:UART:IDLE

**N** (see page 658)

**Command Syntax** :TRIGger:UART:IDLE <time\_value>

<time\_value> ::= time from 1 us to 10 s in NR3 format

The :TRIGger:UART: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** :TRIGger:UART:IDLE?

The :TRIGger:UART: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 411
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:UART:BURSt](#)" on page 490
  - "[:TRIGger:UART:TYPE](#)" on page 498

**:TRIGger:UART:PARity**

**N** (see page 658)

**Command Syntax** :TRIGger:UART:PARity <parity>  
<parity> ::= {EVEN | ODD | NONE}

The :TRIGger: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** :TRIGger:UART:PARity?

The :TRIGger: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 411  
• "[:TRIGger:MODE](#)" on page 417  
• "[:TRIGger:UART:TYPE](#)" on page 498

## **:TRIGger:UART:POLarity**

**N** (see page 658)

**Command Syntax**    :TRIGger:UART:POLarity <polarity>  
                          <polarity> ::= {HIGH | LOW}

The :TRIGger: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**    :TRIGger:UART:POLarity?

The :TRIGger:UART:POLarity? query returns the current UART polarity setting.

**Return Format**    <polarity><NL>  
                          <polarity> ::= {HIGH | LOW}

**See Also**    • "Introduction to :TRIGger Commands" on page 411  
                  • ":TRIGger:MODE" on page 417  
                  • ":TRIGger:UART:TYPE" on page 498

**:TRIGger:UART:QUALifier**

**N** (see page 658)

**Command Syntax** :TRIGger:UART:QUALifier <value>

<value> ::= {EQUAL | NOTEqual | GREaterthan | LESSthan}

The :TRIGger:UART: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** :TRIGger:UART:QUALifier?

The :TRIGger:UART: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 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:UART:TYPE](#)" on page 498

## **:TRIGger:UART:SOURce:RX**

**N** (see page 658)

**Command Syntax**    :TRIGger:UART:SOURce:RX <source>  
  
<source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
  
<source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
  
<n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger: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**    :TRIGger:UART:SOURce:RX?

The :TRIGger: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 411
  - "[:TRIGGER:MODE](#)" on page 417
  - "[:TRIGGER:UART:TYPE](#)" on page 498
  - "[:TRIGGER:UART:BITOrder](#)" on page 489

## :TRIGger:UART:SOURce:TX

**N** (see page 658)

**Command Syntax** :TRIGger:UART:SOURce:TX <source>  
 <source> ::= {CHANnel<n> | EXTERNAL} for the DSO models  
 <source> ::= {CHANnel<n> | DIGITAL0,...,DIGITAL15} for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :TRIGger: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** :TRIGger:UART:SOURce:TX?

The :TRIGger: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 411
  - "[:TRIGGER:MODE](#)" on page 417
  - "[:TRIGGER:UART:TYPE](#)" on page 498
  - "[:TRIGGER:UART:BITOrder](#)" on page 489

### :TRIGger:UART:TYPE

**N** (see page 658)

**Command Syntax** :TRIGger:UART:TYPE <value>

```
<value> ::= {RSTArt | RSTOP | RDATa | RD1 | RD0 | RDX | PARityerror  
| TSTArt | TSTOP | TDATa | TD1 | TD0 | TDX}
```

The :TRIGger:UART:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :TRIGger:UART:DATA and :TRIGger:UART: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** :TRIGger:UART:TYPE?

The :TRIGger:UART: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 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:UART:DATA](#)" on page 491
- "[:TRIGger:UART:QUALifier](#)" on page 495
- "[:TRIGger:UART:WIDTH](#)" on page 499

**:TRIGger:UART:WIDTH**

**N** (see page 658)

**Command Syntax** :TRIGger:UART:WIDTH <width>  
<width> ::= {5 | 6 | 7 | 8 | 9}

The :TRIGger: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** :TRIGger:UART:WIDTH?

The :TRIGger: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 411  
• "[:TRIGger:MODE](#)" on page 417  
• "[:TRIGger:UART:TYPE](#)" on page 498

## :WAVeform Commands

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 502.

**Table 71** :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 507)	:WAVeform:BYTeorder? (see <a href="#">page 507</a> )	<value> ::= {LSBFirst   MSBFFirst}
n/a	:WAVeform:COUNT? (see <a href="#">page 508</a> )	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see <a href="#">page 509</a> )	<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 511)	:WAVeform:FORMAT? (see <a href="#">page 511</a> )	<value> ::= {WORD   BYTE   ASCII}
:WAVeform:POINTS <# points> (see <a href="#">page 512</a> )	:WAVeform:POINTS? (see <a href="#">page 512</a> )	<# 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 <a href="#">page 514</a> )	:WAVeform:POINTS:MODE ? (see <a href="#">page 514</a> )	<points_mode> ::= {NORMAl   MAXimum   RAW}

**Table 71** :WAVEform Commands Summary (continued)

<b>Command</b>	<b>Query</b>	<b>Options and Query Returns</b>
n/a	:WAVEform:PREamble? (see <a href="#">page 516</a> )	<p>&lt;preamble_block&gt; ::= &lt;format NR1&gt;, &lt;type NR1&gt;, &lt;points NR1&gt;, &lt;count NR1&gt;, &lt;xincrement NR3&gt;, &lt;xorigin NR3&gt;, &lt;xreference NR1&gt;, &lt;yincrement NR3&gt;, &lt;yorigin NR3&gt;, &lt;yreference NR1&gt;</p> <p>&lt;format&gt; ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> <li>• 0 for BYTE format</li> <li>• 1 for WORD format</li> <li>• 2 for ASCII format</li> </ul> <p>&lt;type&gt; ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> <li>• 0 for NORMAL type</li> <li>• 1 for PEAK detect type</li> <li>• 2 for AVERAGE type</li> <li>• 3 for HRESolution type</li> </ul> <p>&lt;count&gt; ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</p>
n/a	:WAVEform:SEGmented:COUNT? (see <a href="#">page 519</a> )	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see <a href="#">page 520</a> )	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see <a href="#">page 521</a> )	:WAVEform:SOURce? (see <a href="#">page 521</a> )	<p>&lt;source&gt; ::= {CHANnel&lt;n&gt;   FUNCtion   MATH}</p> <p>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</p>
:WAVEform:SOURce:SUBS ource <subsource> (see <a href="#">page 525</a> )	:WAVEform:SOURce:SUBS ource? (see <a href="#">page 525</a> )	<subsource> ::= {{NONE   RX}   TX}
n/a	:WAVEform:TYPE? (see <a href="#">page 526</a> )	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVEform:UNSIGNED { {0   OFF}   {1   ON} } (see <a href="#">page 527</a> )	:WAVEform:UNSIGNED? (see <a href="#">page 527</a> )	{0   1}
:WAVEform:VIEW <view> (see <a href="#">page 528</a> )	:WAVEform:VIEW? (see <a href="#">page 528</a> )	<view> ::= {MAIN}
n/a	:WAVEform:XINCREMENT? (see <a href="#">page 529</a> )	<return_value> ::= x-increment in the current preamble in NR3 format

**Table 71** :WAveform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAveform:XORigin? (see <a href="#">page 530</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAveform:XREFerence? (see <a href="#">page 531</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAveform:YINCrement? (see <a href="#">page 532</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAveform:YORigin? (see <a href="#">page 533</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAveform:YREFerence? (see <a href="#">page 534</a> )	<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 509](#)) and :WAveform:PREamble (see [page 516](#)). 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 three types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command (see [page 177](#)): NORMal, AVERage, PEAK, and HRESolution. When the data is acquired using the :DIGItize command (see [page 132](#)) or :RUN command (see [page 156](#)), 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 509](#)) 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 170](#)).

#### **Helpful Hints:**

The number of points transferred to the computer is controlled using the :WAVEFORM:POINTS command (see [page 512](#)). If :WAVEFORM:POINTS MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVEFORM:POINTS may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVEFORM:POINTS must be an even divisor of 1,000 or be set to MAXimum. :WAVEFORM:POINTS determines the increment between time buckets that will be transferred. If POINTS = MAXimum, the data cannot be decimated. For example:

- :WAVEFORM:POINTS 1000 – returns time buckets 0, 1, 2, 3, 4, ..., 999.
- :WAVEFORM:POINTS 500 – returns time buckets 0, 2, 4, 6, 8, ..., 998.
- :WAVEFORM:POINTS 250 – returns time buckets 0, 4, 8, 12, 16, ..., 996.
- :WAVEFORM:POINTS 100 – returns time buckets 0, 10, 20, 30, 40, ..., 990.

#### Analog Channel Data

#### **NORMal Data**

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVEFORM:POINTS? query (see [page 512](#)). 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 167](#)). 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 512](#)). 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 512](#)). 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 177](#)), the value returned by the :WAVeform:XINCrement query (see [page 529](#)) should be doubled to find the time difference between the min-max pairs.

### HRESolution Data

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

### Data Conversion

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

voltage = [(data value - yreference) \* yincrement] + yorigin

If the :WAVeform:FORMAT data format is ASCII (see [page 511](#)), the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

time = [(data point number - xreference) \* xincrement] + xorigin

This would result in the following calculation for time bucket 3:

time = [(3 - 0) \* 2 ns] + 16 ns = 22 ns

In :ACQuire:TYPE PEAK mode (see [page 177](#)), because data is acquired in max-min pairs, modify the previous time formula to the following:

time=[(data pair number - xreference) \* xincrement \* 2] + xorigin

### Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCII (see "[:WAVeform:FORMAT](#)" on page 511). 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 527](#)) 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 511) 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 507](#)) and :WAVEform:UNSIGNED (see [page 527](#)) have no effect when the format is ASCII.

### Data Format for Transfer - WORD format

WORD format (see "[:WAVEform:FORMAT](#)" on page 511) 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 512](#)). 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 507](#)) 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 511) 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 507](#)) has no effect when the data format is BYTE.

### Reporting the Setup

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

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS  
NONE
```

**:WAVeform:BYTeorder**

(see page 658)

**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 502
- "[:WAVeform:DATA](#)" on page 509
- "[:WAVeform:FORMAT](#)" on page 511
- "[:WAVeform:PREamble](#)" on page 516

**Example Code**

- "[Example Code](#)" on page 521
- "[Example Code](#)" on page 517

## **:WAVeform:COUNt**



(see page 658)

**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 502
  - "[:ACQuire:COUNt](#)" on page 167
  - "[:ACQuire:TYPE](#)" on page 177

**:WAVeform:DATA**

(see page 658)

**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. Holes can be reasonably expected in the equivalent time acquisition mode (especially at slower horizontal sweep speeds when measuring low frequency signals).

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**

&lt;binary block data&gt;&lt;NL&gt;

**See Also**

- "Introduction to :WAVeform Commands" on page 502
- ":WAVeform:UNSIGNED" on page 527
- ":WAVeform:BYTeorder" on page 507
- ":WAVeform:FORMat" on page 511
- ":WAVeform:POINts" on page 512
- ":WAVeform:PREamble" on page 516
- ":WAVeform:SOURce" on page 521
- ":WAVeform:TYPE" on page 526

**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:
```

## 5 Commands by Subsystem

```
'           <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
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on page 744

**:WAVeform:FORMat**

(see page 658)

**Command Syntax** :WAVeform:FORMat <value>

&lt;value&gt; ::= {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.

**Query Syntax** :WAVeform:FORMat?

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

**Return Format** <value><NL>

&lt;value&gt; ::= {WORD | BYTE | ASC}

- See Also**
- "[Introduction to :WAVeform Commands](#)" on page 502
  - "[":WAVeform:BYTeorder](#)" on page 507
  - "[":WAVeform:DATA](#)" on page 509
  - "[":WAVeform:PREamble](#)" on page 516

- Example Code**
- "[Example Code](#)" on page 521

**:WAVeform:POINts**

(see page 658)

**Command Syntax**

```
:WAVeform:POINts <# points>

<# 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}
```

**NOTE**

The <points\_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog sources, 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 and may be up to 8,000,000. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

See the :WAVeform:POINts:MODE command (see [page 514](#)) for more information on the <points\_mode> option.

Only data visible on the display will be returned.

The maximum number of points returned when the waveform source is math or function is 1000.

**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 514](#)) for more information).

**Return Format**

```
<# points><NL>
```

```

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
    if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000
    in 1-2-5 sequence | <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 502
- "[":ACQuire:POINts](#)" on page 170
- "[":WAVEform:DATA](#)" on page 509
- "[":WAVEform:SOURce](#)" on page 521
- "[":WAVEform:VIEW](#)" on page 528
- "[":WAVEform:PREamble](#)" on page 516
- "[":WAVEform:POINts:MODE](#)" on page 514

**Example Code**

```

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

```

Example program from the start: "["VISA COM Example in Visual Basic"](#)" on page 744

**:WAVeform:POINts:MODE**

**N** (see [page 658](#))

**Command Syntax**

```
:WAVeform:POINts:MODE <points_mode>
<points_mode> ::= {NORMAl | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

If the <points\_mode> is NORMAl, the measurement record is retrieved.

If the <points\_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points\_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), or if the reconstruction filter (Sin(x)/x interpolation) is in use, 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 160](#)) or the :DIGitize command ([see page 132](#)) in the root subsystem) in order to return more than 1000 points.
- :TIMEbase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMAl, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than 1000 points.
- MAXimum or RAW will allow up to 8,000,000 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 502  
• "[:ACQuire:POINts](#)" on page 170  
• "[:WAveform:DATA](#)" on page 509  
• "[:WAveform:VIEW](#)" on page 528  
• "[:WAveform:PREamble](#)" on page 516  
• "[:WAveform:POINts](#)" on page 512  
• "[:TIMEbase:MODE](#)" on page 402  
• "[:ACQuire:TYPE](#)" on page 177  
• "[:ACQuire:COUNt](#)" on page 167

### :WAVeform:PREamble



(see page 658)

#### 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.

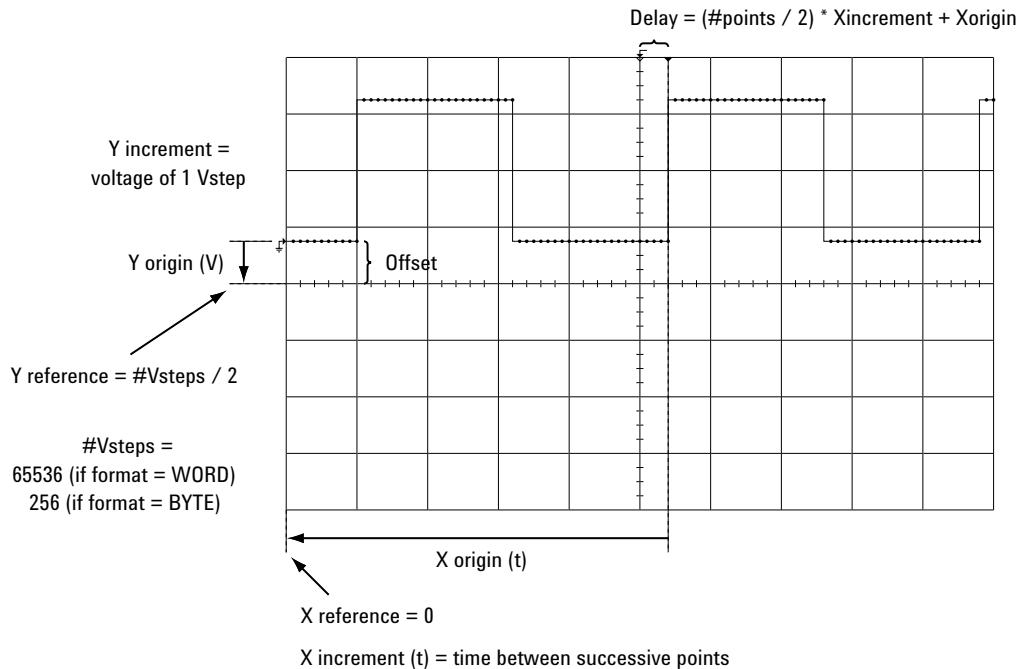
#### Return Format <preamble\_block><NL>

```
<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 502
  - ":ACQuire:COUNt" on page 167
  - ":ACQuire:POINts" on page 170
  - ":ACQuire:TYPE" on page 177
  - ":DIGitize" on page 132
  - ":WAVEform:COUNt" on page 508
  - ":WAVEform:DATA" on page 509
  - ":WAVEform:FORMAT" on page 511
  - ":WAVEform:POINts" on page 512
  - ":WAVEform:TYPE" on page 526
  - ":WAVEform:XINCrement" on page 529
  - ":WAVEform:XORigin" on page 530
  - ":WAVEform:XREFerence" on page 531
  - ":WAVEform:YINCrement" on page 532
  - ":WAVEform:YORigin" on page 533
  - ":WAVEform:YREFerence" on page 534

**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.
```

## 5 Commands by Subsystem

```
' 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)
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

**:WAVeform:SEGmented:COUNt****N** (see page 658)**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 250 in NR1 format (count set by :ACQuire:SEGmented:COUNT).

- See Also**
- "[:ACQuire:MODE](#)" on page 169
  - "[:ACQuire:SEGmented:COUNt](#)" on page 172
  - "[:DIGITIZE](#)" on page 132
  - "[:SINGle](#)" on page 158
  - "[:RUN](#)" on page 156
  - "[Introduction to :WAVeform Commands](#)" on page 502

**Example Code** • "[Example Code](#)" on page 173

## :WAVeform:SEGmented:TTAG

**N** (see page 658)

**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 173
  - "["Introduction to :WAVeform Commands"](#) on page 502

**Example Code**

- "["Example Code"](#) on page 173

**:WAVeform:SOURce**

**C** (see page 658)

**Command Syntax**

```
:WAVeform:SOURce <source>
<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :WAVeform:SOURce command selects the analog channel, function, 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 (SBUS), ASCII is the only waveform format allowed.

**Query Syntax**

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

**NOTE**

MATH is an alias for FUNCtion. The :WAVeform:SOURce? query returns FUNC if the source is FUNCtion or MATH.

**Return Format**

```
<source><NL>
<source> ::= {CHAN<n> | FUNC | SBUS}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

**See Also**

- "[Introduction to :WAVeform Commands](#)" on page 502
- "[:DIGitize](#)" on page 132
- "[:WAVeform:FORMat](#)" on page 511
- "[:WAVeform:BYTeorder](#)" on page 507
- "[:WAVeform:DATA](#)" on page 509
- "[:WAVeform:PREamble](#)" on page 516

**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"
```

## 5 Commands by Subsystem

```
' 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, 2 = 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(db1XIncrement * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'    FormatNumber(db1XOrigin * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'    CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'    FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'    FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'    CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
    FormatNumber(lngVSteps * sngYIncrement / 8) + _
    " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
    FormatNumber((lngVSteps / 2 - lngYReference) * _
    sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
    FormatNumber(lngPoints * db1XIncrement / 10 * _
    1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
    FormatNumber(((lngPoints / 2 - lngXReference) * _
    db1XIncrement + db1XOrigin) * 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.

```

## 5 Commands by Subsystem

```
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
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 744

**:WAVeform:SOURce:SUBSource****C**

(see page 658)

**Command Syntax**

```
:WAVeform:SOURce:SUBSource <subsource>
<subsource> ::= {{NONE | RX} | TX}
```

If the :WAVeform:SOURce is SBUS (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

Currently, only UART serial decode lets you get "TX" data. The default, NONE, specifies "RX" data. (RX is an alias for NONE.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS:MODE is not UART, the only valid subsource is NONE.

**Query Syntax**

```
:WAVeform:SOURce:SUBSource?
```

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

**Return Format**

```
<subsource><NL>
<subsource> ::= {NONE | TX}
```

**See Also**

- "[Introduction to :WAVeform Commands](#)" on page 502
- "[":WAVeform:SOURce"](#) on page 521

## **:WAVeform:TYPE**



(see page 658)

**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}

- See Also**
- "Introduction to :WAVeform Commands" on page 502
  - ":ACQuire:TYPE" on page 177
  - ":WAVeform:DATA" on page 509
  - ":WAVeform:PREamble" on page 516
  - ":WAVeform:SOURce" on page 521

**:WAVeform:UNSIGNED**

(see page 658)

**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.

**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 502  
• ":WAVeform:SOURce" on page 521

## **:WAveform:VIEW**



(see page 658)

**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 502  
                    • ":WAveform:POINts" on page 512

**:WAVeform:XINCrement**

(see page 658)

**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 502
  - ":WAVeform:PREamble" on page 516

**Example Code**

- "Example Code" on page 517

## **:WAVeform:XORigin**



(see page 658)

**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 502
  - "[:WAVeform:PREamble](#)" on page 516
  - "[:WAVeform:XREFerence](#)" on page 531

**Example Code**

- "[Example Code](#)" on page 517

**:WAVeform:XREFerence**

(see page 658)

**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 502
  - ":WAVeform:PREamble" on page 516
  - ":WAVeform:XORigin" on page 530

**Example Code**

- "Example Code" on page 517

## **:WAVeform:YINCrement**



(see page 658)

**Query Syntax**    `:WAVeform:YINCrement?`

The `:WAVeform:YINCrement?` query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values.

**Return Format**    `<value><NL>`

`<value>` ::= y-increment value in the current preamble in 32-bit floating point NR3 format

**See Also**

- "Introduction to :WAVeform Commands" on page 502
- "`:WAVeform:PREamble`" on page 516

**Example Code**

- "Example Code" on page 517

**:WAVeform:YORigin** (see page 658)**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 502
  - "[:WAVeform:PREamble](#)" on page 516
  - "[:WAVeform:YREFerence](#)" on page 534

**Example Code**

- "[Example Code](#)" on page 517

### :WAVeform:YREFerence



(see page 658)

#### **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.

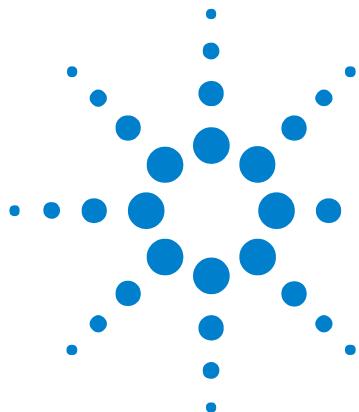
#### **Return Format** <value><NL>

<value> ::= y-reference value in the current preamble in 32-bit  
NR1 format

- See Also**
- "[Introduction to :WAVeform Commands](#)" on page 502
  - "[:WAVeform:PREamble](#)" on page 516
  - "[:WAVeform:YORigin](#)" on page 533

**Example Code**

- "[Example Code](#)" on page 517



## 6 Commands A-Z

A	535
B	536
C	537
D	539
E	540
F	541
G	542
H	542
I	543
L	543
M	544
N	547
O	547
P	548
Q	549
R	550
S	551
T	555
U	558
V	559
W	560
X	561
Y	561

- A**
  - [":ACQuire:AALias"](#) on page 165
  - [":ACQuire:AALias"](#) on page 165
  - [":ACQuire:COMplete"](#) on page 166
  - [":ACQuire:COUNt"](#) on page 167
  - [":ACQuire:DAALias"](#) on page 168
  - [":ACQuire:MODE"](#) on page 169



- "[:ACQuire:POINTs](#)" on page 170
  - "[:ACQuire:SEGMENTed:ANALyze](#)" on page 171
  - "[:ACQuire:SEGMENTed:COUNt](#)" on page 172
  - "[:ACQuire:SEGMENTed:INDEX](#)" on page 173
  - "[:ACQuire:SRATe](#)" on page 176
  - "[:ACQuire:TYPE](#)" on page 177
  - ADDRess, "[:TRIGger:IIC:PATTERn:ADDRess](#)" on page 454
  - "[:AER \(Arm Event Register\)](#)" on page 125
  - AMASk Commands:
    - "[:MTEST:AMASK:CREAtE](#)" on page 323
    - "[:MTEST:AMASK:{SAVE | STORe}](#)" on page 599
    - "[:MTEST:AMASK:SOURce](#)" on page 324
    - "[:MTEST:AMASK:UNITs](#)" on page 325
    - "[:MTEST:AMASK:XDELta](#)" on page 326
    - "[:MTEST:AMASK:YDELta](#)" on page 327
  - ANALyze, "[:ACQuire:SEGMENTed:ANALyze](#)" on page 171
  - APRinter, "[:HARDcopy:APRinter](#)" on page 248
  - AREA Commands:
    - "[:HARDcopy:AREA](#)" on page 247
    - "[:SAVE:IMAGe:AREA](#)" on page 361
  - ASIZE, "[:SBUS:IIC:ASIZE](#)" on page 381
  - "[:AUToscale](#)" on page 126
    - "[:AUToscale:AMODE](#)" on page 128
    - "[:AUToscale:CHANnels](#)" on page 129
  - AVERage Commands:
    - "[:MTEST:AVERage](#)" on page 600
    - "[:MTEST:AVERage:COUNt](#)" on page 601
- B**
- BASE Commands:
    - "[:SBUS:UART:BASE](#)" on page 385
    - "[:TRIGger:UART:BASE](#)" on page 487
  - BAUDrate Commands:
    - "[:TRIGger:CAN:SIGNAl:BAUDrate](#)" on page 429
    - "[:TRIGger:LIN:SIGNAl:BAUDrate](#)" on page 465
    - "[:TRIGger:UART:BAUDrate](#)" on page 488
  - BIND, "[:MTEST:SCALE:BIND](#)" on page 344

- BITorder, ":TRIGger:UART:BITorder" on page 489
  - ":BLANK" on page 130
  - BURSt, ":TRIGger:UART:BURSt" on page 490
  - BWLimit Commands:
    - ":CHANnel<n>:BWLimit" on page 192
    - ":EXTernal:BWLimit" on page 220
  - BYTeorder, ".WAVEform:BYTeorder" on page 507
- C**
- ":CALibrate:DATE" on page 181
  - ":CALibrate:LABel" on page 182
  - ":CALibrate:OUTPut" on page 183
  - ":CALibrate:STARt" on page 184
  - ":CALibrate:STATus" on page 185
  - ":CALibrate:SWITch" on page 186
  - ":CALibrate:TEMPerature" on page 187
  - ":CALibrate:TIME" on page 188
  - CAN Commands:
    - ":SBUS:CAN:COUNt:ERRor" on page 375
    - ":SBUS:CAN:COUNt:OVERload" on page 376
    - ":SBUS:CAN:COUNt:RESet" on page 377
    - ":SBUS:CAN:COUNt:TOTal" on page 378
    - ":SBUS:CAN:COUNt:UTILization" on page 379
    - ":TRIGger:CAN Commands" on page 422
  - ":CDISplay" on page 131
  - CENTER, ":FUNCTION:CENTER" on page 231
  - ":CHANnel:LABel" on page 568
  - ":CHANnel2:SKEW" on page 569
  - ":CHANnel<n>:BWLimit" on page 192
  - ":CHANnel<n>:COUpling" on page 193
  - ":CHANnel<n>:DISPlay" on page 194
  - ":CHANnel<n>:IMPedance" on page 195
  - ":CHANnel<n>:INPut" on page 570
  - ":CHANnel<n>:INVert" on page 196
  - ":CHANnel<n>:LABel" on page 197
  - ":CHANnel<n>:OFFSet" on page 198
  - ":CHANnel<n>:PMODE" on page 571

- "[:CHANnel<n>:PROBe](#)" on page 199
- "[:CHANnel<n>:PROBe:ID](#)" on page 200
- "[:CHANnel<n>:PROBe:SKEW](#)" on page 201
- "[:CHANnel<n>:PROBe:STYPe](#)" on page 202
- "[:CHANnel<n>:PROTection](#)" on page 203
- "[:CHANnel<n>:RANGE](#)" on page 204
- "[:CHANnel<n>:SCALE](#)" on page 205
- "[:CHANnel<n>:UNITS](#)" on page 206
- "[:CHANnel<n>:VERNier](#)" on page 207
- CLEar Commands:
  - "[:DISPlay:CLEar](#)" on page 210
  - "[:MEASure:CLEar](#)" on page 274
- CLOCk Commands:
  - "[:TRIGger:IIC\[:SOURce\]:CLOCK](#)" on page 457
  - "[:TRIGger:SPI:CLOCK:SLOPe](#)" on page 471
  - "[:TRIGger:SPI:CLOCK:TIMEout](#)" on page 472
  - "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
- "[\\*CLS \(Clear Status\)](#)" on page 101
- COMplete, "[:ACQuire:COMplete](#)" on page 166
- CONDITION, "[:HWEregister:CONDITION \(Hardware Event Condition Register\)](#)" on page 136
- CONNect, "[:DISPlay:CONNect](#)" on page 572
- COUNT Commands:
  - "[:ACQuire:COUNT](#)" on page 167
  - "[:ACQuire:SEGmented:COUNT](#)" on page 172
  - "[:MTESt:AVERage:COUNT](#)" on page 601
  - "[:MTESt:COUNT:FWAVEforms](#)" on page 328
  - "[:MTESt:COUNT:RESet](#)" on page 329
  - "[:MTESt:COUNT:TIME](#)" on page 330
  - "[:MTESt:COUNT:WAVEforms](#)" on page 331
  - "[:SBUS:CAN:COUNT:ERRor](#)" on page 375
  - "[:SBUS:CAN:COUNT:OVERload](#)" on page 376
  - "[:SBUS:CAN:COUNT:RESet](#)" on page 377
  - "[:SBUS:CAN:COUNT:TOTal](#)" on page 378
  - "[:SBUS:CAN:COUNT:UTILization](#)" on page 379

- ":SBUS:UART:COUNt:ERRor" on page 386
- ":SBUS:UART:COUNt:RESet" on page 387
- ":SBUS:UART:COUNt:RXFRames" on page 388
- ":SBUS:UART:COUNt:TXFRames" on page 389
- ":WAveform:COUNt" on page 508
- ":WAveform:SEGmented:COUNt" on page 519
- COUNter, ":MEASure:COUNter" on page 275
- COUpling Commands:
  - ":CHANnel<n>:COUpling" on page 193
  - ":TRIGger[:EDGE]:COUpling" on page 440
- CREAtE, ":MTEST:AMASk:CREAtE" on page 323
- D**
  - DAALias, ":ACQuire:DAALias" on page 168
  - DATA Commands:
    - ":DISPlay:DATA" on page 211
    - ":MTEST:DATA" on page 332
    - ":TRIGger:CAN:PATTern:DATA" on page 424
    - ":TRIGger:CAN:PATTern:DATA:LENGth" on page 425
    - ":TRIGger:IIC:PATTern:DATA" on page 455
    - ":TRIGger:IIC:PATTern:DATA2" on page 456
    - ":TRIGger:IIC[:SOURce]:DATA" on page 458
    - ":TRIGger:SPI:PATTern:DATA" on page 474
    - ":TRIGger:SPI:SOURce:DATA" on page 477
    - ":TRIGger:UART:DATA" on page 491
    - ":WAveform:DATA" on page 509
  - DATE Commands:
    - ":CALibrate:DATE" on page 181
    - ":SYSTem:DATE" on page 392
  - DEFInE, ":MEASure:DEFInE" on page 276
  - DELay Commands:
    - ":MEASure:DELay" on page 279
    - ":TIMEbase:DELay" on page 609
  - DELetE, ":MTEST:DELetE" on page 333
  - DESTination, ":HARDcopy:DESTination" on page 578
  - DEVice, ":HARDcopy:DEVice" on page 579
  - ":DIGItize" on page 132

- DISPlay Commands:
    - "[:CHANnel<n>:DISPlay](#)" on page 194
    - "[:FUNCTION:DISPlay](#)" on page 232
    - "[:SBUS:DISPlay](#)" on page 380
  - "[:DISPlay:CLEAr](#)" on page 210
  - "[:DISPlay:CONNect](#)" on page 572
  - "[:DISPlay:DATA](#)" on page 211
  - "[:DISPlay:LABel](#)" on page 213
  - "[:DISPlay:LABList](#)" on page 214
  - "[:DISPlay:PERSistence](#)" on page 215
  - "[:DISPlay:SOURce](#)" on page 216
  - "[:DISPlay:VECTors](#)" on page 217
  - DSP, "[:SYSTem:DSP](#)" on page 393
  - DURation, "[:TRIGger:DURation Commands](#)" on page 433
  - DUTYcycle, "[:MEASure:DUTYcycle](#)" on page 281
- E**
- EDGE, "[:TRIGger\[:EDGE\] Commands](#)" on page 439
  - ENABLE"[:MTEST:ENABLE](#)" on page 334
  - "[:ERASe](#)" on page 573
  - ERRor Commands:
    - "[:SBUS:CAN:COUNt:ERRor](#)" on page 375
    - "[:SBUS:UART:COUNt:ERRor](#)" on page 386
    - "[:SYSTem:ERRor](#)" on page 394
  - "[:\\*ESE \(Standard Event Status Enable\)](#)" on page 102
  - "[:\\*ESR \(Standard Event Status Register\)](#)" on page 104
  - EVENT Commands:
    - "[:HWERegister\[:EVENT\] \(Hardware Event Event Register\)](#)" on page 138
    - "[:MTERegister\[:EVENT\] \(Mask Test Event Event Register\)](#)" on page 143
  - "[:EXTernal:BWLimit](#)" on page 220
  - "[:EXTernal:IMPedance](#)" on page 221
  - "[:EXTernal:INPut](#)" on page 574
  - "[:EXTernal:PMODE](#)" on page 575
  - "[:EXTernal:PROBe](#)" on page 222
  - "[:EXTernal:PROBe:ID](#)" on page 223
  - "[:EXTernal:PROBe:STYPe](#)" on page 224

- "[:EXTernal:PROTection](#)" on page 225
  - "[:EXTernal:RANGE](#)" on page 226
  - "[:EXTernal:UNITs](#)" on page 227
- F**
- FACTion Commands:
    - "[:MTEST:RMODE:FACTion:PRINT](#)" on page 338
    - "[:MTEST:RMODE:FACTion:SAVE](#)" on page 339
    - "[:MTEST:RMODE:FACTion:STOP](#)" on page 340
  - FACTors Commands:
    - "[:HARDcopy:FACTors](#)" on page 249
    - "[:SAVE:IMAGE:FACTors](#)" on page 362
  - FALLtime, "[:MEASure:FALLtime](#)" on page 282
  - FFEed, "[:HARDcopy:FFEed](#)" on page 250
  - FILEname Commands:
    - "[:HARDcopy:FILENAME](#)" on page 580
    - "[:RECall:FILENAME](#)" on page 352
    - "[:SAVE:FILENAME](#)" on page 359
  - FORMat Commands:
    - "[:HARDcopy:FORMAT](#)" on page 581
    - "[:SAVE:IMAGE:FORMAT](#)" on page 363
    - "[:SAVE:WAVEform:FORMAT](#)" on page 370
    - "[:WAVEform:FORMAT](#)" on page 511
  - FRAMe, "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478
  - FRAMing Commands:
    - "[:SBUS:UART:FRAMing](#)" on page 390
    - "[:TRIGger:SPI:FRAMing](#)" on page 473
  - FREQuency, "[:MEASure:FREQuency](#)" on page 283
  - "[:FUNCTION:CENTER](#)" on page 231
  - "[:FUNCTION:DISPLAY](#)" on page 232
  - "[:FUNCTION:GOFT:OPERation](#)" on page 233
  - "[:FUNCTION:GOFT:SOURce1](#)" on page 234
  - "[:FUNCTION:GOFT:SOURce2](#)" on page 235
  - "[:FUNCTION:OFFSet](#)" on page 236
  - "[:FUNCTION:OPERation](#)" on page 237
  - "[:FUNCTION:RANGE](#)" on page 238
  - "[:FUNCTION:REFERENCE](#)" on page 239

- [":FUNCTION:SCALE"](#) on page 240
  - [":FUNCTION:SOURce"](#) on page 576
  - [":FUNCTION:SOURce1"](#) on page 241
  - [":FUNCTION:SOURce2"](#) on page 242
  - [":FUNCTION:SPAN"](#) on page 243
  - [":FUNCTION:VIEW"](#) on page 577
  - [":FUNCTION:WINDOW"](#) on page 244
  - FWAVeforms, [":MTEST:COUNT:FWAVeforms"](#) on page 328
- G**
- GLITch (Pulse Width), [":TRIGger:GLITch Commands"](#) on page 445
  - GOFT Commands:
    - [":FUNCTION:GOFT:OPERation"](#) on page 233
    - [":FUNCTION:GOFT:SOURce1"](#) on page 234
    - [":FUNCTION:GOFT:SOURce2"](#) on page 235
  - GRAYscale, [":HARDcopy:GRAYscale"](#) on page 582
  - GREaterthan Commands:
    - [":TRIGger:DURATION:GREaterthan"](#) on page 434
    - [":TRIGger:GLITch:GREaterthan"](#) on page 446
- H**
- [":HARDcopy:AREA"](#) on page 247
  - [":HARDcopy:APRinter"](#) on page 248
  - [":HARDcopy:DESTination"](#) on page 578
  - [":HARDcopy:DEVice"](#) on page 579
  - [":HARDcopy:FACTors"](#) on page 249
  - [":HARDcopy:FFEd"](#) on page 250
  - [":HARDcopy:FILENAME"](#) on page 580
  - [":HARDcopy:FORMAT"](#) on page 581
  - [":HARDcopy:GRAYscale"](#) on page 582
  - [":HARDcopy:IGColors"](#) on page 583
  - [":HARDcopy:INKSaver"](#) on page 251
  - [":HARDcopy:LAYout"](#) on page 252
  - [":HARDcopy:PALETTE"](#) on page 253
  - [":HARDcopy:PDRIVER"](#) on page 584
  - [":HARDcopy:PRINTER:LIST"](#) on page 254
  - [":HARDcopy:START"](#) on page 255
  - HFReject, [":TRIGger:HFReject"](#) on page 415

- HOLDoff, ":TRIGger:HOLDoff" on page 416
- ":HWEnable (Hardware Event Enable Register)" on page 134
- ":HWERegister:CONDition (Hardware Event Condition Register)" on page 136
- ":HWERegister[:EVENT] (Hardware Event Event Register)" on page 138
- I**
  - ID Commands:
    - ":TRIGger:CAN:PATTern:ID" on page 426
    - ":TRIGger:CAN:PATTern:ID:MODE" on page 427
  - IDLE, ":TRIGger:UART:IDLE" on page 492
  - "\*IDN (Identification Number)" on page 106
  - IIC Commands:
    - ":SBUS:IIC:ASIZe" on page 381
    - ":TRIGger:IIC Commands" on page 453
  - IGColors Commands:
    - ":HARDcopy:IGColors" on page 583
    - ":SAVE:IMAGE:INKSaver" on page 364
  - IMAGe Commands:
    - ":RECall:IMAGE[:STARt]" on page 353
    - ":SAVE:IMAGE:AREA" on page 361
    - ":SAVE:IMAGE:FACTors" on page 362
    - ":SAVE:IMAGE:FORMAT" on page 363
    - ":SAVE:IMAGE:INKSaver" on page 364
    - ":SAVE:IMAGE:PAlette" on page 365
    - ":SAVE:IMAGE[:STARt]" on page 360
  - IMPedance Commands:
    - ":CHANnel<n>:IMPedance" on page 195
    - ":EXternal:IMPedance" on page 221
  - INCReement, ":MEASure:STATistics:INCReement" on page 300
  - INDex, ":ACQuire:SEGmented:INDex" on page 173
  - INKSaver, ":HARDcopy:INKSaver" on page 251
  - INVert, ":CHANnel<n>:INVert" on page 196
- L**
  - LABel Commands:
    - ":CALibrate:LABel" on page 182
    - ":CHANnel:LABel" on page 568
    - ":CHANnel<n>:LABel" on page 197

- [":DISPLAY:LABEL"](#) on page 213
  - [LABList, ":DISPLAY:LABELIST"](#) on page 214
  - [LAYout, ":HARDcopy:LAYout"](#) on page 252
  - [LENGTH Commands:](#)
    - [":SAVE:WAVEform:LENGTH"](#) on page 371
    - [":TRIGGER:CAN:PATTERn:DATA:LENGTH"](#) on page 425
  - [LESSthan Commands:](#)
    - [":TRIGGER:DURATION:LESSthan"](#) on page 435
    - [":TRIGGER:GLITCH:LESSthan"](#) on page 447
  - [LEVEL Commands:](#)
    - [":TRIGGER\[:EDGE\]:LEVEL"](#) on page 441
    - [":TRIGGER:GLITCH:LEVEL"](#) on page 448
  - [LIN Commands:](#)
    - [":SBUS:LIN:PARity"](#) on page 382
    - [":TRIGGER:LIN Commands"](#) on page 462
  - [LINE, ":TRIGGER:TV:LINE"](#) on page 480
  - [LIST, ":HARDcopy:PRINTER:LIST"](#) on page 254
  - [LOAD, ":MTEST:LOAD"](#) on page 602
  - [LOCK Commands:](#)
    - [":MTEST:LOCK"](#) on page 335
    - [":SYSTEM:LOCK"](#) on page 395
    - [":SYSTEM:PROTECTION:LOCK"](#) on page 396
  - [LOWER, ":MEASURE:LOWER"](#) on page 585
  - ["\\*LRN \(Learn Device Setup\)"](#) on page 107
- M**
- [":MARKER:MODE"](#) on page 258
  - [":MARKER:X1Position"](#) on page 259
  - [":MARKER:X1Y1source"](#) on page 260
  - [":MARKER:X2Position"](#) on page 261
  - [":MARKER:X2Y2source"](#) on page 262
  - [":MARKER:XDELTa"](#) on page 263
  - [":MARKER:Y1Position"](#) on page 264
  - [":MARKER:Y2Position"](#) on page 265
  - [":MARKER:YDELTa"](#) on page 266
  - [MASK Commands:](#)
    - [":RECALL:MASK\[:START\]"](#) on page 354

- "[:SAVE:MASK\[:STARt\]](#)" on page 366
- "[:MEASure:CLEar](#)" on page 274
- "[:MEASure:COUNter](#)" on page 275
- "[:MEASure:DEFIne](#)" on page 276
- "[:MEASure:DELay](#)" on page 279
- "[:MEASure:DUTYcycle](#)" on page 281
- "[:MEASure:FALLtime](#)" on page 282
- "[:MEASure:FREQuency](#)" on page 283
- "[:MEASure:LOWer](#)" on page 585
- "[:MEASure:NWIDth](#)" on page 284
- "[:MEASure:OVERshoot](#)" on page 285
- "[:MEASure:PERiod](#)" on page 287
- "[:MEASure:PHASe](#)" on page 288
- "[:MEASure:PREShoot](#)" on page 289
- "[:MEASure:PVIDth](#)" on page 290
- "[:MEASure:RESults](#)" on page 291
- "[:MEASure:RISetime](#)" on page 294
- "[:MEASure:SCRatch](#)" on page 586
- "[:MEASure:SDEViation](#)" on page 295
- "[:MEASure:SHOW](#)" on page 296
- "[:MEASure:SOURce](#)" on page 297
- "[:MEASure:STATistics](#)" on page 299
- "[:MEASure:STATistics:INCReement](#)" on page 300
- "[:MEASure:STATistics:RESet](#)" on page 301
- "[:MEASure:TDELta](#)" on page 587
- "[:MEASure:TEDGE](#)" on page 302
- "[:MEASure:THresholds](#)" on page 588
- "[:MEASure:TMAX](#)" on page 589
- "[:MEASure:TMIN](#)" on page 590
- "[:MEASure:TSTARt](#)" on page 591
- "[:MEASure:TSTOP](#)" on page 592
- "[:MEASure:TVALue](#)" on page 304
- "[:MEASure:TVOLt](#)" on page 593
- "[:MEASure:UPPer](#)" on page 595
- "[:MEASure:VAMPplitude](#)" on page 306

- "[:MEASure:VAverage](#)" on page 307
- "[:MEASure:VBASe](#)" on page 308
- "[:MEASure:VDELta](#)" on page 596
- "[:MEASure:VMAX](#)" on page 309
- "[:MEASure:VMIN](#)" on page 310
- "[:MEASure:VPP](#)" on page 311
- "[:MEASure:VRATio](#)" on page 312
- "[:MEASure:VRMS](#)" on page 313
- "[:MEASure:VSTARt](#)" on page 597
- "[:MEASure:VSTOP](#)" on page 598
- "[:MEASure:VTIMe](#)" on page 314
- "[:MEASure:VTOP](#)" on page 315
- "[:MEASure:XMAX](#)" on page 316
- "[:MEASure:XMIN](#)" on page 317
- "[:MERGe](#)" on page 140
- MODE Commands:
  - "[:ACQuire:MODE](#)" on page 169
  - "[:MARKer:MODE](#)" on page 258
  - "[:SBUS:MODE](#)" on page 383
  - "[:TIMEbase:MODE](#)" on page 402
  - "[:TRIGger:CAN:PATTERn:ID:MODE](#)" on page 427
  - "[:TRIGger:MODE](#)" on page 417
  - "[:TRIGger:TV:MODE](#)" on page 481
  - "[:WAVeform:POINTs:MODE](#)" on page 514
- "[:MTEnable \(Mask Test Event Enable Register\)](#)" on page 141
- "[:MTERegister\[:EVENT\] \(Mask Test Event Event Register\)](#)" on page 143
- "[:MTESt:AMASK:CREAtE](#)" on page 323
- "[:MTESt:AMASK:{SAVE | STORe}](#)" on page 599
- "[:MTESt:AMASK:SOURce](#)" on page 324
- "[:MTESt:AMASK:UNITS](#)" on page 325
- "[:MTESt:AMASK:XDELta](#)" on page 326
- "[:MTESt:AMASK:YDELta](#)" on page 327
- "[:MTESt:AVERage](#)" on page 600
- "[:MTESt:AVERage:COUNT](#)" on page 601
- "[:MTESt:COUNT:FWAVeforms](#)" on page 328

- "[:MTESt:COUNT:RESet](#)" on page 329
  - "[:MTESt:COUNT:TIME](#)" on page 330
  - "[:MTESt:COUNT:WAVeforms](#)" on page 331
  - "[:MTESt:DATA](#)" on page 332
  - "[:MTESt:DELeTe](#)" on page 333
  - "[:MTESt:ENABLE](#)" on page 334
  - "[:MTESt:LOAD](#)" on page 602
  - "[:MTESt:LOCK](#)" on page 335
  - "[:MTESt:OUTPut](#)" on page 336
  - "[:MTESt:RMODE](#)" on page 337
  - "[:MTESt:RMODE:FACTion:PRINT](#)" on page 338
  - "[:MTESt:RMODE:FACTion:SAVE](#)" on page 339
  - "[:MTESt:RMODE:FACTion:STOP](#)" on page 340
  - "[:MTESt:RMODE:SIGMa](#)" on page 341
  - "[:MTESt:RMODE:TIME](#)" on page 342
  - "[:MTESt:RMODE:WAVeforms](#)" on page 343
  - "[:MTESt:RUMode](#)" on page 603
  - "[:MTESt:RUMode:SOFailure](#)" on page 604
  - "[:MTESt:SCALE:BIND](#)" on page 344
  - "[:MTESt:SCALE:X1](#)" on page 345
  - "[:MTESt:SCALE:XDELta](#)" on page 346
  - "[:MTESt:SCALE:Y1](#)" on page 347
  - "[:MTESt:SCALE:Y2](#)" on page 348
  - "[:MTESt:SOURce](#)" on page 349
  - "[:MTESt:{STARt | STOP}](#)" on page 605
  - "[:MTESt:TITLe](#)" on page 350
  - "[:MTESt:TRIGGER:SOURce](#)" on page 606
- N**
- NREject, "[:TRIGGER:NREject](#)" on page 418
  - NWIDth, "[:MEASure:NWIDth](#)" on page 284
- O**
- OFFSet Commands:
    - "[:CHANnel<n>:OFFSet](#)" on page 198
    - "[:FUNCTION:OFFSet](#)" on page 236
    - "[:\\*OPC \(Operation Complete\)](#)" on page 108
    - "[:OPEE \(Operation Status Enable Register\)](#)" on page 145

- OPERation Commands:
    - "[:FUNCTION:GOFT:OPERation](#)" on page 233
    - "[:FUNCTION:OPERation](#)" on page 237
  - "[:OPERegister:CONDITION \(Operation Status Condition Register\)](#)" on page 147
  - "[:OPERegister\[:EVENT\] \(Operation Status Event Register\)](#)" on page 149
  - "[\\*OPT \(Option Identification\)](#)" on page 109
  - OUTPut Commands:
    - "[:CALibrate:OUTPut](#)" on page 183
    - "[:MTEST:OUTPut](#)" on page 336
  - OVERload, "[:SBUS:CAN:COUNT:OVERload](#)" on page 376
  - OVERshoot, "[:MEASure:OVERshoot](#)" on page 285
  - "[:OVLenable \(Overload Event Enable Register\)](#)" on page 151
  - "[:OVLRegister \(Overload Event Register\)](#)" on page 153
- P**
- PAlette Commands:
    - "[:HARDcopy:PALETTE](#)" on page 253
    - "[:SAVE:IMAGE:PALETTE](#)" on page 365
  - PARity Commands:
    - "[:SBUS:LIN:PARITY](#)" on page 382
    - "[:TRIGger:UART:PARITY](#)" on page 493
  - PATtern Commands:
    - "[:TRIGger:CAN:PATTERn:DATA](#)" on page 424
    - "[:TRIGger:CAN:PATTERn:DATA:LENGth](#)" on page 425
    - "[:TRIGger:CAN:PATTERn:ID](#)" on page 426
    - "[:TRIGger:CAN:PATTERn:ID:MODE](#)" on page 427
    - "[:TRIGger:DURation:PATTERn](#)" on page 436
    - "[:TRIGger:IIC:PATTERn:ADDResS](#)" on page 454
    - "[:TRIGger:IIC:PATTERn:DATA](#)" on page 455
    - "[:TRIGger:IIC:PATTERn:DATA2](#)" on page 456
    - "[:TRIGger:PATTERn](#)" on page 419
    - "[:TRIGger:SPI:PATTERn:DATA](#)" on page 474
    - "[:TRIGger:SPI:PATTERn:WIDTH](#)" on page 475
  - PDRiver, "[:HARDcopy:PDRIVER](#)" on page 584
  - PERiod, "[:MEASure:PERiod](#)" on page 287
  - PERSistence, "[:DISPLAY:PERSISTence](#)" on page 215

- PHASe, "[:MEASure:PHASe](#)" on page 288
- PMODE, "[:CHANnel<n>:PMODE](#)" on page 571
- POINts Commands:
  - "[:ACQuire:POINts](#)" on page 170
  - "[:WAVeform:POINts](#)" on page 512
  - "[:WAVeform:POINts:MODE](#)" on page 514
- POLarity Commands:
  - "[:TRIGger:GLITch:POLarity](#)" on page 449
  - "[:TRIGger:TV:POLarity](#)" on page 482
  - "[:TRIGger:UART:POLarity](#)" on page 494
- POSition Commands:
  - "[:TIMEbase:POSITION](#)" on page 403
  - "[:TIMEbase:WINDOW:POSITION](#)" on page 408
- PREamble, "[:WAVeform:PREamble](#)" on page 516
- PREShoot, "[:MEASure:PREshoot](#)" on page 289
- PRInt, "[:MTEST:RMODE:FACTion:PRInt](#)" on page 338
- ":PRINT" on page 155
- ":PRINT?" on page 607
- PRINTER, "[:HARDcopy:PRINTER:LIST](#)" on page 254
- PROBe Commands:
  - "[:CHANnel<n>:PROBe](#)" on page 199
  - "[:EXTernal:PROBe](#)" on page 222
- PROTection Commands:
  - "[:CHANnel<n>:PROTection](#)" on page 203
  - "[:EXTernal:PROTection](#)" on page 225
  - "[:SYSTem:PROTection:LOCK](#)" on page 396
- Pulse Width (GLITch), "[:TRIGger:GLITch Commands](#)" on page 445
- PWD Commands:
  - "[:RECall:PWD](#)" on page 355
  - "[:SAVE:PWD](#)" on page 367
- PWIDth, "[:MEASure:PWIDth](#)" on page 290
- Q** • QUALifier Commands:
  - "[:TRIGger:DURATION:QUALifier](#)" on page 437
  - "[:TRIGger:GLITch:QUALifier](#)" on page 450
  - "[:TRIGger:IIC:TRIGger:QUALifier](#)" on page 459

- "[:TRIGger:UART:QUALifier](#)" on page 495
- R** • RANGE Commands:
  - "[:CHANnel<n>:RANGE](#)" on page 204
  - "[:EXTernal:RANGE](#)" on page 226
  - "[:FUNCTION:RANGE](#)" on page 238
  - "[:TIMEbase:RANGE](#)" on page 404
  - "[:TIMEbase:WINDOW:RANGE](#)" on page 409
  - "[:TRIGger:DURATION:RANGE](#)" on page 438
  - "[:TRIGger:GLITCH:RANGE](#)" on page 451
- "[\\*RCL \(Recall\)](#)" on page 110
- "[:RECall:FILEname](#)" on page 352
- "[:RECall:IMAGe\[:STARt\]](#)" on page 353
- "[:RECall:MASK\[:STARt\]](#)" on page 354
- "[:RECall:PWD](#)" on page 355
- "[:RECall:SETup\[:STARt\]](#)" on page 356
- REFERENCE Commands:
  - "[:FUNCTION:REFERENCE](#)" on page 239
  - "[:TIMEbase:REFERENCE](#)" on page 405
- REJECT, "[:TRIGger\[:EDGE\]:REJECT](#)" on page 442
- RESET Commands:
  - "[:MEASure:STATistics:RESET](#)" on page 301
  - "[:MTEST:COUNT:RESET](#)" on page 329
  - "[:SBUS:CAN:COUNT:RESET](#)" on page 377
  - "[:SBUS:UART:COUNT:RESET](#)" on page 387
- RESULTS, "[:MEASure:RESULTS](#)" on page 291
- RISETIME, "[:MEASure:RISETIME](#)" on page 294
- RMODE Commands:
  - "[:MTEST:RMODE](#)" on page 337
  - "[:MTEST:RMODE:FACTion:PRINT](#)" on page 338
  - "[:MTEST:RMODE:FACTion:SAVE](#)" on page 339
  - "[:MTEST:RMODE:FACTion:STOP](#)" on page 340
  - "[:MTEST:RMODE:SIGMa](#)" on page 341
  - "[:MTEST:RMODE:TIME](#)" on page 342
  - "[:MTEST:RMODE:WAVeforms](#)" on page 343
- "["Root \(\) Commands"](#)" on page 122

- ["\\*RST \(Reset\)" on page 111](#)
- RUMode Commands:
  - [":MTESt:RUMode" on page 603](#)
  - [":MTESt:RUMode:SOFailure" on page 604](#)
- [":RUN" on page 156](#)
- RX, [":TRIGger:UART:SOURce:RX" on page 496](#)
- RXFRAMES, [":SBUS:UART:COUNT:RXFRAMES" on page 388](#)
- S**
  - SAMPLepoint Commands:
    - [":TRIGger:CAN:SAMPLepoint" on page 428](#)
    - [":TRIGger:LIN:SAMPLepoint" on page 464](#)
  - ["\\*SAV \(Save\)" on page 114](#)
  - SAVE Commands:
    - [":MTESt:AMASk:{SAVE | STORe}" on page 599](#)
    - [":MTESt:RMODE:FACTion:SAVE" on page 339](#)
    - [":SAVE:FILEname" on page 359](#)
    - [":SAVE:IMAGe:AREA" on page 361](#)
    - [":SAVE:IMAGe:FACTors" on page 362](#)
    - [":SAVE:IMAGe:FORMAT" on page 363](#)
    - [":SAVE:IMAGe:INKSaver" on page 364](#)
    - [":SAVE:IMAGe:PAlette" on page 365](#)
    - [":SAVE:IMAGe\[:STARt\]" on page 360](#)
    - [":SAVE:MASK\[:STARt\]" on page 366](#)
    - [":SAVE:PWD" on page 367](#)
    - [":SAVE:SETup\[:STARt\]" on page 368](#)
    - [":SAVE:WAVeform:FORMAT" on page 370](#)
    - [":SAVE:WAVeform:LENGTH" on page 371](#)
    - [":SAVE:WAVeform:SEGmented" on page 372](#)
    - [":SAVE:WAVeform\[:STARt\]" on page 369](#)
    - [":SBUS:CAN:COUNT:ERROR" on page 375](#)
    - [":SBUS:CAN:COUNT:OVERload" on page 376](#)
    - [":SBUS:CAN:COUNT:RESET" on page 377](#)
    - [":SBUS:CAN:COUNT:TOTAL" on page 378](#)
    - [":SBUS:CAN:COUNT:UTILization" on page 379](#)
    - [":SBUS:DISPLAY" on page 380](#)
    - [":SBUS:IIC:ASIZE" on page 381](#)

- "[:SBUS:LIN:PARity](#)" on page 382
- "[:SBUS:MODE](#)" on page 383
- "[:SBUS:SPI:WIDTH](#)" on page 384
- "[:SBUS:UART:BASE](#)" on page 385
- "[:SBUS:UART:COUNT:ERRor](#)" on page 386
- "[:SBUS:UART:COUNT:RESET](#)" on page 387
- "[:SBUS:UART:COUNT:RXFRAMES](#)" on page 388
- "[:SBUS:UART:COUNT:TXFRAMES](#)" on page 389
- "[:SBUS:UART:FRAMing](#)" on page 390
- SCALe Commands:
  - "[:CHANnel<n>:SCALe](#)" on page 205
  - "[:FUNCTION:SCALe](#)" on page 240
  - "[:MTEST:SCALe:BIND](#)" on page 344
  - "[:MTEST:SCALe:X1](#)" on page 345
  - "[:MTEST:SCALe:XDELta](#)" on page 346
  - "[:MTEST:SCALe:Y1](#)" on page 347
  - "[:MTEST:SCALe:Y2](#)" on page 348
  - "[:TIMEbase:SCALe](#)" on page 406
  - "[:TIMEbase:WINDOW:SCALe](#)" on page 410
- SCRatch, "[:MEASure:SCRatch](#)" on page 586
- SDEViation, "[:MEASure:SDEViation](#)" on page 295
- "[:SERial](#)" on page 157
- SEGmented Commands:
  - "[:ACQuire:SEGmented:ANALyze](#)" on page 171
  - "[:ACQuire:SEGmented:COUNT](#)" on page 172
  - "[:ACQuire:SEGmented:INDEX](#)" on page 173
  - "[:SAVE:WAVEform:SEGmented](#)" on page 372
  - "[:WAVEform:SEGmented:COUNT](#)" on page 519
  - "[:WAVEform:SEGmented:TTAG](#)" on page 520
- SETup Commands:
  - "[:RECall:SETup\[:STARt\]](#)" on page 356
  - "[:SAVE:SETup\[:STARt\]](#)" on page 368
  - "[:SYSTem:SETup](#)" on page 397
- SHOW, "[:MEASure:SHOW](#)" on page 296
- SIGMa, "[:MTEST:RMODE:SIGMa](#)" on page 341

- SIGNal Commands:
  - "[:TRIGger:CAN:SIGNAl:BAUDrate](#)" on page 429
  - "[:TRIGger:CAN:SIGNAl:DEFinition](#)" on page 611
  - "[:TRIGger:LIN:SIGNAl:BAUDrate](#)" on page 465
  - "[:TRIGger:LIN:SIGNAl:DEFinition](#)" on page 612
- "[:SINGle](#)" on page 158
- SKEW, "[:CHANnel<n>:PROBe:SKEW](#)" on page 201
- SLOPe Commands:
  - "[:TRIGger\[:EDGE\]:SLOPe](#)" on page 443
  - "[:TRIGger:SPI:CLOCK:SLOPe](#)" on page 471
- SOFailure, "[:MTESt:RUMode:SOFailure](#)" on page 604
- SOURce Commands:
  - "[:DISPlay:SOURce](#)" on page 216
  - "[:FUNCTION:SOURce](#)" on page 576
  - "[:MEASure:SOURce](#)" on page 297
  - "[:MTESt:AMASK:SOURce](#)" on page 324
  - "[:MTESt:SOURce](#)" on page 349
  - "[:MTESt:TRIGger:SOURce](#)" on page 606
  - "[:TRIGger:CAN:SOURce](#)" on page 430
  - "[:TRIGger:GLITch:SOURce](#)" on page 452
  - "[:TRIGger:IIC\[:SOURce\]:CLOCK](#)" on page 457
  - "[:TRIGger:IIC\[:SOURce\]:DATA](#)" on page 458
  - "[:TRIGger:LIN:SOURce](#)" on page 466
  - "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
  - "[:TRIGger:SPI:SOURce:DATA](#)" on page 477
  - "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478
  - "[:TRIGger:TV:SOURce](#)" on page 483
  - "[:TRIGger:UART:SOURce:RX](#)" on page 496
  - "[:TRIGger:UART:SOURce:TX](#)" on page 497
  - "[:WAveform:SOURce](#)" on page 521
  - "[:WAveform:SOURce:SUBSource](#)" on page 525
- SOURce1 Commands:
  - "[:FUNCTION:GOFT:SOURce1](#)" on page 234
  - "[:FUNCTION:SOURce1](#)" on page 241
- SOURce2 Commands:

- [":FUNCTION:GOFT:SOURce2" on page 235](#)
- [":FUNCTION:SOURce2" on page 242](#)
- SPAN, [":FUNCTION:SPAN" on page 243](#)
- SPI Commands:
  - [":SBUS:SPI:WIDTh" on page 384](#)
  - [":TRIGger:SPI Commands" on page 470](#)
- SRATE, [":ACQuire:SRATE" on page 176](#)
- [":\\*SRE \(Service Request Enable\)" on page 115](#)
- STANdard Commands:
  - [":TRIGger:LIN:STANdard" on page 467](#)
  - [":TRIGger:TV:STANdard" on page 484](#)
- STARt Commands:
  - [":CALibrate:STARt" on page 184](#)
  - [":HARDcopy:STARt" on page 255](#)
  - [":MTESt:{STARt | STOP}" on page 605](#)
  - [":RECall:IMAGe\[:STARt\]" on page 353](#)
  - [":RECall:MASK\[:STARt\]" on page 354](#)
  - [":RECall:SETup\[:STARt\]" on page 356](#)
  - [":SAVE:IMAGe\[:STARt\]" on page 360](#)
  - [":SAVE:MASK\[:STARt\]" on page 366](#)
  - [":SAVE:SETup\[:STARt\]" on page 368](#)
  - [":SAVE:WAVEform\[:STARt\]" on page 369](#)
- STATistics Commands:
  - [":MEASure:STATistics" on page 299](#)
  - [":MEASure:STATistics:INCReement" on page 300](#)
  - [":MEASure:STATistics:RESet" on page 301](#)
- STATus Commands:
  - [":CALibrate:STATus" on page 185](#)
  - [":STATus" on page 159](#)
- [":\\*STB \(Read Status Byte\)" on page 117](#)
- STOP Commands:
  - [":MTESt:RMODE:FACtion:STOP" on page 340](#)
  - [":MTESt:{STARt | STOP}" on page 605](#)
- [":STOP" on page 160](#)
- STORe, [":MTESt:AMASK:{SAVE | STORe}" on page 599](#)

- SUBSource, "[:WAVEform:SOURce:SUBSource](#)" on page 525
  - SWEep, "[:TRIGger:SWEep](#)" on page 421
  - SWITch, "[:CALibrate:SWITch](#)" on page 186
  - SYNCbreak, "[:TRIGger:LIN:SYNCbreak](#)" on page 468
  - ":SYSTem:DATE" on page 392
  - ":SYSTem:DSP" on page 393
  - ":SYSTem:ERRor" on page 394
  - ":SYSTem:LOCK" on page 395
  - ":SYSTem:SETup" on page 397
  - ":SYSTem:TIME" on page 399
- T**
- TDELta, "[:MEASure:TDELta](#)" on page 587
  - TEDGe, "[:MEASure:TEDGE](#)" on page 302
  - TEMPerature, "[:CALibrate:TEMPerature](#)" on page 187
  - ":TER (Trigger Event Register)" on page 161
  - THReholds, "[:MEASure:THReholds](#)" on page 588
  - TIME Commands:
    - ":CALibrate:TIME" on page 188
    - ":MTEST:COUNt:TIME" on page 330
    - ":MTEST:RMODE:TIME" on page 342
    - ":SYSTem:TIME" on page 399
  - ":TIMEbase:DELay" on page 609
  - ":TIMEbase:MODE" on page 402
  - ":TIMEbase:POSition" on page 403
  - ":TIMEbase:RANGE" on page 404
  - ":TIMEbase:REFERENCE" on page 405
  - ":TIMEbase:SCALe" on page 406
  - ":TIMEbase:VERNier" on page 407
  - ":TIMEbase:WINDOW:POSITION" on page 408
  - ":TIMEbase:WINDOW:RANGE" on page 409
  - ":TIMEbase:WINDOW:SCALe" on page 410
  - TIMEout, "[:TRIGger:SPI:CLOCk:TIMEout](#)" on page 472
  - TITLE, "[:MTEST:TITLE](#)" on page 350
  - TMAX, "[:MEASure:TMAX](#)" on page 589
  - TMIN, "[:MEASure:TMIN](#)" on page 590
  - TOTal, "[:SBUS:CAN:COUNt:TOTal](#)" on page 378

- ["\\*TRG \(Trigger\)" on page 119](#)
- **TRIGger Commands:**
  - [":MTEST:TRIGger:SOURce" on page 606](#)
  - [":TRIGger:CAN:TRIGger" on page 431](#)
  - [":TRIGger:IIC:TRIGger:QUALifier" on page 459](#)
  - [":TRIGger:IIC:TRIGger\[:TYPE\]" on page 460](#)
  - [":TRIGger:LIN:TRIGger" on page 469](#)
  - [":TRIGger:HFReject" on page 415](#)
  - [":TRIGger:HOLDoff" on page 416](#)
  - [":TRIGger:MODE" on page 417](#)
  - [":TRIGger:NREject" on page 418](#)
  - [":TRIGger:PATTern" on page 419](#)
  - [":TRIGger:SWEep" on page 421](#)
  - [":TRIGger:CAN:ACKNowledge" on page 610](#)
  - [":TRIGger:CAN:PATTern:DATA" on page 424](#)
  - [":TRIGger:CAN:PATTern:DATA:LENGth" on page 425](#)
  - [":TRIGger:CAN:PATTern:ID" on page 426](#)
  - [":TRIGger:CAN:PATTern:ID:MODE" on page 427](#)
  - [":TRIGger:CAN:SAMPLEpoint" on page 428](#)
  - [":TRIGger:CAN:SIGNal:BAUDrate" on page 429](#)
  - [":TRIGger:CAN:SIGNal:DEFinition" on page 611](#)
  - [":TRIGger:CAN:SOURce" on page 430](#)
  - [":TRIGger:CAN:TRIGger" on page 431](#)
  - [":TRIGger:DURation:GREaterthan" on page 434](#)
  - [":TRIGger:DURation:LESSthan" on page 435](#)
  - [":TRIGger:DURation:PATTern" on page 436](#)
  - [":TRIGger:DURation:QUALifier" on page 437](#)
  - [":TRIGger:DURation:RANGE" on page 438](#)
  - [":TRIGger\[:EDGE\]:COUpling" on page 440](#)
  - [":TRIGger\[:EDGE\]:LEVel" on page 441](#)
  - [":TRIGger\[:EDGE\]:REJect" on page 442](#)
  - [":TRIGger\[:EDGE\]:SLOPe" on page 443](#)
  - [":TRIGger\[:EDGE\]:SOURce" on page 444](#)
  - [":TRIGger:GLITch:GREaterthan" on page 446](#)
  - [":TRIGger:GLITch:LESSthan" on page 447](#)

- "[:TRIGger:GLITch:LEVel](#)" on page 448
- "[:TRIGger:GLITch:POLarity](#)" on page 449
- "[:TRIGger:GLITch:QUALifier](#)" on page 450
- "[:TRIGger:GLITch:RANGE](#)" on page 451
- "[:TRIGger:GLITch:SOURce](#)" on page 452
- "[:TRIGger:HFReject](#)" on page 415
- "[:TRIGger:HOLDoff](#)" on page 416
- "[:TRIGger:IIC:PATTern:ADDResS](#)" on page 454
- "[:TRIGger:IIC:PATTern:DATA](#)" on page 455
- "[:TRIGger:IIC:PATTern:DATA2](#)" on page 456
- "[:TRIGger:IIC\[:SOURce\]:CLOCK](#)" on page 457
- "[:TRIGger:IIC\[:SOURce\]:DATA](#)" on page 458
- "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 459
- "[:TRIGger:LIN:SIGNal:BAUDrate](#)" on page 465
- "[:TRIGger:LIN:SIGNal:DEFinition](#)" on page 612
- "[:TRIGger:LIN:SOURce](#)" on page 466
- "[:TRIGger:LIN:TRIGger](#)" on page 469
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:NREject](#)" on page 418
- "[:TRIGger:PATTern](#)" on page 419
- "[:TRIGger:SPI:CLOCK:SLOPe](#)" on page 471
- "[:TRIGger:SPI:CLOCK:TIMEout](#)" on page 472
- "[:TRIGger:SPI:FRAMing](#)" on page 473
- "[:TRIGger:SPI:PATTern:DATA](#)" on page 474
- "[:TRIGger:SPI:PATTern:WIDTh](#)" on page 475
- "[:TRIGger:SPI:SOURce:CLOCK](#)" on page 476
- "[:TRIGger:SPI:SOURce:DATA](#)" on page 477
- "[:TRIGger:SPI:SOURce:FRAMe](#)" on page 478
- "[:TRIGger:SWEep](#)" on page 421
- "[:TRIGger:TV:LINE](#)" on page 480
- "[:TRIGger:TV:MODE](#)" on page 481
- "[:TRIGger:TV:POLarity](#)" on page 482
- "[:TRIGger:TV:SOURce](#)" on page 483
- "[:TRIGger:TV:STANDARD](#)" on page 484

- "[:TRIGger:TV:TMode](#)" on page 613
  - "[:TRIGger:UART:BASE](#)" on page 487
  - "[:TRIGger:UART:BAUDrate](#)" on page 488
  - "[:TRIGger:UART:BITorder](#)" on page 489
  - "[:TRIGger:UART:BURSt](#)" on page 490
  - "[:TRIGger:UART:DATA](#)" on page 491
  - "[:TRIGger:UART:IDLE](#)" on page 492
  - "[:TRIGger:UART:PARity](#)" on page 493
  - "[:TRIGger:UART:POLarity](#)" on page 494
  - "[:TRIGger:UART:QUALifier](#)" on page 495
  - "[:TRIGger:UART:SOURce:RX](#)" on page 496
  - "[:TRIGger:UART:SOURce:TX](#)" on page 497
  - "[:TRIGger:UART:TYPE](#)" on page 498
  - "[:TRIGger:UART:WIDTh](#)" on page 499
  - "[\\*TST \(Self Test\)](#)" on page 120
  - TSTArt, "[:MEASure:TSTArt](#)" on page 591
  - TSTOP, "[:MEASure:TSTOP](#)" on page 592
  - TTAG, "[:WAVeform:SEGmented:TTAG](#)" on page 520
  - TV, "[:TRIGger:TV Commands](#)" on page 479
  - TVALue, "[:MEASure:TVALue](#)" on page 304
  - TVOLT, "[:MEASure:TVOLT](#)" on page 593
  - TX, "[:TRIGger:UART:SOURce:TX](#)" on page 497
  - TXFRAMES, "[:SBUS:UART:COUNt:TXFRAMES](#)" on page 389
  - TYPE Commands:
    - "[:ACQuire:TYPE](#)" on page 177
    - "[:WAVeform:TYPE](#)" on page 526
    - "[:TRIGger:IIC:TRIGger\[:TYPE\]](#)" on page 460
    - "[:TRIGger:UART:TYPE](#)" on page 498
- U** • UART Commands:
- "[:SBUS:UART:BASE](#)" on page 385
  - "[:SBUS:UART:COUNt:ERRor](#)" on page 386
  - "[:SBUS:UART:COUNt:RESet](#)" on page 387
  - "[:SBUS:UART:COUNt:RXFRAMES](#)" on page 388
  - "[:SBUS:UART:COUNt:TXFRAMES](#)" on page 389
  - "[:SBUS:UART:FRAmIng](#)" on page 390

- ":TRIGger:UART:BASE" on page 487
  - ":TRIGger:UART:BAUDrate" on page 488
  - ":TRIGger:UART:BITorder" on page 489
  - ":TRIGger:UART:BURSt" on page 490
  - ":TRIGger:UART:DATA" on page 491
  - ":TRIGger:UART:IDLE" on page 492
  - ":TRIGger:UART:PARity" on page 493
  - ":TRIGger:UART:POLarity" on page 494
  - ":TRIGger:UART:QUALifier" on page 495
  - ":TRIGger:UART:SOURce:RX" on page 496
  - ":TRIGger:UART:SOURce:TX" on page 497
  - ":TRIGger:UART:TYPE" on page 498
  - ":TRIGger:UART:WIDTH" on page 499
  - UNITS Commands:
    - ":CHANnel<n>:UNITS" on page 206
    - ":EXTernal:UNITS" on page 227
    - ":MTEST:AMASK:UNITS" on page 325
  - UNSIGNED, ":WAVEform:UNSIGNED" on page 527
  - UPPer, ":MEASure:UPPer" on page 595
  - UTILization, ":SBUS:CAN:COUNT:UTILization" on page 379
- V**
- VAMPLitude, ":MEASure:VAMPLitude" on page 306
  - VAverage, ":MEASure:VAverage" on page 307
  - VBASe, ":MEASure:VBASe" on page 308
  - VDELta, ":MEASure:VDELta" on page 596
  - VECTors, ":DISPLAY:VECTors" on page 217
  - VERNier, ":CHANnel<n>:VERNier" on page 207
  - ":VIEW" on page 162
  - VMAX, ":MEASure:VMAX" on page 309
  - VMIN, ":MEASure:VMIN" on page 310
  - VPP, ":MEASure:VPP" on page 311
  - VRATio, ":MEASure:VRATio" on page 312
  - VRMS, ":MEASure:VRMS" on page 313
  - VSTArt, ":MEASure:VSTArt" on page 597
  - VSTOP, ":MEASure:VSTOP" on page 598
  - VTIMe, ":MEASure:VTIMe" on page 314

- VTOP, "[:MEASure:VTOP](#)" on page 315
- W** • "[\\*WAI \(Wait To Continue\)](#)" on page 121
  - WAveform Commands:
    - "[:SAVE:WAveform:FORMat](#)" on page 370
    - "[:SAVE:WAveform:LENGth](#)" on page 371
    - "[:SAVE:WAveform\[:STARt\]](#)" on page 369
  - "[:WAveform:BYTeorder](#)" on page 507
  - "[:WAveform:COUNt](#)" on page 508
  - "[:WAveform:DATA](#)" on page 509
  - "[:WAveform:FORMat](#)" on page 511
  - "[:WAveform:POINts](#)" on page 512
  - "[:WAveform:POINts:MODE](#)" on page 514
  - "[:WAveform:PREamble](#)" on page 516
  - "[:WAveform:SEGmented:COUNt](#)" on page 519
  - "[:WAveform:SEGmented:TTAG](#)" on page 520
  - "[:WAveform:SOURce](#)" on page 521
  - "[:WAveform:SOURce:SUBSource](#)" on page 525
  - "[:WAveform:TYPE](#)" on page 526
  - "[:WAveform:UNSIGNED](#)" on page 527
  - "[:WAveform:VIEW](#)" on page 528
  - "[:WAveform:XINCrement](#)" on page 529
  - "[:WAveform:XORigin](#)" on page 530
  - "[:WAveform:XREFERENCE](#)" on page 531
  - "[:WAveform:YINCrement](#)" on page 532
  - "[:WAveform:YORigin](#)" on page 533
  - "[:WAveform:YREFERENCE](#)" on page 534
- WAveforms Commands:
  - "[:MTEST:COUNt:WAveforms](#)" on page 331
  - "[:MTEST:RMODE:WAveforms](#)" on page 343
- WIDTh Commands:
  - "[:SBUS:SPI:WIDTh](#)" on page 384
  - "[:TRIGger:SPI:PATTern:WIDTh](#)" on page 475
  - "[:TRIGger:UART:WIDTh](#)" on page 499
- WINDOW, "[:FUNCTION:WINDOW](#)" on page 244

- X**
  - X1, "[:MTEST:SCALE:X1](#)" on page 345
  - X1Position, "[:MARKer:X1Position](#)" on page 259
  - X1Y1source, "[:MARKer:X1Y1source](#)" on page 260
  - X2Position, "[:MARKer:X2Position](#)" on page 261
  - X2Y2source, "[:MARKer:X2Y2source](#)" on page 262
  - XDELta Commands:
    - "[:MARKer:XDELta](#)" on page 263
    - "[:MTEST:AMASK:XDELta](#)" on page 326
    - "[:MTEST:SCALE:XDELta](#)" on page 346
  - XINCrement, "[:WAVEform:XINCrement](#)" on page 529
  - XMAX, "[:MEASure:XMAX](#)" on page 316
  - XMIN, "[:MEASure:XMIN](#)" on page 317
  - XORigin, "[:WAVEform:XORigin](#)" on page 530
  - XREFERENCE, "[:WAVEform:XREFERENCE](#)" on page 531
- Y**
  - Y1, "[:MTEST:SCALE:Y1](#)" on page 347
  - Y1Position, "[:MARKer:Y1Position](#)" on page 264
  - Y2, "[:MTEST:SCALE:Y2](#)" on page 348
  - Y2Position, "[:MARKer:Y2Position](#)" on page 265
  - YDELta Commands:
    - "[:MARKer:YDELta](#)" on page 266
    - "[:MTEST:AMASK:YDELta](#)" on page 327
  - YINCrement, "[:WAVEform:YINCrement](#)" on page 532
  - YORigin, "[:WAVEform:YORigin](#)" on page 533
  - YREFERENCE, "[:WAVEform:YREFERENCE](#)" on page 534



7

## 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 658).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see <a href="#">page 192</a> )	
ANALog<n>:COUpling	:CHANnel<n>:COUpling (see <a href="#">page 193</a> )	
ANALog<n>:INVert	:CHANnel<n>:INVert (see <a href="#">page 196</a> )	
ANALog<n>:LABEL	:CHANnel<n>:LABEL (see <a href="#">page 197</a> )	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see <a href="#">page 198</a> )	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see <a href="#">page 199</a> )	
ANALog<n>:PMODe	none	
ANALog<n>:RANGE	:CHANnel<n>:RANGE (see <a href="#">page 204</a> )	
:CHANnel:LABEL (see <a href="#">page 568</a> )	:CHANnel<n>:LABEL (see <a href="#">page 197</a> )	
:CHANnel2:SKEW (see <a href="#">page 569</a> )	:CHANnel<n>:PROBe:SKEW (see <a href="#">page 201</a> )	
:CHANnel<n>:INPut (see <a href="#">page 570</a> )	:CHANnel<n>:IMPedance (see <a href="#">page 195</a> )	
:CHANnel<n>:PMODe (see <a href="#">page 571</a> )	none	
:DISPlay:CONNect (see <a href="#">page 572</a> )	:DISPlay:VECTors (see <a href="#">page 217</a> )	
:ERASe (see <a href="#">page 573</a> )	:CDISplay (see <a href="#">page 131</a> )	



## 7 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:EXternal:INPut (see page 574)	:EXternal:IMPedance (see page 221)	
:EXternal:PMODe (see page 575)	none	
FUNCTION1, FUNCTION2	:FUNCTION Commands (see page 228)	ADD not included
:FUNCTION:SOURce (see page 576)	:FUNCTION:SOURce1 (see page 241)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNCTION:VIEW (see page 577)	:FUNCTION:DISPlay (see page 232)	
:HARDcopy:DESTination (see page 578)	:HARDcopy:FILEname (see page 580)	
:HARDcopy:DEvice (see page 579)	:HARDcopy:FORMAT (see page 581)	PLOTter, THINKjet not supported; TIF, BMP, CSV, SEiko added
:HARDcopy:FILEname (see page 580)	:RECall:FILEname (see page 352) :SAVE:FILEname (see page 352)	
:HARDcopy:FORMAT (see page 581)	:HARDcopy:APRinter (see page 248) :SAVE:IMAGe:FORMAT (see page 363) :SAVE:WAVEform:FORMAT (see page 370)	
:HARDcopy:GRAYscale (see page 582)	:HARDcopy:PAlette (see page 253)	
:HARDcopy:IGColors (see page 583)	:HARDcopy:INKSaver (see page 251)	
:HARDcopy:PDRiver (see page 584)	:HARDcopy:APRinter (see page 248)	
:MEASure:LOWER (see page 585)	:MEASure:DEFine:THResholds (see page 276)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see page 586)	:MEASure:CLEar (see page 274)	
:MEASure:TDELta (see page 587)	:MARKer:XDELta (see page 263)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:THResholds (see page 588)	:MEASure:DEFine:THResholds (see page 276)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 589)	:MEASure:XMAX (see page 316)	
:MEASure:TMIN (see page 590)	:MEASure:XMIN (see page 317)	
:MEASure:TSTArt (see page 591)	:MARKer:X1Position (see page 259)	
:MEASure:TSTOP (see page 592)	:MARKer:X2Position (see page 261)	
:MEASure:TVOLt (see page 593)	:MEASure:TVALue (see page 304)	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see page 595)	:MEASure:DEFine:THResholds (see page 276)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 596)	:MARKer:YDELta (see page 266)	
:MEASure:VSTArt (see page 597)	:MARKer:Y1Position (see page 264)	
:MEASure:VSTOP (see page 598)	:MARKer:Y2Position (see page 265)	
:MTEST:AMASK{:SAVE   STORe} (see page 599)	:SAVE:MASK[:STARt] (see page 366)	
:MTEST:AVERage (see page 600)	:ACQuire:TYPE AVERage (see page 177)	
:MTEST:AVERage:COUNT (see page 601)	:ACQuire:COUNt (see page 167)	
:MTEST:LOAD (see page 602)	:RECall:MASK[:STARt] (see page 354)	
:MTEST:RUMode (see page 603)	:MTEST:RMODE (see page 337)	
:MTEST:RUMode:SOFailure (see page 604)	:MTEST:RMODE:FACTion:STOP (see page 340)	
:MTEST{:STARt   STOP} (see page 605)	:RUN (see page 156) or :STOP (see page 160)	
:MTEST:TRIGger:SOURce (see page 606)	:TRIGger Commands (see page 411)	There are various commands for setting the source with different types of triggers.

Obsolete Command	Current Command Equivalent	Behavior Differences
:PRINt? (see <a href="#">page 607</a> )	:DISPlay:DATA? (see <a href="#">page 211</a> )	
:TIMEbase:DELay (see <a href="#">page 609</a> )	:TIMEbase:POSIon (see <a href="#">page 403</a> ) or :TIMEbase:WINDOW:POSIon (see <a href="#">page 408</a> )	TIMEbase:POSIon is position value of main time base; TIMEbase:WINDOW:POSIon is position value of zoomed (delayed) time base window.
:TRIGger:CAN:ACKNowledge (see <a href="#">page 610</a> )	none	
:TRIGger:CAN:SIGNAl:DEFinition (see <a href="#">page 611</a> )	none	
:TRIGger:LIN:SIGNAl:DEFinition (see <a href="#">page 612</a> )	none	
:TRIGger:TV:TMode (see <a href="#">page 613</a> )	:TRIGger:TV:MODE (see <a href="#">page 481</a> )	

**Discontinued Commands**

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 5000 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 <a href="#">page 215</a> )	
CHANnel:MATH	:FUNCTION:OPERation (see <a href="#">page 237</a> )	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see <a href="#">page 203</a> )	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:GRID	none	
DISPlay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSITION	none	

Discontinued Command	Current Command Equivalent	Comments
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCTION:MOVE	none	
FUNCTION:PEAKs	none	
HARDcopy:ADDResS	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see <a href="#">page 120</a> )	
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 <a href="#">page 481</a> )	
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
VAUToscale	none	

**Discontinued Parameters**

Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 5000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

## :CHANnel:LABEL

 (see [page 658](#))

**Command Syntax**    :CHANnel:LABEL <source\_text><string>

    <source\_text> ::= {CHANnel1 | CHANnel2 | DIGital0,...,DIGital15}

    <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 197](#)) instead.

**Query Syntax**    :CHANnel:LABEL?

The :CHANnel:LABEL? query returns the label associated with a particular analog channel.

**Return Format**    <string><NL>

    <string> ::= quoted ASCII string

## :CHANnel2:SKEW

 (see page 658)

**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 201](#)) 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 190](#)

## :CHANnel<n>:INPut

 (see [page 658](#))

**Command Syntax**    :CHANnel<n>:INPut <impedance>

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 195](#)) 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 658)

**Command Syntax** :CHANnel<n>:PMODe <pmod value>

<pmod value> ::= {AUTo | MANual}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

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 658)

**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 217) 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 217

## :ERASe

 (see [page 658](#))

### 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 `:CDISplay` command (see [page 131](#)) instead.

---

## :EXTernal:INPut

 (see [page 658](#))

**Command Syntax**    :EXTernal:INPut <impedance>  
<impedance> ::= {ONEMeg | FIFTy}

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

### NOTE

The :EXTernal:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :EXTernal:IMPedance command (see [page 221](#)) instead.

**Query Syntax**    :EXTernal:INPut?

The :EXTernal:INPut? query returns the current input impedance setting for the external trigger.

**Return Format**    <impedance value><NL>

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

**See Also**    • "Introduction to :EXTernal Trigger Commands" on page 218  
• "Introduction to :TRIGger Commands" on page 411  
• "[:CHANnel<n>:IMPedance](#)" on page 195

## :EXTernal:PMODE

 (see page 658)

**Command Syntax** :EXTernal:PMODE <pmode value>  
<pmode value> ::= {AUTo | MANual}

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

### NOTE

The :EXTernal:PMODE command is an obsolete command provided for compatibility to previous oscilloscopes.

**Query Syntax** :EXTernal:PMODE?

The :EXTernal:PMODE? query returns AUT if an autosense probe is attached and MAN otherwise.

**Return Format** <pmode value><NL>  
<pmode value> ::= {AUT | MAN}

**:FUNCTION:SOURce**

 (see [page 658](#))

**Command Syntax**

```
:FUNCTION:SOURce <value>
<value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 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 241](#)) 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 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

**See Also**

- "[Introduction to :FUNCTION Commands](#)" on page 230
- "[:FUNCTION:OPERation](#)" on page 237

## :FUNCTION:VIEW

 (see page 658)

**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 232](#)) 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 658)

**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 580) 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 246  
• ":HARDcopy:FORMAT" on page 581

## :HARDcopy:DEvice

 (see [page 658](#))

**Command Syntax** :HARDcopy:DEvice <device>

```
<device> ::= {TIFF | GIF | BMP | LASerjet | EPSON | DESKjet  
| BWDeskjet | SEIKO}
```

The HARDcopy:DEvice command sets the hardcopy device type.

**NOTE**

BWDeskjet option refers to the monochrome Deskjet printer.

**NOTE**

The :HARDcopy:DEvice command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FORMAT command ([see page 581](#)) instead.

**Query Syntax**

```
:HARDcopy:DEvice?
```

The :HARDcopy:DEvice? query returns the selected hardcopy device type.

**Return Format**

```
<device><NL>
```

```
<device> ::= {TIFF | GIF | BMP | LAS | EPS | DESK | BWD | SEIK}
```

## :HARDcopy:FILEname

 (see [page 658](#))

**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 359](#)) and :RECall:FILEname command ([see page 352](#)) 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 246](#)  
                          • "[:HARDcopy:FORMAT](#)" on [page 581](#)

## :HARDcopy:FORMAT

 (see [page 658](#))

### Command Syntax

```
:HARDcopy:FORMAT <format>
<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCIIxy | BINary
              | PRINTER0 | PRINTER1}
```

The HARDcopy:FORMAT command sets the hardcopy format type.

PRINTER0 and PRINTER1 are only valid when printers are connected to the oscilloscope's USB ports. (The first printer connected/identified is PRINTER0 and the second is PRINTER1.)

### NOTE

The :HARDcopy:FORMAT command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:IMAGe:FORMAT (see [page 363](#)), :SAVE:WAveform:FORMAT (see [page 370](#)), and :HARDcopy:APRinter (see [page 248](#)) commands instead.

### Query Syntax

```
:HARDcopy:FORMAT?
```

The :HARDcopy:FORMAT? query returns the selected hardcopy format type.

### Return Format

```
<format><NL>
<format> ::= {BMP | BMP8 | PNG | CSV | ASC | BIN | PRIN0 | PRIN1}
```

### See Also

- "Introduction to :HARDcopy Commands" on [page 246](#)

## :HARDcopy:GRAYscale

 (see page 658)

**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 253](#)) 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 246](#)

## :HARDcopy:IGColors

 (see page 658)

**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 251) 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 246

**:HARDcopy:PDRiver**

 (see page 658)

**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 248](#)) 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 246
- "[":HARDcopy:FORMAT](#)" on page 581

## :MEASure:LOWER

 (see page 658)

### 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 276](#)) 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 272
- "[":MEASure:THresholds](#)" on page 588
- "[":MEASure:UPPer](#)" on page 595

## **:MEASure:SCRatch**

 (see [page 658](#))

**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 274](#)) instead.

---

**:MEASure:TDELta**

 (see page 658)

**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 263) instead.

**Return Format** <value><NL>

<value> ::= time difference between start and stop markers in NR3 format

**See Also**

- "Introduction to :MARKer Commands" on page 257
- "Introduction to :MEASure Commands" on page 272
- ":MARKer:X1Position" on page 259
- ":MARKer:X2Position" on page 261
- ":MARKer:XDELta" on page 263
- ":MEASure:TSTArt" on page 591
- ":MEASure:TSTOP" on page 592

## :MEASure:THResholds

 (see page 658)

**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 276) 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 272

• "[":MEASure:LOWER](#)" on page 585

• "[":MEASure:UPPer](#)" on page 595

## :MEASure:TMAX

 (see page 658)

### Command Syntax

```
:MEASure:TMAX [<source>]  
<source> ::= {CHANnel<n> | FUNCtion | MATH}  
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 316) 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 272
- "[":MEASure:TMIN"](#) on page 590
- "[":MEASure:XMAX"](#) on page 316
- "[":MEASure:XMIN"](#) on page 317

**:MEASure:TMIN**

(see page 658)

**Command Syntax**

```
:MEASure:TMIN [<source>]
<source> ::= {CHANnel<n> | FUNCtion | MATH}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 317](#)) 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 272
- "[":MEASure:TMAX"](#) on page 589
- "[":MEASure:XMAX"](#) on page 316
- "[":MEASure:XMIN"](#) on page 317

## :MEASure:TSTArt

 (see [page 658](#))

**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 660](#)). 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 259](#)) 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 257
- "[Introduction to :MEASure Commands](#)" on page 272
- "[:MARKer:X1Position](#)" on page 259
- "[:MARKer:X2Position](#)" on page 261
- "[:MARKer:XDELta](#)" on page 263
- "[:MEASure:TDELta](#)" on page 587
- "[:MEASure:TSTOP](#)" on page 592

## :MEASure:TSTOp

 (see page 658)

**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 660](#)). 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 261](#)) 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 257
- "[Introduction to :MEASure Commands](#)" on page 272
- "[":MARKer:X1Position](#)" on page 259
- "[":MARKer:X2Position](#)" on page 261
- "[":MARKer:XDELta](#)" on page 263
- "[":MEASure:TDELta](#)" on page 587
- "[":MEASure:TSTArt](#)" on page 591

## :MEASure:TVOLt

 (see page 658)

### Query Syntax

```
:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform. A rising slope is indicated by
            a plus sign (+). A falling edge is indicated by a minus
            sign (-).

<occurrence> ::= the transition to be reported. If the occurrence
                  number is one, the first crossing is reported. If
                  the number is two, the second crossing is reported,
                  etc.

<source> ::= {CHANnel<n> | FUNCTion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 304](#)) instead.

### Return Format

<value><NL>

## **7    Obsolete and Discontinued Commands**

<value> ::= time in seconds of the specified voltage crossing  
              in NR3 format

## :MEASure:UPPer

 (see page 658)

### 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 276](#)) 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 272
- "[":MEASure:LOWER"](#) on page 585
- "[":MEASure:THresholds"](#) on page 588

**:MEASure:VDELta** (see page 658)**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 266](#)) instead.

**Return Format**

```
<value><NL>
<value> ::= delta V value in NR1 format
```

**See Also**

- "[Introduction to :MARKer Commands](#)" on page 257
- "[Introduction to :MEASure Commands](#)" on page 272
- "[:MARKer:Y1Position](#)" on page 264
- "[:MARKer:Y2Position](#)" on page 265
- "[:MARKer:YDELta](#)" on page 266
- "[:MEASure:TDELta](#)" on page 587
- "[:MEASure:TSTArt](#)" on page 591

## :MEASure:VSTArt

 (see [page 658](#))

**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 660](#)). 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 264](#)) 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 257
- "[Introduction to :MEASure Commands](#)" on page 272
- "[":MARKer:Y1Position](#)" on page 264
- "[":MARKer:Y2Position](#)" on page 265
- "[":MARKer:YDELta](#)" on page 266
- "[":MARKer:X1Y1source](#)" on page 260
- "[":MEASure:SOURce](#)" on page 297
- "[":MEASure:TDELta](#)" on page 587
- "[":MEASure:TSTARt](#)" on page 591

**:MEASure:VSTOp**

 (see page 658)

**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 660](#)). 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 265](#)) 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 257
- "[Introduction to :MEASure Commands](#)" on page 272
- "[:MARKer:Y1Position](#)" on page 264
- "[:MARKer:Y2Position](#)" on page 265
- "[:MARKer:YDELta](#)" on page 266
- "[:MARKer:X2Y2source](#)" on page 262
- "[:MEASure:SOURce](#)" on page 297
- "[:MEASure:TDELta](#)" on page 587
- "[:MEASure:TSTARt](#)" on page 591

## :MTEST:AMASK:{SAVE | STORe}



(see page 658)

### 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 366) instead.

### See Also

- "Introduction to :MTEST Commands" on page 320

## :MTEST:AVERage

 (see page 658)

**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 177) 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 320
  - ":MTEST:AVERage:COUNt" on page 601

**:MTEST:AVERage:COUNt**

 (see page 658)

**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 167](#)) 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 320
- "[":MTEST:AVERage"](#) on page 600

## :MTEST:LOAD



(see [page 658](#))

### 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 354](#)) instead.

### See Also

- "[Introduction to :MTEST Commands](#)" on [page 320](#)
- "[:MTEST:AMASK:{SAVE | STORe}](#)" on [page 599](#)

## :MTEST:RUMode

 (see [page 658](#))

**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 337](#)) 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 320](#)
- "[:MTEST:RUMode:SOFailure](#)" on [page 604](#)

## :MTESt:RUMode:SOFailure

 (see [page 658](#))

**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:RUMODE:FACTion:STOP command (see [page 340](#)) 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 320
- "[":MTEST:RUMode"](#) on page 603

**:MTEST:{STARt | STOP}**

(see page 658)

**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 156) and :STOP command (see page 160) instead.

**See Also**

- "Introduction to :MTEST Commands" on page 320

## :MTEST:TRIGger:SOURce

 (see [page 658](#))

**Command Syntax** :MTEST:TRIGger:SOURce <source>

```
<source> ::= CHANnel<n>
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

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 411](#)) instead.

**Query Syntax** :MTEST:TRIGger:SOURce?

The :MTEST:TRIGger:SOURce? query returns the currently selected trigger source.

**Return Format** <source> ::= CHAN<n>

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

**See Also** • "Introduction to :MTEST Commands" on [page 320](#)

## :PRINt?

 (see page 658)

### 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 211](#)) 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 124
- "[Introduction to :HARDcopy Commands](#)" on page 246
- "[:HARDcopy:FORMat](#)" on page 581
- "[:HARDcopy:FACTors](#)" on page 249
- "[:HARDcopy:GRAYscale](#)" on page 582
- "[:DISPlay:DATA](#)" on page 211

**:TIMEbase:DElay**

 (see [page 658](#))

**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 405](#)).

**NOTE**

The :TIMEbase:DElay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POsition command (see [page 403](#)) 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"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 744

## :TRIGger:CAN:ACKNowledge

 (see page 658)

**Command Syntax**    :TRIGger:CAN:ACKnowledge <value>  
                      <value> ::= {0 | OFF}

This command was used with the N2758A CAN trigger module for 54620/54640 Series mixed-signal oscilloscopes. The InfiniiVision 5000 Series oscilloscopes do not support the N2758A CAN trigger module.

**Query Syntax**    :TRIGger:CAN:ACKnowledge?

The :TRIGger:CAN:ACKnowledge? query returns the current CAN acknowledge setting.

**Return Format**    <value><NL>  
                      <value> ::= 0

**See Also**    • "Introduction to :TRIGger Commands" on page 411  
                  • ":TRIGger:MODE" on page 417  
                  • ":TRIGger:CAN:TRIGger" on page 431

## :TRIGger:CAN:SIGNAl:DEFinition

 (see page 658)

### Command Syntax

```
:TRIGger:CAN:SIGNAl:DEFinition <value>
<value> ::= {CANH | CANL | RX | TX | DIFFerential}
```

The :TRIGger:CAN:SIGNAl:DEFinition command sets the CAN signal type when :TRIGger:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signal:

- CANH – the actual CAN\_H differential bus signal.

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.
- DIFFerential – the CAN differential bus signal connected to an analog source channel using a differential probe.

### NOTE

With InfiniiVision 5000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is DIFF.

### Query Syntax

```
:TRIGger:CAN:SIGNAl:DEFinition?
```

The :TRIGger:CAN:SIGNAl:DEFinition? query returns the current CAN signal type.

### Return Format

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

```
<value> ::= DIFF
```

### See Also

- "[Introduction to :TRIGger Commands](#)" on page 411
- "[:TRIGger:MODE](#)" on page 417
- "[:TRIGger:CAN:SIGNAl:BAUDrate](#)" on page 429
- "[:TRIGger:CAN:SOURce](#)" on page 430
- "[:TRIGger:CAN:TRIGger](#)" on page 431

## :TRIGger:LIN:SIGNAl:DEFinition

 (see page 658)

**Command Syntax**    :TRIGger:LIN:SIGNAl:DEFinition <value>  
<value> ::= {LIN | RX | TX}

The :TRIGger: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

With InfiniiVision 5000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is LIN.

**Query Syntax**    :TRIGger:LIN:SIGNAl:DEFinition?

The :TRIGger:LIN:SIGNAl:DEFinition? query returns the current LIN signal type.

**Return Format**    <value><NL>  
<value> ::= LIN

**See Also**    • "Introduction to :TRIGger Commands" on page 411  
• ":TRIGger:MODE" on page 417  
• ":TRIGger:LIN:SIGNAl:BAUDrate" on page 465  
• ":TRIGger:LIN:SOURce" on page 466

## :TRIGger:TV:TMode

 (see page 658)

### Command Syntax

```
:TRIGger:TV:TMode <mode>
<mode> ::= {FIELD1 | FIELD2 | AFIELDS | ALINES | LINE | VERTical
             | LFIELD1 | LFIELD2 | LALTernate | LVERTical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANDARD is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANDARD is GENeric (see [page 484](#)).

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 481](#)) instead.

### Query Syntax

```
:TRIGger:TV:TMODE?
```

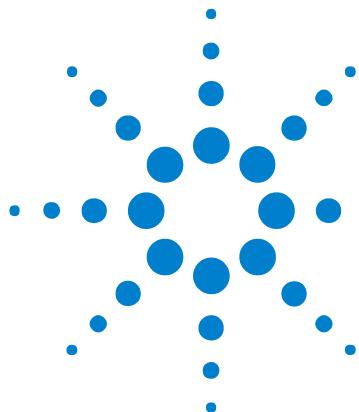
The :TRIGger:TV:TMODE? query returns the TV trigger mode.

### Return Format

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

```
<value> ::= {FIELD1 | FIELD2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
             | LALT | LVER}
```

## **7    Obsolete and Discontinued Commands**



8  
**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**

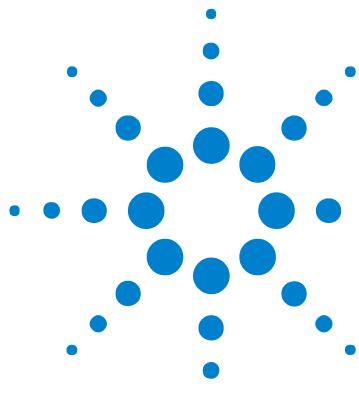
**+108, Compression Error**

**+109, No Data For Operation**

**+112, Unknown File Type**

**+113, Directory Not Supported**

## **8 Error Messages**



## 9

# Status Reporting

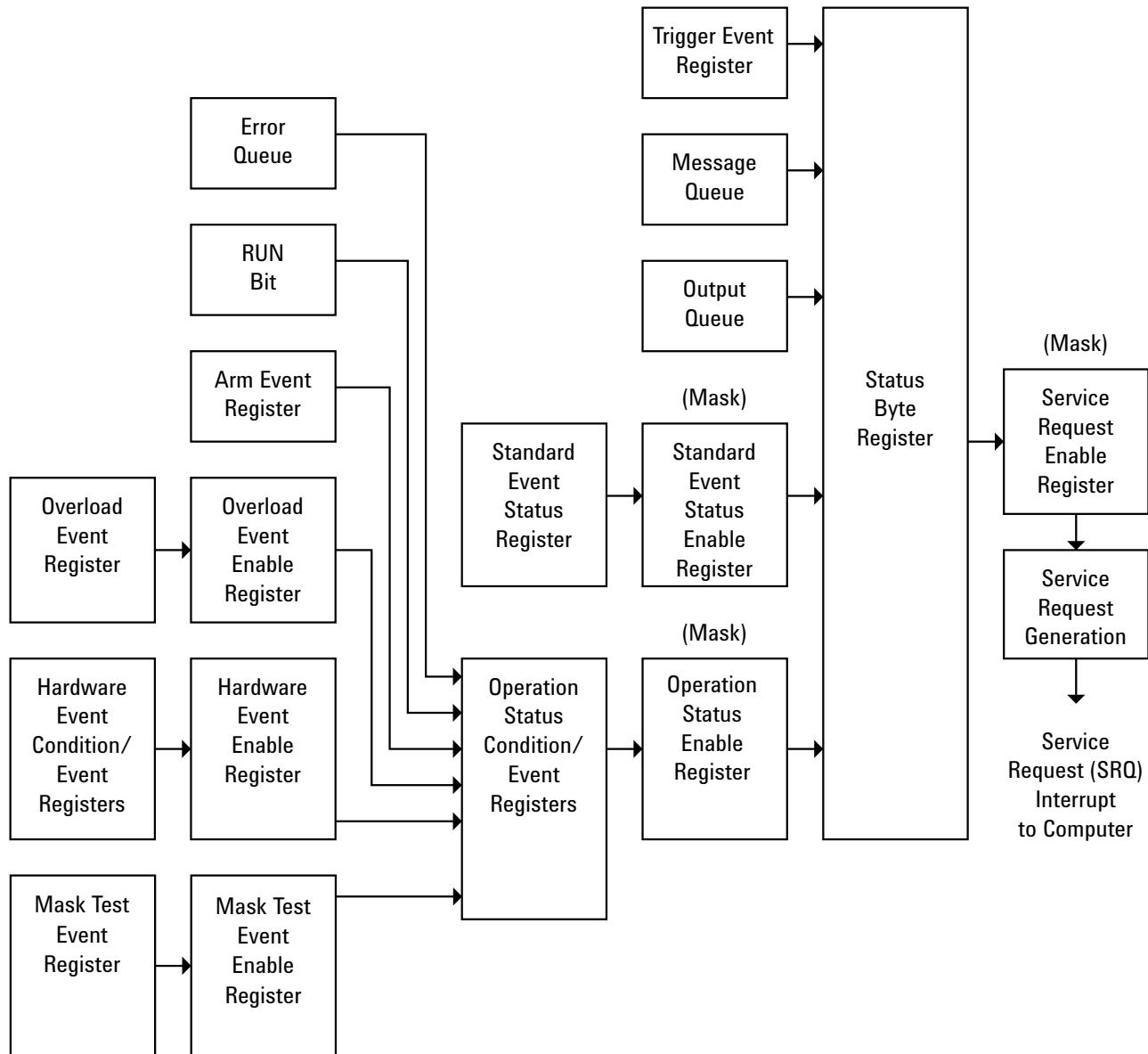
- Status Reporting Data Structures [626](#)
- Status Byte Register (STB) [629](#)
- Service Request Enable Register (SRE) [631](#)
- Trigger Event Register (TER) [632](#)
- Output Queue [633](#)
- Message Queue [634](#)
- (Standard) Event Status Register (ESR) [635](#)
- (Standard) Event Status Enable Register (ESE) [636](#)
- Error Queue [637](#)
- Operation Status Event Register (:OPERegister[:EVENT]) [638](#)
- Operation Status Condition Register (:OPERegister:CONDITION) [639](#)
- Arm Event Register (AER) [640](#)
- Overload Event Register (:OVLRegister) [641](#)
- Hardware Event Event Register (:HWERegister[:EVENT]) [642](#)
- Hardware Event Condition Register (:HWERegister:CONDITION) [643](#)
- Mask Test Event Event Register (:MTERegister[:EVENT]) [644](#)
- Clearing Registers and Queues [645](#)
- Status Reporting Decision Chart [646](#)

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.



## 9 Status Reporting



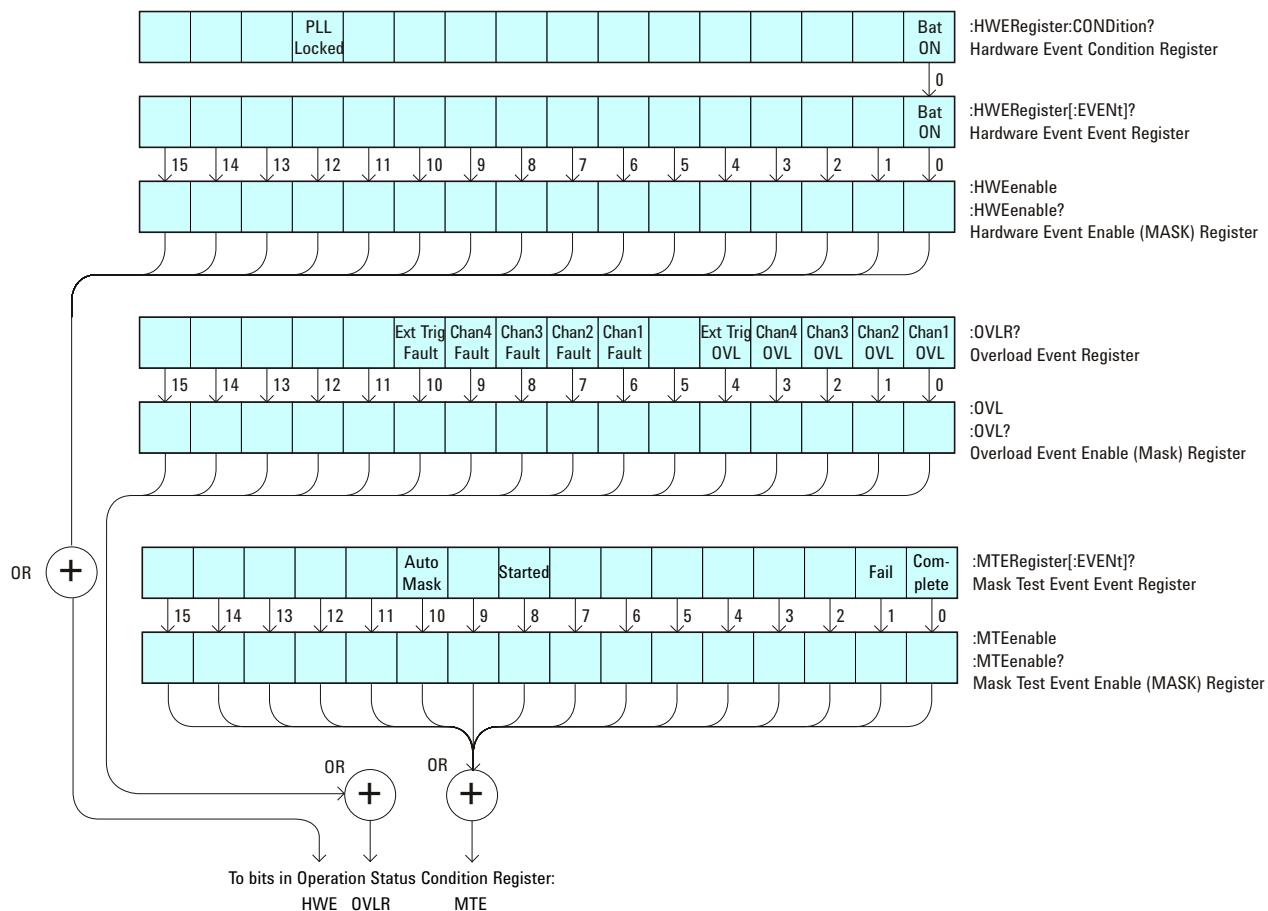
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

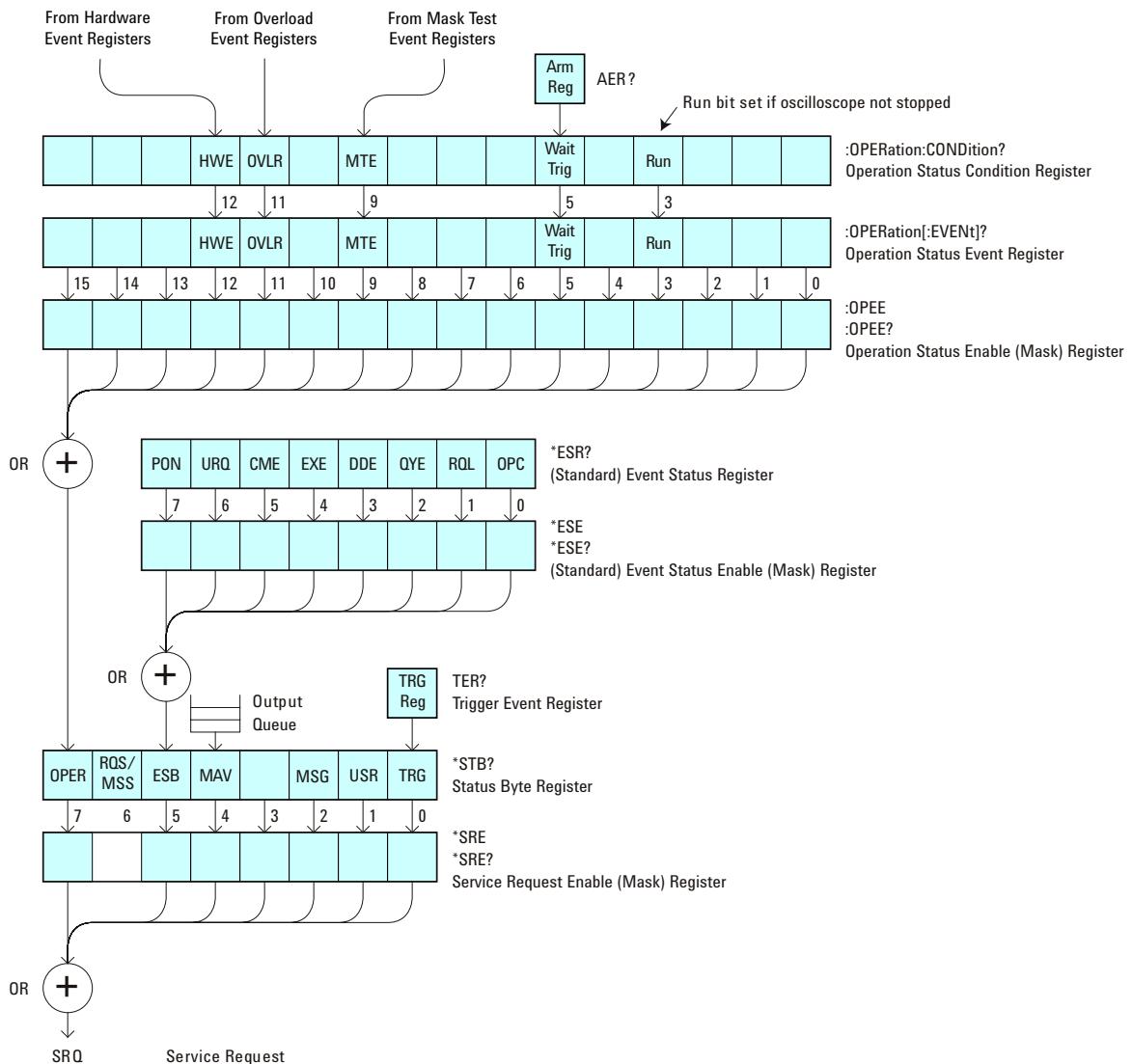
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

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:

- ["Status Byte Register \(STB\)" on page 117](#)
- ["Standard Event Status Register \(ESR\)" on page 104](#)
- ["Operation Status Condition Register" on page 147](#)
- ["Operation Status Event Register" on page 149](#)
- ["Overload Event Register \(OVLR\)" on page 153](#)
- ["Hardware Event Condition Register" on page 136](#)
- ["Hardware Event Event Register" on page 138](#)
- ["Mask Test Event Event Register" on page 143](#)

## 9 Status Reporting

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.

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.

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:

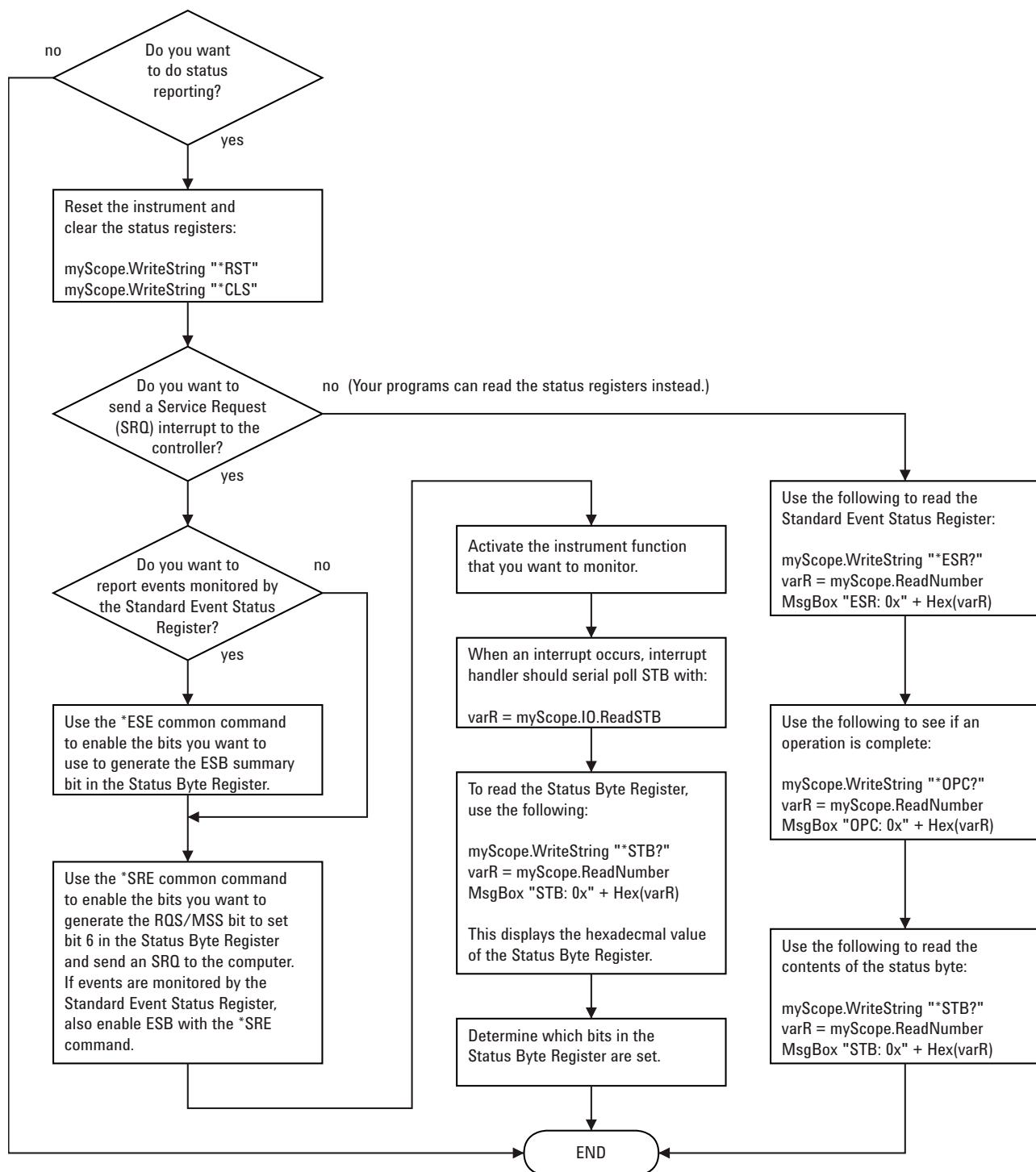
Name	Location	Description
Complete	bit 0	Is set when the mask test is complete.
Fail	bit 1	Is set when there is a mask test failure.
Started	bit 8	Is set when mask testing is started.
Auto Mask	bit 10	Is set when auto mask creation is completed.

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



## 10 Synchronizing Acquisitions

- [Synchronization in the Programming Flow 648](#)
- [Blocking Synchronization 649](#)
- [Polling Synchronization With Timeout 650](#)
- [Synchronizing with a Single-Shot Device Under Test \(DUT\) 652](#)
- [Synchronization with an Averaging Acquisition 654](#)

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 648](#)).
- 2 Acquire a waveform (see [page 648](#)).
- 3 Retrieve results (see [page 648](#)).

### 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.

	<b>Blocking Wait</b>	<b>Polling Wait</b>
<b>Use When</b>	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.
<b>Advantages</b>	No need for polling. Fastest method.	Remote interface will not timeout No need for device clear if no trigger.
<b>Disadvantages</b>	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.
<b>Implementation Details</b>	See " <a href="#">Blocking Synchronization</a> " on page 649.	See " <a href="#">Polling Synchronization With Timeout</a> " on page 650.

### 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 649 and "Polling Synchronization With Timeout" on page 650 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 650 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

```

## **10 Synchronizing Acquisitions**

## 11

# More About Oscilloscope Commands

Command Classifications 658

Valid Command/Query Strings 659

Query Return Values 677

All Oscilloscope Commands Are Sequential 678



## 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 658
- "[Non-Core Commands](#)" on page 658
- "[Obsolete Commands](#)" on page 658

### **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:

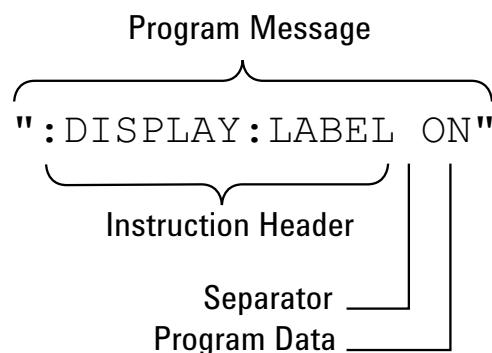
- "[Obsolete and Discontinued Commands](#)" on page 563
- As well as: "[Commands A-Z](#)" on page 535

## Valid Command/Query Strings

- "Program Message Syntax" on page 659
- "Command Tree" on page 663
- "Duplicate Mnemonics" on page 674
- "Tree Traversal Rules and Multiple Commands" on page 675

### 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 660), 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.

<b>Instruction Header</b>	The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. The "Command Tree" on page 663 illustrates how all the mnemonics can be joined together to form a complete header.  ":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.
<b>White Space (Separator)</b>	There are three types of headers: <ul style="list-style-type: none"><li>• "<a href="#">Simple Command Headers</a>" on page 661</li><li>• "<a href="#">Compound Command Headers</a>" on page 661</li><li>• "<a href="#">Common Command Headers</a>" on page 662</li></ul> 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).
<b>Program Data</b>	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. " <a href="#">Program Data Syntax Rules</a> " on page 662 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.
<b>Program Message Terminator</b>	The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the programming interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

### NOTE

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

### Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGE	RANG
PATTERn	PATT
TIMebase	TIM
DElay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

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

### Simple Command Headers

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

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

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGITIZE CHANnel1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

### Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLimit ON

### Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

\*<command header><terminator>

No space or separator is allowed between the asterisk (\*) and the command header. \*CLS is an example of a common command header.

### Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

<program mnemonic><separator><data><terminator>

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

<program mnemonic><separator><data>, <data><terminator>

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

#### Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEbase:MODE command can be set to normal, zoomed (delayed), XY, or ROLL. The character program data in this case may be MAIN, WINDow, XY, or ROLL. The command :TIMEbase:MODE WINDow sets the time base mode to zoomed.

The available mnemonics for character program data are always included with the command's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

#### Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEbase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

## Command Tree

The command tree shows all of the commands and the relationships of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed-ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the root of the command tree.

- : (root)**
  - :ACQuire (see [page 163](#))
  - :AALias (see [page 165](#))
  - :COMplete (see [page 166](#))
  - :COUNt (see [page 167](#))
  - :DAALias (see [page 168](#))
  - :MODE (see [page 169](#))
  - :POINts (see [page 170](#))
  - :SEGmented
    - :ANALyze (see [page 171](#))
    - :COUNt (see [page 172](#))
    - :INDEX (see [page 173](#))
    - :SRATe (see [page 176](#))
    - :TYPE (see [page 177](#))
  - :AER (Arm Event Register) (see [page 125](#))
  - :AUToscale (see [page 126](#))
    - :AMODE (see [page 128](#))
    - :CHANnels (see [page 129](#))
  - :BLANk (see [page 130](#))
  - :CALibrate (see [page 179](#))
    - :DATE (see [page 181](#))
    - :LABEL (see [page 182](#))

- :OUTPut (see [page 183](#))
- :STARt (see [page 184](#))
- :STATus (see [page 185](#))
- :SWITch (see [page 186](#))
- :TEMPerature (see [page 187](#))
- :TIME (see [page 188](#))
- :CDISplay (see [page 131](#))
- :CHANnel<n> (see [page 189](#))
  - :BWLimit (see [page 192](#))
  - :COUpling (see [page 193](#))
  - :DISPlay (see [page 194](#))
  - :IMPedance (see [page 195](#))
  - :INVert (see [page 196](#))
  - :LABEL (see [page 197](#))
  - :OFFSet (see [page 198](#))
  - :PROBe (see [page 199](#))
    - :ID (see [page 200](#))
    - :SKEW (see [page 201](#))
    - :STYPe (see [page 202](#))
  - :PROTection (see [page 203](#))
  - :RANGE (see [page 204](#))
  - :SCALe (see [page 205](#))
  - :UNITs (see [page 206](#))
  - :VERNier (see [page 207](#))
- :DIGItize (see [page 132](#))
- :DISPlay (see [page 208](#))
  - :CLEar (see [page 210](#))
  - :DATA (see [page 211](#))
  - :LABEL (see [page 213](#))
  - :LABList (see [page 214](#))
  - :PERSistence (see [page 215](#))
  - :SOURce (see [page 216](#))
  - :VECTors (see [page 217](#))
- :EXTernal (see [page 218](#))
- :BWLimit (see [page 220](#))

- :IMPedance (see [page 221](#))
- :PROBe (see [page 222](#))
  - :ID (see [page 223](#))
  - :STYPe (see [page 224](#))
- :PROTection (see [page 225](#))
- :RANGE (see [page 226](#))
- :UNITs (see [page 227](#))
- :FUNCtion (see [page 228](#))
  - :CENTer (see [page 231](#))
  - :DISPlay (see [page 232](#))
  - :GOFT
    - :OPERation (see [page 233](#))
    - :SOURce1 (see [page 234](#))
    - :SOURce2 (see [page 235](#))
  - :OFFSet (see [page 236](#))
  - :OPERation (see [page 237](#))
  - :RANGE (see [page 238](#))
  - :REFerence (see [page 239](#))
  - :SCALe (see [page 240](#))
  - :SOURce1 (see [page 241](#))
  - :SOURce2 (see [page 242](#))
  - :SPAN (see [page 243](#))
  - :WINDow (see [page 244](#))
- :HARDcopy (see [page 245](#))
  - :AREA (see [page 247](#))
  - :APRinter (see [page 248](#))
  - :FACTors (see [page 249](#))
  - :FFEed (see [page 250](#))
  - :INKSaver (see [page 251](#))
  - :LAYout (see [page 252](#))
  - :PAlette (see [page 253](#))
  - [:PRINter]
    - :LIST (see [page 254](#))
  - [:STARt] (see [page 255](#))
- :HWEenable (Hardware Event Enable Register) (see [page 134](#))

- :HWERegister
  - :CONDITION (Hardware Event Condition Register) (see [page 136](#))
  - [:EVENT] (Hardware Event Event Register) (see [page 138](#))
- :MARKer (see [page 256](#))
  - :MODE (see [page 258](#))
  - :X1Position (see [page 259](#))
  - :X1Y1source (see [page 260](#))
  - :X2Position (see [page 261](#))
  - :X2Y2source (see [page 262](#))
  - :XDELta (see [page 263](#))
  - :Y1Position (see [page 264](#))
  - :Y2Position (see [page 265](#))
  - :YDELta (see [page 266](#))
- :MEASure (see [page 267](#))
  - :CLEar (see [page 274](#))
  - :COUNter (see [page 275](#))
  - :DEFine (see [page 276](#))
  - :DELay (see [page 279](#))
  - :DUTYcycle (see [page 281](#))
  - :FALLtime (see [page 282](#))
  - :FREQuency (see [page 283](#))
  - :NWIDth (see [page 284](#))
  - :OVERshoot (see [page 285](#))
  - :PERiod (see [page 287](#))
  - :PHASE (see [page 288](#))
  - :PRESHoot (see [page 289](#))
  - :PWIDth (see [page 290](#))
  - :RISetime (see [page 294](#))
  - :RESULTS (see [page 291](#))
  - :SDEViation (see [page 295](#))
  - :SHOW (see [page 296](#))
  - :SOURce (see [page 297](#))
  - :STATistics (see [page 299](#))
    - :INCReement (see [page 300](#))
    - :RESET (see [page 301](#))

- :TEDGe (see [page 302](#))
- :TVALue (see [page 304](#))
- :VAMPlitude (see [page 306](#))
- :VAverage (see [page 307](#))
- :VBASe (see [page 308](#))
- :VMAX (see [page 309](#))
- :VMIN (see [page 310](#))
- :VPP (see [page 311](#))
- :VRATio (see [page 312](#))
- :VRMS (see [page 313](#))
- :VTIMe (see [page 314](#))
- :VTOP (see [page 315](#))
- :XMAX (see [page 316](#))
- :XMIN (see [page 317](#))
- :MERGe (see [page 140](#))
- :MTEenable (Mask Test Event Enable Register) (see [page 141](#))
- :MTERegister[:EVENT] (Mask Test Event Event Register) (see [page 143](#))
- :MTEST (see [page 318](#))
  - :AMASK
    - :CREate (see [page 323](#))
    - :SOURCe (see [page 324](#))
    - :UNITs (see [page 325](#))
    - :XDELta (see [page 326](#))
    - :YDELta (see [page 327](#))
  - :COUNt
    - :FWAVeforms (see [page 328](#))
    - :RESet (see [page 329](#))
    - :TIME (see [page 330](#))
    - :WAVeforms (see [page 331](#))
- :DATA (see [page 332](#))
- :DElete (see [page 333](#))
- :ENABLE (see [page 334](#))
- :LOCK (see [page 335](#))
- :OUTPut (see [page 336](#))
- :RMODE (see [page 337](#))

- :FACTion
  - :PRINt (see [page 338](#))
  - :SAVE (see [page 339](#))
  - :STOP (see [page 340](#))
  - :SIGMa (see [page 341](#))
  - :TIME (see [page 342](#))
  - :WAVeforms (see [page 343](#))
- :SCALe
  - :BIND (see [page 344](#))
  - :X1 (see [page 345](#))
  - :XDELta (see [page 346](#))
  - :Y1 (see [page 347](#))
  - :Y2 (see [page 348](#))
- :SOURce (see [page 349](#))
- :TITLE (see [page 350](#))
- :OPEE (Operation Status Enable Register) (see [page 145](#))
- :OPERegister
  - :CONDITION (Operation Status Condition Register) (see [page 147](#))
  - [:EVENT] (Operation Status Event Register) (see [page 149](#))
- :OVLenable (Overload Event Enable Register) (see [page 151](#))
- :OVLRegister (Overload Event Register) (see [page 153](#))
- :RECall
  - :FILEname (see [page 352](#))
  - :IMAGe (see [page 353](#))
    - [:STARt] (see [page 353](#))
  - :MASK (see [page 354](#))
    - [:STARt] (see [page 354](#))
  - :PWD (see [page 355](#))
  - :SETup (see [page 356](#))
    - [:STARt] (see [page 356](#))
- :RUN (see [page 156](#))
- :SAVE
  - :FILEname (see [page 359](#))
  - :IMAGe (see [page 360](#))
    - [:STARt] (see [page 360](#))

- :AREA (see [page 361](#))
- :FACTors (see [page 362](#))
- :FORMat (see [page 363](#))
- :IGColors (see [page 364](#))
- :PALETTE (see [page 365](#))
- :MASK (see [page 366](#))
  - [:STARt] (see [page 366](#))
- :PWD (see [page 367](#))
- :SETup (see [page 368](#))
  - [:STARt] (see [page 368](#))
- :WAveform (see [page 369](#))
  - [:STARt] (see [page 369](#))
  - :FORMAT (see [page 370](#))
  - :LENGth (see [page 371](#))
  - :SEGMENTed (see [page 372](#))
- :SBUS (see [page 373](#))
  - :CAN
    - :COUNt
    - :ERRor (see [page 375](#))
    - :OVERload (see [page 376](#))
    - :RESet (see [page 377](#))
    - :TOTal (see [page 378](#))
    - :UTILization (see [page 379](#))
  - :DISPlay (see [page 380](#))
  - :IIC
    - :WIDTh (see [page 384](#))
  - :LIN
    - :PARity (see [page 382](#))
  - :MODE (see [page 383](#))
  - :SPI
    - :ASIZE (see [page 381](#))
  - :UART
    - :BASE (see [page 385](#))
  - :COUNt
    - :ERRor (see [page 386](#))

- :RESet (see [page 387](#))
- :RXFRAMES (see [page 388](#))
- :TXFRAMES (see [page 389](#))
- :FRAMing (see [page 390](#))
- :SERial (see [page 157](#))
- :SINGle (see [page 158](#))
- :STATus (see [page 159](#))
- :STOP (see [page 160](#))
- :SYSTem (see [page 391](#))
  - :DATE (see [page 392](#))
  - :DSP (see [page 393](#))
  - :ERRor (see [page 394](#))
  - :LOCK (see [page 395](#))
  - :PROTection
    - :LOCK (see [page 381](#))
  - :SETup (see [page 397](#))
  - :TIME (see [page 399](#))
- :TER (Trigger Event Register) (see [page 161](#))
- :TIMEbase (see [page 400](#))
  - :MODE (see [page 402](#))
  - :POSITION (see [page 403](#))
  - :RANGE (see [page 404](#))
  - :REFERENCE (see [page 405](#))
  - :SCALE (see [page 406](#))
  - :VERNier (see [page 407](#))
  - :WINDOW
    - :POSITION (see [page 408](#))
    - :RANGE (see [page 409](#))
    - :SCALE (see [page 410](#))
- :TRIGger (see [page 411](#))
  - :HFReject (see [page 415](#))
  - :HOLDoff (see [page 416](#))
  - :MODE (see [page 417](#))
  - :NREject (see [page 418](#))
  - :PATTERn (see [page 419](#))

- :SWEep (see [page 421](#))
- :CAN (see [page 422](#))
  - :ACKNology (see [page 610](#))
- :PATTern
  - :DATA (see [page 424](#))
    - :LENGth (see [page 425](#))
  - :ID (see [page 426](#))
    - :MODE (see [page 427](#))
- :SAMPllepoint (see [page 428](#))
- :SIGNal
  - :BAUDrate (see [page 429](#))
  - :DEFinition (see [page 611](#))
- :SOURce (see [page 430](#))
- :TRIGger (see [page 431](#))
- :DURation (see [page 433](#))
  - :GREaterthan (see [page 434](#))
  - :LESSthan (see [page 435](#))
  - :PATTern (see [page 436](#))
  - :QUALifier (see [page 437](#))
  - :RANGE (see [page 438](#))
- [:EDGE] (see [page 439](#))
  - :COUpling (see [page 440](#))
  - :LEVel (see [page 441](#))
  - :REJect (see [page 442](#))
  - :SLOPe (see [page 443](#))
  - :SOURce (see [page 444](#))
- :GLITch (see [page 445](#))
  - :GREaterthan (see [page 446](#))
  - :LESSthan (see [page 447](#))
  - :LEVel (see [page 448](#))
  - :POLarity (see [page 449](#))
  - :QUALifier (see [page 450](#))
  - :RANGE (see [page 451](#))
  - :SOURce (see [page 452](#))
- :HFReject (see [page 415](#))

- :HOLDoff (see [page 416](#))
- :IIC (see [page 453](#))
- :PATTern
  - :ADDRess (see [page 454](#))
  - :DATA (see [page 455](#))
  - :DATA2 (see [page 456](#))
- :SOURce
  - :CLOCk (see [page 457](#))
  - :DATA (see [page 458](#))
- :TRIGger
  - :QUALifier (see [page 459](#))
  - [:TYPE] (see [page 460](#))
- :LIN (see [page 462](#))
  - :ID (see [page 463](#))
  - :SAMPLEpoint (see [page 464](#))
  - :SIGNal
    - :BAUDrate (see [page 465](#))
    - :DEFinition (see [page 612](#))
  - :SOURce (see [page 466](#))
  - :STANDard (see [page 467](#))
  - :SYNCbreak (see [page 468](#))
  - :TRIGger (see [page 469](#))
- :MODE (see [page 417](#))
- :NREject (see [page 418](#))
- :PATTern (see [page 419](#))
- :SPI (see [page 470](#))
  - :CLOCK
    - :SLOPe (see [page 471](#))
    - :TIMEout (see [page 472](#))
  - :FRAMing (see [page 473](#))
  - :PATTern
    - :DATA (see [page 474](#))
    - :WIDTh (see [page 475](#))
- :SOURce
  - :CLOCk (see [page 476](#))

- :DATA (see [page 477](#))
- :FRAMe (see [page 478](#))
- :SWEep (see [page 421](#))
- :TV (see [page 479](#))
  - :LINE (see [page 480](#))
  - :MODE (see [page 481](#))
  - :POLarity (see [page 482](#))
  - :SOURce (see [page 483](#))
  - :STANdard (see [page 484](#))
  - :TVMode (see [page 613](#))
- :UART (see [page 485](#))
  - :BASE (see [page 487](#))
  - :BAUDrate (see [page 488](#))
  - :BITorder (see [page 489](#))
  - :BURSt (see [page 490](#))
  - :DATA (see [page 491](#))
  - :IDLE (see [page 492](#))
  - :PARity (see [page 493](#))
  - :QUALifier (see [page 495](#))
  - :POLarity (see [page 494](#))
  - :SOURce
    - :RX (see [page 496](#))
    - :TX (see [page 497](#))
    - :TYPE (see [page 498](#))
    - :WIDTh (see [page 499](#))
- :VIEW (see [page 162](#))
- :WAVeform (see [page 500](#))
  - :BYTeorder (see [page 507](#))
  - :COUNt (see [page 508](#))
  - :DATA (see [page 509](#))
  - :FORMat (see [page 511](#))
  - :POINts (see [page 512](#))
    - :MODE (see [page 514](#))
  - :PREamble (see [page 516](#))
  - :SEGmented

- |   |   |
|---|---|
|   | <ul style="list-style-type: none"><li>• :COUNT (see <a href="#">page 519</a>)</li><li>• :TTAG (see <a href="#">page 520</a>)</li><li>• :SOURce (see <a href="#">page 521</a>)<ul style="list-style-type: none"><li>• :SUBSource (see <a href="#">page 525</a>)</li></ul></li><li>• :TYPE (see <a href="#">page 526</a>)</li><li>• :UNSIGNED (see <a href="#">page 527</a>)</li><li>• :VIEW (see <a href="#">page 528</a>)</li><li>• :XINCrement (see <a href="#">page 529</a>)</li><li>• :XORigin (see <a href="#">page 530</a>)</li><li>• :XREFerence (see <a href="#">page 531</a>)</li><li>• :YINCrement (see <a href="#">page 532</a>)</li><li>• :YORigin (see <a href="#">page 533</a>)</li><li>• :YREFerence (see <a href="#">page 534</a>)</li></ul> |
| <b>Common<br/>Commands (IEEE<br/>488.2)</b> | <ul style="list-style-type: none"><li>• *CLS (see <a href="#">page 101</a>)</li><li>• *ESE (see <a href="#">page 102</a>)</li><li>• *ESR (see <a href="#">page 104</a>)</li><li>• *IDN (see <a href="#">page 106</a>)</li><li>• *LRN (see <a href="#">page 107</a>)</li><li>• *OPC (see <a href="#">page 108</a>)</li><li>• *OPT (see <a href="#">page 109</a>)</li><li>• *RCL (see <a href="#">page 110</a>)</li><li>• *RST (see <a href="#">page 111</a>)</li><li>• *SAV (see <a href="#">page 114</a>)</li><li>• *SRE (see <a href="#">page 115</a>)</li><li>• *STB (see <a href="#">page 117</a>)</li><li>• *TRG (see <a href="#">page 119</a>)</li><li>• *TST (see <a href="#">page 120</a>)</li><li>• *WAI (see <a href="#">page 121</a>)</li></ul>   |

### 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 (see [page 663](#)). 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.

## 12 Programming Examples

- SICL Examples 680
- VISA Examples 698
- VISA COM Examples 744

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.



## SICL Examples

- "SICL Example in C" on page 680
- "SICL Example in Visual Basic" on page 689

### SICL Example in C

To compile and run this example in Microsoft Visual Studio 2005:

- 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 2005, 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 most of the commonly-used programming
 * features of your Agilent oscilloscope.
 * This program is to be built as a WIN32 console application.
```

```

* Edit the DEVICE_ADDRESS line to specify the address of the
* applicable device.
*/
#include <stdio.h>          /* For printf(). */
#include "sicl.h"           /* SICL routines. */

/* #define DEVICE_ADDRESS "gpib0,7" */          /* GPIB */
/* #define DEVICE_ADDRESS "lan[a-mso6102-90541]:inst0" */ /* LAN */
#define DEVICE_ADDRESS "usb0[2391::5970::30D3090541::0]" /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT      5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);           /* Initialize the oscilloscope. */
void extra(void);                /* Miscellaneous commands not executed,
                                 shown for reference purposes. */
void capture(void);              /* Digitize data from oscilloscope. */
void analyze(void);              /* Make some measurements. */
void get_waveform(void);         /* Download waveform data from
                                 oscilloscope. */
void save_waveform(void);        /* Save waveform data to a file. */
void retrieve_waveform(void);    /* Load waveform data from a file. */

/* Global variables */
INST id;                         /* Device session ID. */
char buf[256];                   /* Buffer for IDN string. */

/* Array for waveform data. */
unsigned char waveform_data[WAVE_DATA_SIZE];
double preamble[10];              /* Array for preamble. */

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 DEVICE_ADDRESS */
    id = iopen(DEVICE_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf ("Oscilloscope session initialized!\n");

        /* Set the I/O timeout value for this session to 5 seconds. */
        itimeout(id, TIMEOUT);
    }
}

```

## 12 Programming Examples

```
/* Clear the interface. */
iclear(id);
iremote(id);
}

initialize();

/* The extras function contains miscellaneous commands that do not
 * need to be executed for the proper operation of this example.
 * The commands in the extras function are shown for reference
 * purposes only.
*/
/* extra(); */ /* <-- Uncomment to execute the extra function */

capture();

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
* -----
* This function initializes both the interface and the oscilloscope
* to a known state.
*/
void initialize (void)
{
/* RESET - This command puts the oscilloscope in a known state.
* Without this command, the oscilloscope settings are unknown.
* This command is very important for program control.
*
* Many of the following initialization commands are initialized
* by this command. It is not necessary to reinitialize them
* unless you want to change the default setting.
*/
iprintf(id, "*RST\n");

/* Write the *IDN? string and send an EOI indicator, then read
* the response into buf.
ipromptf(id, "*IDN?\n", "%t", buf);
printf("%s\n", buf);
*/

/* AUTOSCALE - This command evaluates all the input signals and
* sets the correct conditions to display all of the active signals.
```

```

        */
        iprintf(id, ":AUTOSCALE\n");

        /* CHANNEL_PROBE - Sets the probe attenuation factor for the
         * selected channel. The probe attenuation factor may be from
         * 0.1 to 1000.
         */
        iprintf(id, ":CHAN1:PROBE 10\n");

        /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
         * The range value is eight times the volts per division.
         */
        iprintf(id, ":CHANNEL1:RANGE 8\n");

        /* TIME_RANGE - Sets the full scale horizontal time in seconds.
         * The range value is ten times the time per division.
         */
        iprintf(id, ":TIM:RANG 2e-3\n");

        /* TIME_REFERENCE - Possible values are LEFT and CENTER:
         * - LEFT sets the display reference one time division from the
         *   left.
         * - CENTER sets the display reference to the center of the screen.
         */
        iprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

        /* TRIGGER_SOURCE - Selects the channel that actually produces the
         * TV trigger. Any channel can be selected.
         */
        iprintf(id, ":TRIGGER:TV:SOURCE CHANNEL1\n");

        /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH, PATTern,
         * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
         */
        iprintf(id, ":TRIGGER:MODE EDGE\n");

        /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
         * to either POSITIVE or NEGATIVE.
         */
        iprintf(id, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
    }

    /*
     * extra
     * -----
     * The commands in this function are not executed and are shown for
     * reference purposes only. To execute these commands, call this
     * function from main.
     */
}

void extra (void)
{
    /* RUN_STOP (not executed in this example):
     * - RUN starts the acquisition of data for the active waveform
     *   display.
     * - STOP stops the data acquisition and turns off AUTOSTORE.
     */
}

```

## 12 Programming Examples

```
iprintf(id, ":RUN\n");
iprintf(id, ":STOP\n");

/* VIEW_BLANK (not executed in this example):
 * - VIEW turns on (starts displaying) an active channel or pixel
 *   memory.
 * - BLANK turns off (stops displaying) a specified channel or
 *   pixel memory.
 */
iprintf(id, ":BLANK CHANNEL1\n");
iprintf(id, ":VIEW CHANNEL1\n");

/* TIME_MODE (not executed in this example) - Set the time base
 * mode to MAIN, DELAYED, XY or ROLL.
 */
iprintf(id, ":TIMEBASE:MODE MAIN\n");
}

/*
* capture
* -----
* This function prepares the scope for data acquisition and then
* uses the DIGITIZE MACRO to capture some data.
*/
void capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    iprintf(id, ":ACQUIRE:TYPE NORMAL\n");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    iprintf(id, ":ACQUIRE:COMPLETE 100\n");

    /* DIGITIZE - Used to acquire the waveform data for transfer over
     * the interface. Sending this command causes an acquisition to
     * take place with the resulting data being placed in the buffer.
     */
    /* NOTE! The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement. Keep in mind when the oscilloscope is running,
     * communication with the computer interrupts data acquisition.
     * Setting up the oscilloscope over the bus causes the data
     * buffers to be cleared and internal hardware to be reconfigured.
     * If a measurement is immediately requested there may not have
     * been enough time for the data acquisition process to collect
     * data and the results may not be accurate. An error value of
     * 9.9E+37 may be returned over the bus in this situation.
     */
    iprintf(id, ":DIGITIZE CHAN1\n");
}
```

```

/*
 * analyze
 *
 * -----
 * In this example we will do the following:
 * - Save the system setup to a file for restoration at a later time.
 * - Save the oscilloscope display to a file which can be printed.
 * - Make single channel measurements.
 */

void analyze (void)
{
    double frequency, vpp;           /* Measurements. */
    double vdiv, off, sdiv, delay;   /* Calculated from preamble data. */
    int i;                          /* Loop counter. */

    /* Array for setup string. */
    unsigned char setup_string[SETUP_STR_SIZE];
    int setup_size;
    FILE *fp;
    unsigned char image_data[IMG_SIZE]; /* Array for image data. */
    int img_size;

    /* 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,
     * #800002204<setup string><NL>
     * where the setup string is 2204 bytes in length.
     */
    setup_size = SETUP_STR_SIZE;
    /* Query and read setup string. */
    ipromptf(id, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
    printf("Read setup string query (%d bytes).\n", setup_size);
    /* Write setup string to file. */
    fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
    setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
                       fp);
    fclose (fp);
    printf("Wrote setup string (%d bytes) to file.\n", setup_size);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
     * to the oscilloscope.
     */
    /* Read setup string from file. */
    fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
    setup_size = fread (setup_string, sizeof(unsigned char),
                       SETUP_STR_SIZE, fp);
    fclose (fp);
    printf("Read setup string (%d bytes) from file.\n", setup_size);
    /* Restore setup string. */
    iprintf(id, ":SYSTEM:SETUP #%08d", setup_size);
    ifwrite(id, setup_string, setup_size, 1, &setup_size);
    printf("Restored setup string (%d bytes).\n", setup_size);

    /* IMAGE_TRANSFER - In this example we will query for the image
     * data with ":DISPLAY:DATA?" to read the data and save the data
     * to the file "image.dat" which you can then send to a printer.
     */
}

```

## 12 Programming Examples

```
itimeout(id, 30000);
printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
img_size = IMG_SIZE;
ipromptf(id, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n",
         &img_size, image_data);
printf("Read display data query (%d bytes).\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
itimeout(id, 5000);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

/* Set source to measure. */
iprintf(id, ":MEASURE:SOURCE CHANNEL1\n");

/* Query for frequency. */
ipromptf(id, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
printf("The frequency is: %.4f kHz\n", frequency / 1000);

/* Query for peak to peak voltage. */
ipromptf(id, ":MEASURE:VPP?\n", "%lf", &vpp);
printf("The peak to peak voltage is: %.2f V\n", vpp);

/* WAVEFORM_DATA - Get waveform data from oscilloscope.
 */
get_waveform();

/* Make some calculations from the preamble data. */
vdiv = 32 * preamble [7];
off = preamble [8];
sdiv = preamble [2] * preamble [4] / 10;
delay = (preamble [2] / 2) * preamble [4] + preamble [5];

/* Print them out... */
printf ("Scope Settings for Channel 1:\n");
printf ("Volts per Division = %f\n", vdiv);
printf ("Offset = %f\n", off);
printf ("Seconds per Division = %f\n", sdiv);
printf ("Delay = %f\n", delay);

/* print out the waveform voltage at selected points */
for (i = 0; i < 1000; i = i + 50)
    printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
           ((float)waveform_data[i] - preamble[9]) * preamble[7] +
           preamble[8],
           ((float)i - preamble[6]) * preamble[4] + preamble[5]);

    save_waveform();          /* Save waveform data to disk. */
    retrieve_waveform();     /* Load waveform data from disk. */
}

/*

```

```

* get_waveform
* -----
* This function transfers the data displayed on the oscilloscope to
* the computer for storage, plotting, or further analysis.
*/
void get_waveform (void)
{
    int waveform_size;

    /* 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:PREAMBLE?"
     * query provides information concerning the vertical and horizontal
     * scaling of the waveform data.
     *
     * With the preamble information you can then use the
     * ":WAVEFORM:DATA?" query and read the data block in the
     * correct format.
    */

    /* WAVE_FORMAT - Sets the data transmission mode for waveform data
     * output. This command controls how the data is formatted when
     * sent from the oscilloscope and can be set to WORD or BYTE format.
    */

    /* Set waveform format to BYTE. */
    iprintf(id, ":WAVEFORM:FORMAT BYTE\n");

    /* WAVE_POINTS - Sets the number of points to be transferred.
     * The number of time points available is returned by the
     * "ACQUIRE:POINTS?" query. This can be set to any binary
     * fraction of the total time points available.
    */
    iprintf(id, ":WAVEFORM:POINTS 1000\n");

    /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
     * settings returned in the form <preamble block><NL> where the
     * <preamble block> is:
     *   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
     *   YINCREMENT : float32 - voltage difference between data points.
     *   YORIGIN    : float32 - value of the voltage at center screen.
     *   YREFERENCE : int32 - data point where y-origin occurs.
    */
    printf("Reading preamble\n");
    ipromptf(id, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
    /*
    printf("Preamble FORMAT: %e\n", preamble[0]);

```

## 12 Programming Examples

```
printf("Preamble TYPE: %e\n", preamble[1]);
printf("Preamble POINTS: %e\n", preamble[2]);
printf("Preamble COUNT: %e\n", preamble[3]);
printf("Preamble XINCREMENT: %e\n", preamble[4]);
printf("Preamble XORIGIN: %e\n", preamble[5]);
printf("Preamble XREFERENCE: %e\n", preamble[6]);
printf("Preamble YINCREMENT: %e\n", preamble[7]);
printf("Preamble YORIGIN: %e\n", preamble[8]);
printf("Preamble YREFERENCE: %e\n", preamble[9]);
*/

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEFORM:SOURCE" command.
 */
iprintf(id, ":WAVEFORM:DATA?\n"); /* Query waveform data. */

/* READ_WAVE_DATA - The wave data consists of two parts: the header,
 * and the actual waveform data followed by an New Line (NL)
 * character. The query data has the following format:
 *
 *      <header><waveform data block><NL>
 *
 * Where:
 *
 *      <header> = #800002048      (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 which can be set using the ":WAVEFORM:POINTS"
 * command. You may then read that number of bytes from the
 * oscilloscope; then, read the following NL character to
 * terminate the query.
 */
waveform_size = WAVE_DATA_SIZE;
/* Read waveform data. */
iscanf(id, "%#b\n", &waveform_size, waveform_data);
if ( waveform_size == WAVE_DATA_SIZE )
{
    printf("Waveform data buffer full: ");
    printf("May not have received all points.\n");
}
else
{
    printf("Reading waveform data... size = %d\n", waveform_size);
}
}

/*
 * save_waveform
 * -----
 * This function saves the waveform data from the get_waveform
 * function to disk. The data is saved to a file called "wave.dat".
 */
void save_waveform(void)
```

```

{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "wb");
    /* Write preamble. */
    fwrite(preamble, sizeof(preamble[0]), 10, fp);
    /* Write actually waveform data. */
    fwrite(waveform_data, sizeof(waveform_data[0]),
           (int)preamble[2], fp);
    fclose (fp);
}

/*
 * retrieve_waveform
 * -----
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "rb");
    /* Read preamble. */
    fread (preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread (waveform_data, sizeof(waveform_data[0]),
           (int)preamble[2], fp);
    fclose (fp);
}

```

## 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.

## 12 Programming Examples

```
'-----  
' 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  
  
'-----  
' Main Program  
'-----  
  
Sub Main()  
  
    On Error GoTo ErrorHandler  
  
    ' Open a device session using the SICL_ADDRESS.  
    id = iopen("lan[130.29.69.12]:inst0")  
    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"

        ' 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"

        ' 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.

    End Sub

```

## 12 Programming Examples

```
' Or, configure the settings with individual commands:  
' -----  
  
' Set trigger mode 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?")  
  
' Set vertical scale and offset.  
DoCommand ":CHANnel1:SCALe 0.5"  
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.  
' -----  
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)

        ' Acquire data.
        '
        DoCommand ":DIGITIZE"

        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:VAMPlitude"
    dblQueryResult = DoQueryNumber(":MEASURE:VAMPlitude?")
    MsgBox "Vertical amplitude:" + vbCrLf + _
        FormatNumber(dblQueryResult, 4) + " V"

    DoCommand ":MEASURE:FREQuency"
    dblQueryResult = DoQueryNumber(":MEASURE:FREQuency?")
    MsgBox "Frequency:" + vbCrLf + _
        FormatNumber(dblQueryResult / 1000, 4) + " kHz"

    ' Download the screen image.
    '
    ' Get screen image.
    Dim lngBlockSize As Long
    lngBlockSize = _
        DoQueryIEEEBlock_Bytes(":DISPLAY:DATA? PNG, SCReen, COLOR")
    Debug.Print "Image IEEEBlock bytes: " + CStr(lngBlockSize)

    ' Save screen image to a file:
    Dim strPath As String
    strPath = "c:\scope\data\screen.png"
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    For lngI = 10 To lngBlockSize - 1      ' Skip past 10-byte header.
        Put hFile, , byteArray(lngI)      ' Write data.
    Next lngI

```

## 12 Programming Examples

```
Close hFile      ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----
Dim lngPoints As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double
Dim dblYReference As Double

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Get the number of waveform points:
' How do you get max depth like when saving CSV from front panel?
dblQueryResult = DoQueryNumber(":WAVeform:POINTs?")
lngPoints = dblQueryResult
Debug.Print "Waveform points, channel 1: " + _
    CStr(lngPoints)

' Display the waveform settings:
dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment, channel 1: " + _
    Format(dblXIncrement, "Scientific")
dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin, channel 1: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment, channel 1: " + _
    Format(dblYIncrement, "Scientific")
dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin, channel 1: " + _
    Format(dblYOrigin, "Scientific")
dblYReference = DoQueryNumber(":WAVeform:YREFERence?")
Debug.Print "Waveform Y reference, channel 1: " + _
    Format(dblYReference, "Scientific")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMAT BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMAT?")
' Data in range 0 to 255.
Dim lngVSteps As Long
Dim intBytesPerData As Integer
lngVSteps = 256
intBytesPerData = 1

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Waveform data IEEEBlock bytes: " + 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 = 10 To lngNumBytes - 2      ' Skip past 10-byte header.
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        Format(dblXOrigin + lngI * dblXIncrement, "Scientific") + _
        ", " + _
        FormatNumber((lngDataValue - dblYReference) * dblYIncrement + _
        dblYOrigin)

Next lngI

' Close output file.
Close hFile      ' Close file.
MsgBox "Waveform format BYTE data written to " + _
    "c:\scope\data\waveform_data.csv."

Exit Sub

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Sub DoCommand(command As String)

    On Error GoTo ErrorHandler

    Call ivprintf(id, command + vbLf)
    CheckForInstrumentErrors command

    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.

```

## 12 Programming Examples

```
Call ivprintf(id, command + " ")
' Write definite-length block bytes.
Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)
' retCount is now actual number of bytes written.
CheckForInstrumentErrors command
DoCommandIEEEBlock = retCount

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 ret_val As Integer
Dim strResult As String * 200

Call ivprintf(id, query + vbLf)
Call ivscanf(id, "%200t", strResult)
CheckForInstrumentErrors query
DoQueryString = strResult

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)
CheckForInstrumentErrors query
DoQueryNumber = dblResult

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End
```

```

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

    On Error GoTo ErrorHandler

        ' Send query.
        Call ivprintf(id, query + vbLf)
        ' Read definite-length block bytes.
        Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)
        ' retCount is now actual number of bytes returned by read.
        CheckForInstrumentErrors query
        DoQueryIEEEBlock_Bytes = retCount

    Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Function

Private Sub CheckForInstrumentErrors(strCmdOrQuery As String)

    On Error GoTo ErrorHandler

    Dim strErrVal As String * 200
    Dim strOut As String

    Do
        Call ivprintf(id, "SYSTem:ERRor?" + vbLf) ' Request error message.
        Call ivscanf(id, "%200t", strErrVal) ' Read: Errno,"Error String".
        If Val(strErrVal) <> 0 Then
            strOut = strOut + "INST Error: " + RTrim(strErrVal) + vbLf
        End If
    Loop While Val(strErrVal) <> 0      ' End if find: 0,"No Error".

    If Not strOut = "" Then
        MsgBox strOut, vbExclamation, "INST Error Messages, " + _
            strCmdOrQuery
        Call iflush(id, I_BUF_DISCARD_READ Or I_BUF_DISCARD_WRITE)
    End If

    Exit Sub

ErrorHandler:
    MsgBox "*** Error: " + Error, vbExclamation

End Sub

```

## VISA Examples

- "VISA Example in C" on page 698
- "VISA Example in Visual Basic" on page 707
- "VISA Example in C#" on page 717
- "VISA Example in Visual Basic .NET" on page 731

### VISA Example in C

To compile and run this example in Microsoft Visual Studio 2005:

- 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 2005, 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\VISA\winnt\include).
  - d Show directories for **Library files**, and add the library files directory (for example, Program Files\VISA\winnt\lib\msc).
  - e Click **OK** to close the Options dialog.
- 9 Build and run the program.

```
/*
 * Agilent VISA Example in C
 * -----
 */
```

```

* This program illustrates most of the commonly-used programming
* features of your Agilent oscilloscope.
* This program is to be built as a WIN32 console application.
* Edit the RESOURCE line to specify the address of the
* applicable device.
*/

#include <stdio.h>           /* For printf(). */
#include <visa.h>             /* Agilent VISA routines. */

/*
/* GPIB */
/* #define RESOURCE "GPIBO::7::INSTR" */

/*
/* LAN */
/* #define RESOURCE "TCPIP0::a-mso6102-90541::inst0::INSTR" */

/*
/* USB */
#define RESOURCE "USB0::2391::5970::30D3090541::0::INSTR"

#define WAVE_DATA_SIZE 5000
#define TIMEOUT      5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);          /* Initialize the oscilloscope. */
void extra(void);               /* Miscellaneous commands not executed,
                                shown for reference purposes. */
void capture(void);             /* Digitize data from oscilloscope. */
void analyze(void);              /* Make some measurements. */
void get_waveform(void);         /* Download waveform data from
                                oscilloscope. */
void save_waveform(void);        /* Save waveform data to a file. */
void retrieve_waveform(void);    /* Load waveform data from a file. */

/* Global variables */
ViSession defaultRM, vi;         /* Device session ID. */
char buf[256];                  /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE]; /* Array for waveform
                                             data. */
double preamble[10];             /* Array for preamble. */

void main(void)
{
    /* Open session. */
    viOpenDefaultRM(&defaultRM);
    viOpen(defaultRM, RESOURCE, VI_NULL, VI_NULL, &vi);
    printf ("Oscilloscope session initialized!\n");

    /* Clear the interface. */
    viClear(vi);

    initialize();

    /* The extras function contains miscellaneous commands that do not
     * need to be executed for the proper operation of this example.

```

## 12 Programming Examples

```
* The commands in the extras function are shown for reference
* purposes only.
*/
/* extra(); */ /* <-- Uncomment to execute the extra function */

capture();

analyze();

/* Close session */
viClose(vi);
viClose(defaultRM);
printf ("Program execution is complete...\n");
}

/*
* initialize
* -----
* This function initializes both the interface and the oscilloscope
* to a known state.
*/
void initialize (void)
{
    /* RESET - This command puts the oscilloscope in a known state.
     * Without this command, the oscilloscope settings are unknown.
     * This command is very important for program control.
     *
     * Many of the following initialization commands are initialized
     * by this command. It is not necessary to reinitialize them
     * unless you want to change the default setting.
     */
    viPrintf(vi, "*RST\n");

    /* Write the *IDN? string and send an EOI indicator, then read
     * the response into buf.
    viQueryf(vi, "*IDN?\n", "%t", buf);
    printf("%s\n", buf);
    */

    /* AUTOSCALE - This command evaluates all the input signals and
     * sets the correct conditions to display all of the active signals.
     */
    viPrintf(vi, ":AUTOSCALE\n");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from
     * 0.1 to 1000.
     */
    viPrintf(vi, ":CHAN1:PROBE 10\n");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
     * The range value is eight times the volts per division.
     */
    viPrintf(vi, ":CHANNEL1:RANGE 8\n");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
```

```

        * The range value is ten times the time per division.
        */
viPrintf(vi, ":TIM:RANG 2e-3\n");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
 * - LEFT sets the display reference one time division from the
 *   left.
 * - CENTER sets the display reference to the center of the screen.
 */
viPrintf(vi, ":TIMEBASE:REFERENCE CENTER\n");

/* TRIGGER_SOURCE - Selects the channel that actually produces the
 * TV trigger. Any channel can be selected.
 */
viPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1\n");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
 * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
 */
viPrintf(vi, ":TRIGGER:MODE EDGE\n");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
 * to either POSITIVE or NEGATIVE.
 */
viPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * -----
 * The commands in this function are not executed and are shown for
 * reference purposes only. To execute these commands, call this
 * function from main.
 */

void extra (void)
{
    /* RUN_STOP (not executed in this example):
     * - RUN starts the acquisition of data for the active waveform
     *   display.
     * - STOP stops the data acquisition and turns off AUTOSTORE.
     */
    viPrintf(vi, ":RUN\n");
    viPrintf(vi, ":STOP\n");

    /* VIEW_BLANK (not executed in this example):
     * - VIEW turns on (starts displaying) an active channel or pixel
     *   memory.
     * - BLANK turns off (stops displaying) a specified channel or
     *   pixel memory.
     */
    viPrintf(vi, ":BLANK CHANNEL1\n");
    viPrintf(vi, ":VIEW CHANNEL1\n");

    /* TIME_MODE (not executed in this example) - Set the time base
     * mode to MAIN, DELAYED, XY or ROLL.
     */
}

```

## 12 Programming Examples

```
    viPrintf(vi, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * -----
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    viPrintf(vi, ":ACQUIRE:TYPE NORMAL\n");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    viPrintf(vi, ":ACQUIRE:COMPLETE 100\n");

    /* DIGITIZE - Used to acquire the waveform data for transfer over
     * the interface. Sending this command causes an acquisition to
     * take place with the resulting data being placed in the buffer.
     */
    /* NOTE! The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement. Keep in mind when the oscilloscope is running,
     * communication with the computer interrupts data acquisition.
     * Setting up the oscilloscope over the bus causes the data
     * buffers to be cleared and internal hardware to be reconfigured.
     * If a measurement is immediately requested there may not have
     * been enough time for the data acquisition process to collect
     * data and the results may not be accurate. An error value of
     * 9.9E+37 may be returned over the bus in this situation.
     */
    viPrintf(vi, ":DIGITIZE CHAN1\n");
}

/*
 * analyze
 * -----
 * In this example we will do the following:
 * - Save the system setup to a file for restoration at a later time.
 * - Save the oscilloscope display to a file which can be printed.
 * - Make single channel measurements.
 */

void analyze (void)
{
    double frequency, vpp;           /* Measurements. */
    double vdiv, off, sdiv, delay;   /* Values calculated from preamble
                                     * data. */
}
```

```

int i;                                /* Loop counter. */
unsigned char setup_string[SETUP_STR_SIZE];    /* Array for setup
                                                string. */
int setup_size;
FILE *fp;
unsigned char image_data[IMG_SIZE];      * Array for image data. */
int img_size;

/* 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,
 *     #800002204<setup string><NL>
 * where the setup string is 2204 bytes in length.
 */
setup_size = SETUP_STR_SIZE;
/* Query and read setup string. */
viQueryf(vi, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
printf("Read setup string query (%d bytes).\n", setup_size);
/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
                   fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to file.\n", setup_size);

/* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
 * to the oscilloscope.
 */
/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
setup_size = fread (setup_string, sizeof(unsigned char),
                    SETUP_STR_SIZE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file.\n", setup_size);
/* Restore setup string. */
viPrintf(vi, ":SYSTEM:SETUP #8%08d", setup_size);
viBufWrite(vi, setup_string, setup_size, &setup_size);
viPrintf(vi, "\n");
printf("Restored setup string (%d bytes).\n", setup_size);

/* IMAGE_TRANSFER - In this example we will query for the image
 * data with ":DISPLAY:DATA?" to read the data and save the data
 * to the file "image.dat" which you can then send to a printer.
 */
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 30000);
printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
img_size = IMG_SIZE;
viQueryf(vi, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n",
         &img_size, image_data);
printf("Read display data query (%d bytes).\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 5000);

```

## 12 Programming Examples

```
/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

/* Set source to measure. */
viPrintf(vi, ":MEASURE:SOURCE CHANNEL1\n");

/* Query for frequency. */
viQueryf(vi, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
printf("The frequency is: %.4f kHz\n", frequency / 1000);

/* Query for peak to peak voltage. */
viQueryf(vi, ":MEASURE:VPP?\n", "%lf", &vpp);
printf("The peak to peak voltage is: %.2f V\n", vpp);

/* WAVEFORM_DATA - Get waveform data from oscilloscope.
 */
get_waveform();

/* Make some calculations from the preamble data. */
vdiv = 32 * preamble[7];
off = preamble[8];
sdiv = preamble[2] * preamble[4] / 10;
delay = (preamble[2] / 2) * preamble[4] + preamble[5];

/* Print them out... */
printf ("Scope Settings for Channel 1:\n");
printf ("Volts per Division = %f\n", vdiv);
printf ("Offset = %f\n", off);
printf ("Seconds per Division = %f\n", sdiv);
printf ("Delay = %f\n", delay);

/* print out the waveform voltage at selected points */
for (i = 0; i < 1000; i = i + 50)
    printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
           ((float)waveform_data[i] - preamble[9]) * preamble[7] +
           preamble[8],
           ((float)i - preamble[6]) * preamble[4] + preamble[5]);

save_waveform();          /* Save waveform data to disk. */
retrieve_waveform();     /* Load waveform data from disk. */
}

/*
 * get_waveform
 * -----
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
    int waveform_size;

    /* 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:PREAMBLE?"
 * query provides information concerning the vertical and horizontal
 * scaling of the waveform data.
 *
 * With the preamble information you can then use the
 * ":WAVEFORM:DATA?" query and read the data block in the
 * correct format.
 */

/* WAVE_FORMAT - Sets the data transmission mode for waveform data
 * output. This command controls how the data is formatted when
 * sent from the oscilloscope and can be set to WORD or BYTE format.
 */

/* Set waveform format to BYTE. */
viPrintf(vi, ":WAVEFORM:FORMAT BYTE\n");

/* WAVE_POINTS - Sets the number of points to be transferred.
 * The number of time points available is returned by the
 * "ACQUIRE:POINTS?" query. This can be set to any binary
 * fraction of the total time points available.
 */
viPrintf(vi, ":WAVEFORM:POINTS 1000\n");

/* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
 * settings returned in the form <preamble block><NL> where the
 * <preamble block> is:
 *   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
 *   YINCREMENT : float32 - voltage difference between data points.
 *   YORIGIN    : float32 - value of the voltage at center screen.
 *   YREFERENCE : int32 - data point where y-origin occurs.
 */
printf("Reading preamble\n");
viQueryf(vi, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
/*
printf("Preamble FORMAT: %e\n", preamble[0]);
printf("Preamble TYPE: %e\n", preamble[1]);
printf("Preamble POINTS: %e\n", preamble[2]);
printf("Preamble COUNT: %e\n", preamble[3]);
printf("Preamble XINCREMENT: %e\n", preamble[4]);
printf("Preamble XORIGIN: %e\n", preamble[5]);
printf("Preamble XREFERENCE: %e\n", preamble[6]);
printf("Preamble YINCREMENT: %e\n", preamble[7]);
printf("Preamble YORIGIN: %e\n", preamble[8]);
printf("Preamble YREFERENCE: %e\n", preamble[9]);
*/
/*
 * QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 */

```

## 12 Programming Examples

```
* specified with the ":WAVEFORM:SOURCE" command.  
*/  
viPrintf(vi, ":WAVEFORM:DATA?\n"); /* Query waveform data. */  
  
/* READ_WAVE_DATA - The wave data consists of two parts: the header,  
* and the actual waveform data followed by an New Line (NL)  
* character. The query data has the following format:  
*  
*      <header><waveform data block><NL>  
*  
* Where:  
*  
*      <header> = #800002048      (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 which can be set using the ":WAVEFORM:POINTS"  
* command. You may then read that number of bytes from the  
* oscilloscope; then, read the following NL character to  
* terminate the query.  
*/  
waveform_size = WAVE_DATA_SIZE;  
/* Read waveform data. */  
viScanf(vi, "%#b\n", &waveform_size, waveform_data);  
if ( waveform_size == WAVE_DATA_SIZE )  
{  
    printf("Waveform data buffer full: ");  
    printf("May not have received all points.\n");  
}  
else  
{  
    printf("Reading waveform data... size = %d\n", waveform_size);  
}  
}  
  
/*  
* save_waveform  
* -----  
* This function saves the waveform data from the get_waveform  
* function to disk. The data is saved to a file called "wave.dat".  
*/  
  
void save_waveform(void)  
{  
    FILE *fp;  
  
    fp = fopen("c:\\scope\\data\\wave.dat", "wb");  
    /* Write preamble. */  
    fwrite(preamble, sizeof(preamble[0]), 10, fp);  
    /* Write actually waveform data. */  
    fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],  
          fp);  
    fclose(fp);  
}  
  
/*
```

```

* retrieve_waveform
* -----
* This function retrieves previously saved waveform data from a
* file called "wave.dat".
*/
void retrieve_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "rb");
    /* Read preamble. */
    fread(preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
          fp);
    fclose(fp);
}

```

## 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\VISA\winnt\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 VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```

' Agilent VISA Example in Visual Basic
' -----
' This program illustrates most of the 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.

```

## 12 Programming Examples

```
' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const DblArraySize = 20
Public Const ByteArraySize = 5000000
Public retCount As Long
Public dblArray(DblArraySize) As Double
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

'

' MAIN PROGRAM
' -----
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form. Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -----

Sub Main()

    ' Open the default resource manager session.
    err = viOpenDefaultRM(drm)

    ' Open the session to the resource.
    ' The "GPIB0" parameter is the VISA Interface name to
    ' an GPIB instrument as defined in:
    '     Start->Programs->Agilent IO Libraries->IO Config
    ' Change this name to whatever you have defined for your
    ' VISA Interface.
    ' "GPIB0::7::INSTR" is the address string for the device -
    ' this address will be the same as seen in:
    '     Start->Programs->Agilent IO Libraries->VISA Assistant
    ' (after the VISA Interface Name is defined in IO Config).

    ' err = viOpen(drm, "GPIB0::7::INSTR", 0, 0, vi)
    ' err = viOpen(drm, "TCPIP0::a-mso6102-90541::inst0::INSTR", 0, 0, vi)
    err = viOpen(drm, _
                  "USB0::2391::5970::30D3090541::0::INSTR", 0, 60000, vi)

    ' Initialize - Initialization will start the program with the
    ' oscilloscope in a known state.
    Initialize

    ' Capture - After initialization, you must make waveform data
    ' available to analyze. To do this, capture the data using the
```

```

' DIGITIZE command.
Capture

' Analyze - Once the waveform has been captured, it can be analyzed.
' There are many parts of a waveform to analyze. This example shows
' some of the possible ways to analyze various parts of a waveform.
Analyze

' Close the vi session and the resource manager session.
err = viClose(vi)
err = viClose(drm)

End Sub

'

' Initialize
' -----
' Initialize will start the program with the oscilloscope in a known
' state. This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
' - Oscilloscope
' - Channel 1 range
' - Display Grid
' - Timebase reference, range, and delay
' - Trigger mode and type
'
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -----
Private Sub Initialize()

' Clear the interface.
err = viClear(vi)

' 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.

' Reset the oscilloscope to the defaults.
err = viVPrintf(vi, "*RST" + vbLf, 0)

' IDN - Ask for the device's *IDN string.
err = viVPrintf(vi, "*IDN?" + vbLf, 0)
err = viVScanf(vi, "%t", strQueryResult) ' Read the results as a
' string.

' Display results.
MsgBox "Result is: " + strQueryResult, vbOKOnly, "*IDN? Result"

' AUTOSCALE - This command evaluates all the input signals and sets
' the correct conditions to display all of the active signals.
err = viVPrintf(vi, ":AUTOSCALE" + vbLf, 0)      ' Same as pressing
' the Autoscale key.

```

## 12 Programming Examples

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
' channel. The probe attenuation factor may be set from 0.1 to 1000.

' Set Probe to 10:1.
err = viVPrintf(vi, ":CHAN1:PROBE 10" + vbLf, 0)

' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
' range value is 8 times the volts per division.

' Set the vertical range to 8 volts.
err = viVPrintf(vi, ":CHANNEL1:RANGE 8" + vbLf, 0)

' TIME_RANGE - Sets the full scale horizontal time in seconds. The
' range value is 10 times the time per division.

' Set the time range to 0.002 seconds.
err = viVPrintf(vi, ":TIM:RANG 2e-3" + vbLf, 0)

' TIME_REFERENCE - Possible values are LEFT and CENTER.
' - LEFT sets the display reference on time division from the left.
' - CENTER sets the display reference to the center of the screen.

' Set reference to center.
err = viVPrintf(vi, ":TIMEBASE:REFERENCE CENTER" + vbLf, 0)

' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
' TV trigger. Any channel can be selected.
err = viVPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1" + vbLf, 0)

' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.

' Set the trigger mode to EDGE.
err = viVPrintf(vi, ":TRIGGER:MODE EDGE" + vbLf, 0)

' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
err = viVPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE" + vbLf, 0)

' The following commands are not executed and are shown for reference
' purposes only. To execute these commands, uncomment them.

' RUN_STOP - (not executed in this example)
' - RUN starts the acquisition of data for the active waveform
' display.
' - STOP stops the data acquisition and turns off AUTOSTORE.

' Start data acquisition.
err = viVPrintf(vi, ":RUN" + vbLf, 0)

' Stop the data acquisition.
err = viVPrintf(vi, ":STOP" + vbLf, 0)

' VIEW_BLANK - (not executed in this example)
' - VIEW turns on (starts displaying) a channel or pixel memory.
```

```

' - BLANK turns off (stops displaying) a channel or pixel memory.

' Turn channel 1 off.
err = viVPrintf(vi, ":BLANK CHANNEL1" + vbLf, 0)

' Turn channel 1 on.
err = viVPrintf(vi, ":VIEW CHANNEL1" + vbLf, 0)

' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
err = viVPrintf(vi, ":TIMEBASE:MODE MAIN" + vbLf, 0)

End Sub

'

' Capture
' -----
' We will capture the waveform using the digitize command.
' -----


Private Sub Capture()

' ACQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
' PEAK, or AVERAGE.
err = viVPrintf(vi, ":ACQUIRE:TYPE NORMAL" + vbLf, 0)

' ACQUIRE_COMPLETE - Specifies the minimum completion criteria for
' an acquisition. The parameter determines the percentage of time
' buckets needed to be "full" before an acquisition is considered
' to be complete.
err = viVPrintf(vi, ":ACQUIRE:COMPLETE 100" + vbLf, 0)

' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface. Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.

'

' NOTE! The DIGITIZE command is highly recommended for triggering
' modes other than SINGLE. This ensures that sufficient data is
' available for measurement. If DIGITIZE is used with single mode,
' the completion criteria may never be met. The number of points
' gathered in Single mode is related to the sweep speed, memory
' depth, and maximum sample rate. For example, take an oscilloscope
' with a 1000-point memory, a sweep speed of 10 us/div (100 us
' total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s. Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured. If a
' measurement is immediately requested, there may have not been

```

## 12 Programming Examples

```
' enough time for the data acquisition process to collect data,
' and the results may not be accurate. An error value of 9.9E+37
' may be returned over the bus in this situation.
'

err = viVPrintf(vi, ":DIGITIZE CHAN1" + vbLf, 0)

End Sub

'

' Analyze
' -----
' In analyze, we will do the following:
' - Save the system setup to a file and restore it.
' - Save the waveform data to a file on the computer.
' - Make single channel measurements.
' - Save the oscilloscope display to a file that can be sent to a
' printer.
' -----
'

Private Sub Analyze()

    ' Set up arrays for multiple parameter query returning an array
    ' with viVScanf/viVQueryf. Set retCount to the maximum number
    ' of elements that the array can hold.
    paramsArray(0) = VarPtr(retCount)
    paramsArray(1) = VarPtr(byteArray(0))

    ' 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,
    '     #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    Dim lngSetupStringSize As Long
    err = viVPrintf(vi, ":SYSTEM:SETUP?" + vbLf, 0)
    retCount = ByteArraySize

    ' Unsigned integer bytes.
    err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
    lngSetupStringSize = retCount

    ' Output setup string to a file:
    Dim strPath As String
    Dim lngI As Long
    strPath = "c:\scope\config\setup.dat"
    Close #1      ' If #1 is open, close it.

    ' Open file for output.
    Open strPath For Binary Access Write Lock Write As #1
    For lngI = 0 To lngSetupStringSize - 1
        Put #1, , byteArray(lngI)      ' Write data.
    Next lngI
    Close #1      ' Close file.

    ' 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.
    err = viVPrintf(vi, ":DISPLAY:DATA? BMP, SCREEN, COLOR" + vbLf, 0)
    retCount = ByteArraySize
```

```

' Unsigned integer bytes.
err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
' 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 file for output.
Open strPath For Binary Access Write Lock Write As #1
For lngI = 0 To retCount - 1
    Put #1, , byteArray(lngI)      ' Write data.
Next lngI
Close #1      ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1      ' Open file for input.
Get #1, , byteArray      ' Read data.
Close #1      ' Close file.
' Write learn string back to oscilloscope using ":SYSTEM:SETUP"
' command:
retCount = lngSetupStringSize
err = viVPrintf(vi, ":SYSTEM:SETUP %#b" + vbLf, paramsArray(0))

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure
err = viVPrintf(vi, ":MEASURE:SOURCE CHANNEL1" + vbLf, 0)

' Query for frequency.
err = viVPrintf(vi, ":MEASURE:FREQUENCY?" + vbLf, 0)
' Read frequency.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
err = viVPrintf(vi, ":MEASURE:DUTYCYCLE?" + vbLf, 0)
' Read duty cycle.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(dblQueryResult, 3) + "%"

' Query for risetime.
err = viVPrintf(vi, ":MEASURE:RISETIME?" + vbLf, 0)
' Read risetime.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Risetime:" + vbCrLf + _
      FormatNumber(dblQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
err = viVPrintf(vi, ":MEASURE:VPP?" + vbLf, 0)

```

## 12 Programming Examples

```
' Read VPP.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Peak to peak voltage:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"

' Query for Vmax.
err = viVPrintf(vi, ":MEASURE:VMAX?" + vbLf, 0)
' Read Vmax.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Maximum voltage:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"

' 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.
err = viVPrintf(vi, ":WAVEFORM:SOURCE CHAN1" + vbLf, 0)

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
err = viVPrintf(vi, ":WAVEFORM:POINTS 1000" + vbLf, 0)

' 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.
err = viVPrintf(vi, ":WAVEFORM:FORMAT WORD" + vbLf, 0)
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'err = viVPrintf(vi, ":WAVEFORM:FORMAT BYTE" + vbLf, 0)
'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, 2 = 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 difference between data points.
'   YORIGIN    : float32 - value is the voltage at center screen.
'   YREFERENCE : int32 - specifies the data point where y-origin
'                      occurs.
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

' Query for the preamble.
err = viVPrintf(vi, ":WAVEFORM:PREAMBLE?" + vbLf, 0)
paramsArray(1) = VarPtr(dblArray(0))
retCount = DblArraySize

' Read preamble information.
err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
sngYOrigin = dblArray(8)
lngYReference = dblArray(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
    FormatNumber(dblXIncrement * 1000000) + _
    " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
    FormatNumber(dblXOrigin * 1000000) + _
    " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
    CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
    FormatNumber(sngYIncrement * 1000) + _
    " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
    FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
    CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
    FormatNumber(lngVSteps * sngYIncrement / 8) + _
    " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
    FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
    FormatNumber(lngPoints * dblXIncrement / 10 * _
    1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _

```

## 12 Programming Examples

```
FormatNumber(((lngPoints / 2) * _
dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
err = viVPrintf(vi, ":WAV:DATA?" + vbLf, 0)

' 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

paramsArray(1) = VarPtr(byteArray(0))
retCount = ByteArraySize
' Unsigned integer bytes.
err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
' retCount is now actual number of bytes returned by query.
For lngI = 0 To retCount - 1 Step (retCount / 20)      ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = CLng(byteArray(lngI)) * 256 + _
        CLng(byteArray(lngI + 1))      ' 16-bit value.
    Else
        lngDataValue = CLng(byteArray(lngI))      ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + _
    sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) * _
    dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' 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.
```

```

err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN1" + vbLf, 0)

' Read time at edge 1 on ch 1.
err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge1))

' Query time at 1st rising edge on ch2.
err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN2" + vbLf, 0)

' Read time at edge 1 on ch 2.
err = viVScanf(vi, "%lf", VarPtr(dblChan2Edge1))

' 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.
err = viVPrintf(vi, ":MEASURE:TEDGE? +2, CHAN1" + vbLf, 0)

' Read time at edge 2 on ch 1.
err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge2))

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

End Sub

```

## VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

- 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\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 most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -----
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
    class VisaInstrumentApp
    {
        private static VisaInstrument oscp;

        public static void Main(string[] args)
        {
            try
            {
                oscp = new
                    VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR");

                Initialize();

                /* The extras function contains miscellaneous commands that
                 * do not need to be executed for the proper operation of
                 * this example. The commands in the extras function are
                 * shown for reference purposes only.
                */
                // Extra(); // Uncomment to execute the extra function.
            }
        }
}
```

```

        Capture();
        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
    {
        oscp.Close();
    }
}

/*
 * Initialize()
 * -----
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
    StringBuilder strResults;

    /* 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.
    */
    oscp.DoCommand("*RST");    // Reset the to the defaults.
    oscp.DoCommand("*CLS");    // Clear the status data structures.

    /* IDN - Ask for the device's *IDN string.
     */
    strResults = oscp.DoQueryString("*IDN?");

    // Display results.
    Console.Write("Result is: {0}", strResults);

    /* AUTOSCALE - This command evaluates all the input signals
     * and sets the correct conditions to display all of the
     * active signals.
    */
    oscp.DoCommand(":AUToscale");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from

```

## 12 Programming Examples

```
    * 0.1 to 1000.  
    */  
    oscp.DoCommand(":CHANnel1:PROBe 10");  
  
    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.  
     * The range value is eight times the volts per division.  
     */  
    oscp.DoCommand(":CHANnel1:RANGE 8");  
  
    /* TIME_RANGE - Sets the full scale horizontal time in seconds.  
     * The range value is ten times the time per division.  
     */  
    oscp.DoCommand(":TIMEbase:RANGE 2e-3");  
  
    /* TIME_REFERENCE - Possible values are LEFT and CENTER:  
     * - LEFT sets the display reference one time division from  
     *   the left.  
     * - CENTER sets the display reference to the center of the  
     *   screen.  
     */  
    oscp.DoCommand(":TIMEbase:REFERENCE CENTER");  
  
    /* TRIGGER_SOURCE - Selects the channel that actually produces  
     * the TV trigger. Any channel can be selected.  
     */  
    oscp.DoCommand(":TRIGger:TV:SOURCe CHANnel1");  
  
    /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH,  
     * PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,  
     * UART, or USB.  
     */  
    oscp.DoCommand(":TRIGger:MODE EDGE");  
  
    /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the  
     * trigger to either POSITIVE or NEGATIVE.  
     */  
    oscp.DoCommand(":TRIGger:EDGE:SLOPe POSitive");  
}  
  
/*  
 * Extra()  
 * -----  
 * The commands in this function are not executed and are shown  
 * for reference purposes only. To execute these commands, call  
 * this function from main.  
 */  
private static void Extra()  
{  
    /* RUN_STOP (not executed in this example):  
     * - RUN starts the acquisition of data for the active  
     *   waveform display.  
     * - STOP stops the data acquisition and turns off AUTOSTORE.  
     */  
    oscp.DoCommand(":RUN");  
    oscp.DoCommand(":STOP");  
  
    /* VIEW_BLANK (not executed in this example):  
     */  
}
```

```

* - VIEW turns on (starts displaying) an active channel or
*   pixel memory.
* - BLANK turns off (stops displaying) a specified channel or
*   pixel memory.
*/
oscp.DoCommand(":BLANK CHANnel1");
oscp.DoCommand(":VIEW CHANnel1");

/* TIME_MODE (not executed in this example) - Set the time base
 * mode to MAIN, DELAYED, XY or ROLL.
 */
oscp.DoCommand(":TIMEbase:MODE MAIN");
}

/*
* Capture()
* -----
* This function prepares the scope for data acquisition and then
* uses the DIGITIZE MACRO to capture some data.
*/
private static void Capture()
{
/* ACQUIRE_TYPE - Sets the acquisition mode. There are three
 * acquisition types NORMAL, PEAK, or AVERAGE.
 */
oscp.DoCommand(":ACQuire:TYPE NORMAL");

/* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
 * for an acquisition. The parameter determines the percentage
 * of time buckets needed to be "full" before an acquisition is
 * considered to be complete.
 */
oscp.DoCommand(":ACQuire:COMPLETE 100");

/* DIGITIZE - Used to acquire the waveform data for transfer
 * over the interface. Sending this command causes an
 * acquisition to take place with the resulting data being
 * placed in the buffer.
*/
/* NOTE! The use of the DIGITIZE command is highly recommended
 * as it will ensure that sufficient data is available for
 * measurement. Keep in mind when the oscilloscope is running,
 * communication with the computer interrupts data acquisition.
 * Setting up the oscilloscope over the bus causes the data
 * buffers to be cleared and internal hardware to be
 * reconfigured.
 * If a measurement is immediately requested there may not have
 * been enough time for the data acquisition process to collect
 * data and the results may not be accurate. An error value of
 * 9.9E+37 may be returned over the bus in this situation.
 */
oscp.DoCommand(":DIGITIZE CHANnel1");
}

/*
* Analyze()

```

## 12 Programming Examples

```
* -----
* In this example we will do the following:
* - Save the system setup to a file for restoration at a later
*   time.
* - Save the oscilloscope display to a file which can be
*   printed.
* - Make single channel measurements.
*/
private static void Analyze()
{
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.

    /* 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,
     * #800002204<setup string><NL>
     * where the setup string is 2204 bytes in length.
    */
    Console.WriteLine("Saving oscilloscope setup to " +
        "c:\\scope\\config\\setup.dat");
    if (File.Exists("c:\\scope\\config\\setup.dat"))
        File.Delete("c:\\scope\\config\\setup.dat");

    // Query and read setup string.
    nLength = oscp.DoQueryIEEEBlock(":SYSTem:SETup?",
        out ResultsArray);
    Console.WriteLine("Read oscilloscope setup ({0} bytes).",
        nLength);

    // Write setup string to file.
    File.WriteAllBytes("c:\\scope\\config\\setup.dat",
        ResultsArray);
    Console.WriteLine("Wrote setup string ({0} bytes) to file.",
        nLength);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
     * string to the oscilloscope.
    */
    byte[] dataArray;
    int nBytesWritten;

    // Read setup string from file.
    dataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
    Console.WriteLine("Read setup string ({0} bytes) from file.",
        dataArray.Length);

    // Restore setup string.
    nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup",
        dataArray);
    Console.WriteLine("Restored setup string ({0} bytes).",
        nBytesWritten);

    /* IMAGE_TRANSFER - In this example, we query for the screen
     * data with the ":DISPLAY:DATA?" query. The .png format
     * data is saved to a file in the local file system.
    */
}
```

```

        */
Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
oscp.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
nLength = oscp.DoQueryIEEEBlock(
    ":DISPlay:DATA? PNG, SCreen, COLOR", out ResultsArray);
Console.WriteLine("Read screen image ({0} bytes).", nLength);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
    ResultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.", 
    nLength);

// Return I/O timeout to five seconds.
oscp.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */
 
// Set source to measure.
oscp.DoCommand(":MEASure:SOURce CHANnel1");

// Query for frequency.
double fResults;
fResults = oscp.DoQueryValue(":MEASure:FREQuency?");
Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

// Query for peak to peak voltage.
fResults = oscp.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. 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:PREAMBLE?" query provides information concerning
 * the vertical and horizontal scaling of the waveform data.
 *
 * With the preamble information you can then use the
 * ":WAVEFORM:DATA?" query and read the data block in the
 * correct format.
 */
 
/* WAVE_FORMAT - Sets the data transmission mode for waveform
 * data output. This command controls how the data is

```

## 12 Programming Examples

```
* formatted when sent from the oscilloscope and can be set
* to WORD or BYTE format.
*/
// Set waveform format to BYTE.
oscp.DoCommand(":WAVEform:FORMAT BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
* The number of time points available is returned by the
* "ACQUIRE:POINTS?" query. This can be set to any binary
* fraction of the total time points available.
*/
oscp.DoCommand(":WAVEform:POINTS 1000");

/* GET_PREAMBLE - The preamble contains all of the current
* WAVEFORM settings returned in the form <preamble block><NL>
* where the <preamble block> is:
*      FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
*      YINCREMENT : float32 - voltage difference between data
*                      points.
*      YORIGIN    : float32 - value of the voltage at center
*                      screen.
*      YREFERENCE : int32 - data point where y-origin occurs.
*/
Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = oscp.DoQueryValues(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMAT: {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: {0:e}", fType);

double fPoints = fResultsArray[2];
Console.WriteLine("Preamble POINTs: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: {0:e}", fCount);

double fxincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fxincrement);

double fxorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fxorigin);

double fxreference = fResultsArray[6];
```

```

Console.WriteLine("Preamble XREFERENCE: {0:e}", fxreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCREMENT: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORIGIN: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFERENCE: {0:e}", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEform:SOURce" command.
 */

/* 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 block><NL>
 *
 * Where:
 *
 *      <header> = #800002048      (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 which can be set using the
 * ":WAVEform:POINTS" command. You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVEform:DATA?",
    out ResultsArray);
Console.WriteLine("Read waveform data ({0} bytes).", nLength);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv = fPoints * fxincrement / 10;
double fDelay = (fPoints / 2) * fxincrement + fxorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < 1000; i = i + 50)
    Console.WriteLine("Data point {0:d} = {1:f2} Volts at "

```

## 12 Programming Examples

```
+ "{2:f10} Seconds", i,
((float)ResultsArray[i] - fYreference) * fYincrement +
fYorigin,
((float)i - fxreference) * fxincrement + fxorigin);

/* SAVE_WAVE_DATA - saves the waveform data to a CSV format
 * file named "waveform.csv".
 */
if (File.Exists("c:\\scope\\data\\waveform.csv"))
    File.Delete("c:\\scope\\data\\waveform.csv");

StreamWriter writer =
    File.CreateText("c:\\scope\\data\\waveform.csv");
for (int i = 0; i < 1000; i++)
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fxreference) * fxincrement + fxorigin,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin);
writer.Close();
}
}

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 instrument errors (another command and result).
        CheckForInstrumentErrors(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{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,
    out nBytesWritten);
CheckVisaStatus(nViStatus);

// Write command termination character.
nViStatus = visa32.viPrintf(m_nSession, "\n");
CheckVisaStatus(nViStatus);

// Check for instrument errors (another command and result).
CheckForInstrumentErrors(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 instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}

public double DoQueryValue(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultValue();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

```

## 12 Programming Examples

```
}

public double[] DoQueryValues(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double[] fResultsArray;
    fResultsArray = VisaGetResultValues();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(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 instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return length;
}

private void CheckForInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    StringBuilder strInstrumentError = new StringBuilder(1000);
    bool bFirstError = true;
    do
    {
        VisaSendCommandOrQuery(":SYSTem:ERRor?");
        strInstrumentError = VisaGetResultString();

        if (strInstrumentError.ToString() != "+0,\"No error\"\n")
        {
            if (bFirstError)
            {
                Console.WriteLine("ERROR(s) for command '{0}': {1}",
                    strCommand);
                bFirstError = false;
            }
            Console.Write(strInstrumentError);
        }
    } while (strInstrumentError.ToString() != "+0,\"No error\"\n");
}
```

```

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 VisaGetResultValue()
{
    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[] VisaGetResultValues()
{
    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).
}

```

## 12 Programming Examples

```
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 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 2005:

- 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\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

the VISA Help that comes with Agilent IO Libraries Suite 15.

*[REDACTED]*

' This program illustrates most of the commonly-used programming  
' features of your Agilent oscilloscope.

```
Imports System  
Imports System.IO
```

## 12 Programming Examples

```
Imports System.Text

Namespace InfiniiVision
    Class VisaInstrumentApp
        Private Shared oscp As VisaInstrument

        Public Shared Sub Main(ByVal args As String())
            Try
                oscp = _
                    New VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR")

                Initialize()

                ' The extras function contains miscellaneous commands that
                ' do not need to be executed for the proper operation of
                ' this example. The commands in the extras function are
                ' shown for reference purposes only.

                ' Extra()      ' Uncomment to execute the extra function.
                Capture()
                Analyze()
            Catch err As System.ApplicationException
                MsgBox("*** Error : " & err.Message, vbExclamation, _
                    "VISA Error Message")
            Exit Sub
            Catch err As System.SystemException
                MsgBox("*** Error : " & err.Message, vbExclamation, _
                    "System Error Message")
            Exit Sub
            Catch err As System.Exception
                Debug.Fail("Unexpected Error")
                MsgBox("*** Error : " & err.Message, vbExclamation, _
                    "Unexpected Error")
            Exit Sub
        Finally
            oscp.Close()
        End Try
    End Sub

    ' Initialize()
    ' -----
    ' This function initializes both the interface and the
    ' oscilloscope to a known state.

    Private Shared Sub Initialize()
        Dim strResults As StringBuilder

        ' 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.

        ' Reset the to the defaults.
        oscp.DoCommand("*RST")
        ' Clear the status data structures.
```

```

oscop.DoCommand("*CLS")

' IDN - Ask for the device's *IDN string.
strResults = oscop.DoQueryString("*IDN?")
' Display results.
Console.WriteLine("Result is: {0}", strResults)

' AUTOSCALE - This command evaluates all the input signals
' and sets the correct conditions to display all of the
' active signals.
oscop.DoCommand(":AUToscale")

' CHANNEL_PROBE - Sets the probe attenuation factor for the
' selected channel. The probe attenuation factor may be from
' 0.1 to 1000.
oscop.DoCommand(":CHANnel1:PROBe 10")

' CHANNEL_RANGE - Sets the full scale vertical range in volts.
' The range value is eight times the volts per division.
oscop.DoCommand(":CHANnel1:RANGE 8")

' TIME_RANGE - Sets the full scale horizontal time in seconds.
' The range value is ten times the time per division.
oscop.DoCommand(":TIMEbase:RANGE 2e-3")

' TIME_REFERENCE - Possible values are LEFT and CENTER:
' - LEFT sets the display reference one time division from
'   the left.
' - CENTER sets the display reference to the center of the
'   screen.
oscop.DoCommand(":TIMEbase:REFERENCE CENTER")

' TRIGGER_SOURCE - Selects the channel that actually produces
' the TV trigger. Any channel can be selected.
oscop.DoCommand(":TRIGger:TV:SOURCe CHANnel1")

' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH,
' PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
' UART, or USB.
oscop.DoCommand(":TRIGger:MODE EDGE")

' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
' trigger to either POSITIVE or NEGATIVE.
oscop.DoCommand(":TRIGger:EDGE:SLOPe POSitive")

End Sub

' Extra()
' -----
' The commands in this function are not executed and are shown
' for reference purposes only. To execute these commands, call
' this function from main.

Private Shared Sub Extra()

    ' RUN_STOP (not executed in this example):

```

## 12 Programming Examples

```
' - RUN starts the acquisition of data for the active
'   waveform display.
' - STOP stops the data acquisition and turns off AUTOSTORE.
oscp.DoCommand(":RUN")
oscp.DoCommand(":STOP")

' VIEW_BLANK (not executed in this example):
' - VIEW turns on (starts displaying) an active channel or
'   pixel memory.
' - BLANK turns off (stops displaying) a specified channel or
'   pixel memory.
oscp.DoCommand(":BLANK CHANnel1")
oscp.DoCommand(":VIEW CHANnel1")

' TIME_MODE (not executed in this example) - Set the time base
' mode to MAIN, DELAYED, XY or ROLL.
oscp.DoCommand(":TIMEbase:MODE MAIN")

End Sub

' Capture()
' -----
' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.

Private Shared Sub Capture()

    ' ACQUIRE_TYPE - Sets the acquisition mode. There are three
    ' acquisition types NORMAL, PEAK, or AVERAGE.
    oscp.DoCommand(":ACQuire:TYPE NORMAL")

    ' ACQUIRE_COMPLETE - Specifies the minimum completion criteria
    ' for an acquisition. The parameter determines the percentage
    ' of time buckets needed to be "full" before an acquisition is
    ' considered to be complete.
    oscp.DoCommand(":ACQuire:COMPLETE 100")

    ' DIGITIZE - Used to acquire the waveform data for transfer
    ' over the interface. Sending this command causes an
    ' acquisition to take place with the resulting data being
    ' placed in the buffer.

    ' NOTE! The use of the DIGITIZE command is highly recommended
    ' as it will ensure that sufficient data is available for
    ' measurement. Keep in mind when the oscilloscope is running,
    ' communication with the computer interrupts data acquisition.
    ' Setting up the oscilloscope over the bus causes the data
    ' buffers to be cleared and internal hardware to be
    ' reconfigured.
    ' If a measurement is immediately requested there may not have
    ' been enough time for the data acquisition process to collect
    ' data and the results may not be accurate. An error value of
    ' 9.9E+37 may be returned over the bus in this situation.

    ' oscp.DoCommand(":DIGITIZE CHANnel1")
End Sub
```

```

' Analyze()
' -----
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
'   time.
' - Save the oscilloscope display to a file which can be
'   printed.
' - Make single channel measurements.

Private Shared Sub Analyze()

    ' Results array.
    Dim ResultsArray As Byte()
    ' Number of bytes returned from instrument.
    Dim nLength As Integer

    ' 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,
    '     #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    Console.WriteLine("Saving oscilloscope setup to " _
        + "c:\scope\config\setup.dat")
    If File.Exists("c:\scope\config\setup.dat") Then
        File.Delete("c:\scope\config\setup.dat")
    End If

    ' Query and read setup string.
    nLength = oscp.DoQueryIEEEBlock(":SYSTem:SETup?", ResultsArray)
    Console.WriteLine("Read oscilloscope setup ({0} bytes).", _
        nLength)

    ' Write setup string to file.
    File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
    Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
        nLength)

    ' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
    ' string to the oscilloscope.
    Dim DataArray As Byte()
    Dim nBytesWritten As Integer

    ' Read setup string from file.
    DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
    Console.WriteLine("Read setup string ({0} bytes) from file.", _
        DataArray.Length)

    ' Restore setup string.
    nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup", _
        DataArray)
    Console.WriteLine("Restored setup string ({0} bytes).", _
        nBytesWritten)

    ' IMAGE_TRANSFER - In this example, we query for the screen
    ' data with the ":DISPLAY:DATA?" query. The .png format

```

## 12 Programming Examples

```
' data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to " _
+ "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
oscp.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
nLength = _
    oscp.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, SCReen, COLOR", _
    ResultsArray)
Console.WriteLine("Read screen image ({0} bytes).", nLength)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
    nLength)

' Return I/O timeout to five seconds.
oscp.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to
' make measurements on displayed waveforms.

' Set source to measure.
oscp.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = oscp.DoQueryValue(":MEASure:FREQuency?")
Console.WriteLine("The frequency is: {0:F4} kHz", _
    fResults / 1000)

' Query for peak to peak voltage.
fResults = oscp.DoQueryValue(":MEASure:VPP?")
Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
    fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. 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:PREAMBLE?" query provides information concerning
' the vertical and horizontal scaling of the waveform data.
'
' With the preamble information you can then use the
' ":WAVEFORM:DATA?" query and read the data block in the
' correct format.

' WAVE_FORMAT - Sets the data transmission mode for waveform
' data output. This command controls how the data is
' formatted when sent from the oscilloscope and can be set
```

```

' to WORD or BYTE format.

' Set waveform format to BYTE.
oscp.DoCommand(":WAVEform:FORMAT BYTE")

' WAVE_POINTS - Sets the number of points to be transferred.
' The number of time points available is returned by the
' "ACQUIRE:POINTS?" query. This can be set to any binary
' fraction of the total time points available.
oscp.DoCommand(":WAVEform:POINTS 1000")

' GET_PREAMBLE - The preamble contains all of the current
' WAVEFORM settings returned in the form <preamble block><NL>
' where the <preamble block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
'   YINCREMENT : float32 - voltage difference between data
'                     points.
'   YORIGIN    : float32 - value of the voltage at center
'                     screen.
'   YREFERENCE : int32 - data point where y-origin occurs.
Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double()
fResultsArray = oscp.DoQueryValues(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMAT: {0:e}", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: {0:e}", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINTs: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNT: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCREMENT: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREFERENCE: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCREMENT: {0:e}", fYincrement)

```

## 12 Programming Examples

```
Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YOrigin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YReference: {0:e}", fYreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVEform:SOURce" command.

' 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 block><NL>
'
' Where:
'
' <header> = #800002048      (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 which can be set using the
' ":WAVEFORM:POINTS" command. You may then read that number
' of bytes from the oscilloscope; then, read the following NL
' character to terminate the query.

' Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVEform:DATA?", ResultsArray)
Console.WriteLine("Read waveform data ({0} bytes).", nLength)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < 1000
    Console.WriteLine("Data point {0:d} = {1:f2} Volts at " +
        "{2:f10} Seconds", i, _
        (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, _
        (CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + 50
End While
```

```

' SAVE_WAVE_DATA - saves the waveform data to a CSV format
' file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter =
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To 999
    writer.WriteLine("{0:E}, {1:f6}", _
        (CSng(index) - fXreference) * fxincrement + fxorigin, _
        (CSng(ResultsArray(index)) - fyreference) * fyincrement _ 
        + fyorigin)
Next
writer.Close()
End Sub
End Class

Class VisaInstrument
    Private m_nResourceManager As Integer
    Private m_nSession As Integer
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)
        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA resource manager.
        OpenResourceManager()

        ' Open a VISA resource session.
        OpenSession()

        ' Clear the interface.
        Dim nViStatus As Integer
        nViStatus = visa32.viClear(m_nSession)
    End Sub

    Public Sub DoCommand(ByVal strCommand As String)
        ' Send the command.
        VisaSendCommandOrQuery(strCommand)

        ' Check for instrument errors (another command and result).
        CheckForInstrumentErrors(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

```

## 12 Programming Examples

```
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)

' Write command termination character.
nViStatus = visa32.viPrintf(m_nSession, "" & Chr(10) & "")
CheckVisaStatus(nViStatus)

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(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 instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)

' Return string results.
Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
' Send the query.
VisaSendCommandOrQuery(strQuery)

' Get the result string.
Dim fResults As Double
fResults = VisaGetResultValue()

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)

' Return string results.
Return fResults
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
' Send the query.
VisaSendCommandOrQuery(strQuery)
```

```

' Get the result string.
Dim fResultsArray As Double()
fResultsArray = VisaGetResultValues()

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(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.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(ResultsArray)

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return length
End Function

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As New StringBuilder(1000)
    Dim bFirstError As Boolean = True
    Do
        VisaSendCommandOrQuery(":SYSTem:ERRor?")
        strInstrumentError = VisaGetResultString()

        If strInstrumentError.ToString() <> _
            "+0, ""No error"" & Chr(10) & "" Then
            If bFirstError Then
                Console.WriteLine("ERROR(s) for command '{0}': ", _
                    strCommand)
                bFirstError = False
            End If
            Console.Write(strInstrumentError)
        End If
    Loop While strInstrumentError.ToString() <> _
        "+0, ""No error"" & Chr(10) & ""
End Sub

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)

```

## 12 Programming Examples

```
    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 VisaGetResultValue() 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 VisaGetResultValues() As Double()
    Dim fResultsArray As Double()
    fResultsArray = New Double(9) {}

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, _
        "%,10lf" & Chr(10) & "", fResultsArray)
    CheckVisaStatus(nViStatus)

    Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _ 
    As Byte()) As Integer
    ' Results array, big enough to hold a PNG.
    ResultsArray = New Byte(299999) {}
    Dim length As Integer
    ' Number of bytes returned from instrument.
    ' Set the default number of bytes that will be contained in
    ' the ResultsArray to 300,000 (300kB).
    length = 300000

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
        ResultsArray)
    CheckVisaStatus(nViStatus)

    ' Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
    CheckVisaStatus(nViStatus)
```

```

nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
CheckVisaStatus(nViStatus)

    Return length
End Function

Private Sub 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 COM Examples

- "VISA COM Example in Visual Basic" on page 744
- "VISA COM Example in C#" on page 754
- "VISA COM Example in Visual Basic .NET" on page 765

#### 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 most of the commonly used programming
' features of your Agilent oscilloscopes.
' -----
```

```
Option Explicit
```

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

```
' MAIN PROGRAM
' -----
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
```

```
' The commands sent to the oscilloscope are written in both long and
' short form. Both forms are acceptable.
'
```

```
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
```

```

' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -----
Sub Main()

On Error GoTo VisaComError

' Create the VISA COM I/O resource.
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' GPIB.
' Set myScope.IO = myMgr.Open("GPIB0::7::INSTR")

' LAN.
' Set myScope.IO = myMgr.Open("TCPIP0::a-mso6102-90541::inst0::INSTR")

' USB.
Set myScope.IO = myMgr.Open("USB0::2391::5970::30D3090541::0::INSTR")

' Initialize - Initialization will start the program with the
' oscilloscope in a known state.
Initialize

' Capture - After initialization, you must make waveform data
' available to analyze. To do this, capture the data using the
' DIGITIZE command.
Capture

' Analyze - Once the waveform has been captured, it can be analyzed.
' There are many parts of a waveform to analyze. This example shows
' some of the possible ways to analyze various parts of a waveform.
Analyze

Exit Sub

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

End Sub

' Initialize
' -----
' Initialize will start the program with the oscilloscope in a known
' state. This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
' - Oscilloscope
' - Channel 1 range
' - Display Grid
' - Timebase reference, range, and delay
' - Trigger mode and type
'
```

## 12 Programming Examples

```
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -----
Private Sub Initialize()

    On Error GoTo VisaComError

    ' Clear the interface.
    myScope.IO.Clear

    ' 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.

    ' AUTOSCALE - This command evaluates all the input signals and sets
    ' the correct conditions to display all of the active signals.

    ' Same as pressing the Autoscale key.
    myScope.WriteString ":AUTOSCALE"

    ' 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 ":CHAN1:PROBE 10"      ' Set Probe to 10:1.

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
    ' range value is 8 times the volts per division.

    ' Set the vertical range to 8 volts.
    myScope.WriteString ":CHANNEL1:RANGE 8"

    ' TIME_RANGE - Sets the full scale horizontal time in seconds. The
    ' range value is 10 times the time per division.

    ' Set the time range to 0.002 seconds.
    myScope.WriteString ":TIM:RANG 2e-3"

    ' TIME_REFERENCE - Possible values are LEFT and CENTER.
    ' - LEFT sets the display reference on time division from the left.
    ' - CENTER sets the display reference to the center of the screen.

    ' Set reference to center.
    myScope.WriteString ":TIMEBASE:REFERENCE CENTER"

    ' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
    ' TV trigger. Any channel can be selected.
    myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"

    ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
    ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.

    ' Set the trigger mode to EDGE.
    myScope.WriteString ":TRIGGER:MODE EDGE"
```

```

' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.
' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"

' The following commands are not executed and are shown for reference
' purposes only. To execute these commands, uncomment them.

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

' VIEW_BLANK - (not executed in this example)
' - VIEW turns on (starts displaying) a channel or pixel memory.
' - BLANK turns off (stops displaying) a channel or pixel memory.
' myScope.WriteString ":BLANK CHANNEL1"    ' Turn channel 1 off.
' myScope.WriteString ":VIEW CHANNEL1"     ' Turn channel 1 on.

' 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"

Exit Sub

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

End Sub

'

' Capture
' -----
' We will capture the waveform using the digitize command.
' -----


Private Sub Capture()

On Error GoTo VisaComError

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

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

' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface. Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.

```

## 12 Programming Examples

```
'  
' NOTE! The DIGITIZE command is highly recommended for triggering  
' modes other than SINGLE. This ensures that sufficient data is  
' available for measurement. If DIGITIZE is used with single mode,  
' the completion criteria may never be met. The number of points  
' gathered in Single mode is related to the sweep speed, memory  
' depth, and maximum sample rate. For example, take an oscilloscope  
' with a 1000-point memory, a sweep speed of 10 us/div (100 us  
' total time across the screen), and a 20 MSa/s maximum sample rate.  
' 1000 divided by 100 us equals 10 MSa/s. Because this number is  
' less than or equal to the maximum sample rate, the full 1000 points  
' will be digitized in a single acquisition. Now, use 1 us/div  
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;  
' because this is greater than the maximum sample rate by 5 times,  
' only 400 points (or 1/5 the points) can be gathered on a single  
' trigger. Keep in mind when the oscilloscope is running,  
' communication with the computer interrupts data acquisition.  
' Setting up the oscilloscope over the bus causes the data buffers  
' to be cleared and internal hardware to be reconfigured. If a  
' measurement is immediately requested, there may have not been  
' enough time for the data acquisition process to collect data,  
' and the results may not be accurate. An error value of 9.9E+37  
' may be returned over the bus in this situation.  
'  
myScope.WriteString ":DIGITIZE CHAN1"  
  
Exit Sub  
  
VisaComError:  
    MsgBox "VISA COM Error:" + vbCrLf + Err.Description  
  
End Sub  
  
'  
' Analyze  
'-----  
' In analyze, we will do the following:  
' - Save the system setup to a file and restore it.  
' - Save the waveform data to a file on the computer.  
' - Make single channel measurements.  
' - Save the oscilloscope display to a file that can be sent to a  
'   printer.  
'-----  
  
Private Sub Analyze()  
  
On Error GoTo VisaComError  
  
' 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,  
'     #800002204<setup string><NL>  
' where the setup string is 2204 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"
Close #1      ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult      ' Write data.
Close #1      ' Close file.

' 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, SCREEN, 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 file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , byteData      ' Write data.
Close #1      ' Close file.
myScope.IO.Timeout = 5000

' 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 strPath For Binary Access Read As #1      ' Open file for input.
Get #1, , varSetupString      ' Read data.
Close #1      ' Close file.
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"

' Query for frequency.
myScope.WriteString ":MEASURE:FREQUENCY?"
varQueryResult = myScope.ReadNumber      ' Read frequency.
MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(varQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
myScope.WriteString ":MEASURE:DUTYCYCLE?"
varQueryResult = myScope.ReadNumber      ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + _
      FormatNumber(varQueryResult, 3) + "%"

```

## 12 Programming Examples

```
' Query for risetime.
myScope.WriteString ":MEASURE:RISETIME?"
varQueryResult = myScope.ReadNumber      ' Read risetime.
MsgBox "Risetime:" + vbCrLf + _
      FormatNumber(varQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
myScope.WriteString ":MEASURE:VPP?"
varQueryResult = myScope.ReadNumber      ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + _
      FormatNumber(varQueryResult, 4) + " V"

' Query for Vmax.
myScope.WriteString ":MEASURE:VMAX?"
varQueryResult = myScope.ReadNumber      ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + _
      FormatNumber(varQueryResult, 4) + " V"

' 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, 2 = 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 difference between data points.
' YORIGIN      : float32 - value is the voltage at center screen.
' YREFERENCE    : int32 - specifies the data point where y-origin
'                   occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"      ' Query for the preamble.
Preamble() = myScope.ReadList      ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'           FormatNumber(dblXIncrement * 1000000) + _
'           " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'           FormatNumber(dblXOrigin * 1000000) + _
'           " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'           CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'           FormatNumber(sngYIncrement * 1000) + _
'           " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'           FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'           CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
           FormatNumber(lngVSteps * sngYIncrement / 8) + _
           " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
           FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
           FormatNumber(lngPoints * dblXIncrement / 10 * _

```

## 12 Programming Examples

```
    1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
    FormatNumber(((lngPoints / 2) * _
        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) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"
```

```

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

Exit Sub

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

End Sub

Private Sub CheckForInstrumentErrors()

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 2005:

- 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 most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -----
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
    class VisaComInstrumentApp
    {
        private static VisaComInstrument myScope;

        public static void Main(string[] args)
```

```

{
try
{
    myScope = new
        VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR");

    Initialize();

    /* The extras function contains miscellaneous commands that
     * do not need to be executed for the proper operation of
     * this example. The commands in the extras function are
     * shown for reference purposes only.
    */
    // Extra();    // Uncomment to execute the extra function.
    Capture();
    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()
* -----
* This function initializes both the interface and the
* oscilloscope to a known state.
*/
private static void Initialize()
{
    string strResults;

    /* 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.DoCommand("*RST");    // Reset the to the defaults.
    myScope.DoCommand("*CLS");    // Clear the status data structures.

    /* IDN - Ask for the device's *IDN string.

```

## 12 Programming Examples

```
/*
strResults = myScope.DoQueryString("*IDN?");

// Display results.
Console.WriteLine("Result is: {0}", strResults);

/* AUTOSCALE - This command evaluates all the input signals
 * and sets the correct conditions to display all of the
 * active signals.
*/
myScope.DoCommand(":AUToscale");

/* CHANNEL_PROBE - Sets the probe attenuation factor for the
 * selected channel. The probe attenuation factor may be from
 * 0.1 to 1000.
*/
myScope.DoCommand(":CHANnel1:PROBe 10");

/* CHANNEL_RANGE - Sets the full scale vertical range in volts.
 * The range value is eight times the volts per division.
*/
myScope.DoCommand(":CHANnel1:RANGE 8");

/* TIME_RANGE - Sets the full scale horizontal time in seconds.
 * The range value is ten times the time per division.
*/
myScope.DoCommand(":TIMEbase:RANGE 2e-3");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
 * - LEFT sets the display reference one time division from
 *   the left.
 * - CENTER sets the display reference to the center of the
 *   screen.
*/
myScope.DoCommand(":TIMEbase:REFERENCE CENTER");

/* TRIGGER_SOURCE - Selects the channel that actually produces
 * the TV trigger. Any channel can be selected.
*/
myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH,
 * PATTERN, CAN, DURATION, IIC, LIN, SEQUENCE, SPI, TV,
 * UART, or USB.
*/
myScope.DoCommand(":TRIGger:MODE EDGE");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
 * trigger to either POSITIVE or NEGATIVE.
*/
myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
}

/*
* Extra()
* -----
* The commands in this function are not executed and are shown
```

```

* for reference purposes only. To execute these commands, call
* this function from main.
*/
private static void Extra()
{
    /* RUN_STOP (not executed in this example):
     * - RUN starts the acquisition of data for the active
     *   waveform display.
     * - STOP stops the data acquisition and turns off AUTOSTORE.
     */
    myScope.DoCommand(":RUN");
    myScope.DoCommand(":STOP");

    /* VIEW_BLANK (not executed in this example):
     * - VIEW turns on (starts displaying) an active channel or
     *   pixel memory.
     * - BLANK turns off (stops displaying) a specified channel or
     *   pixel memory.
     */
    myScope.DoCommand(":BLANK CHANnel1");
    myScope.DoCommand(":VIEW CHANnel1");

    /* TIME_MODE (not executed in this example) - Set the time base
     * mode to MAIN, DELAYED, XY or ROLL.
     */
    myScope.DoCommand(":TIMEbase:MODE MAIN");
}

/*
* Capture()
* -----
* This function prepares the scope for data acquisition and then
* uses the DIGITIZE MACRO to capture some data.
*/
private static void Capture()
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    myScope.DoCommand(":ACQuire:TYPE NORMAL");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    myScope.DoCommand(":ACQuire:COMPLETE 100");

    /* DIGITIZE - Used to acquire the waveform data for transfer
     * over the interface. Sending this command causes an
     * acquisition to take place with the resulting data being
     * placed in the buffer.
     */
    /* NOTE! The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement. Keep in mind when the oscilloscope is running,

```

## 12 Programming Examples

```
* communication with the computer interrupts data acquisition.  
* Setting up the oscilloscope over the bus causes the data  
* buffers to be cleared and internal hardware to be  
* reconfigured.  
* If a measurement is immediately requested there may not have  
* been enough time for the data acquisition process to collect  
* data and the results may not be accurate. An error value of  
* 9.9E+37 may be returned over the bus in this situation.  
*/  
myScope.DoCommand(":DIGITIZE CHANnel1");  
}  
  
/*  
 * Analyze()  
 * -----  
 * In this example we will do the following:  
 * - Save the system setup to a file for restoration at a later  
 *   time.  
 * - Save the oscilloscope display to a file which can be  
 *   printed.  
 * - Make single channel measurements.  
 */  
private static void Analyze()  
{  
    byte[] ResultsArray; // Results array.  
    int nBytes; // Number of bytes returned from instrument.  
  
    /* 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,  
     *      #800002204<setup string><NL>  
     * where the setup string is 2204 bytes in length.  
     */  
    Console.WriteLine("Saving oscilloscope setup to " +  
        "c:\\scope\\config\\setup.dat");  
    if (File.Exists("c:\\scope\\config\\setup.dat"))  
        File.Delete("c:\\scope\\config\\setup.dat");  
  
    // Query and read setup string.  
    ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETUp?");  
    nBytes = ResultsArray.Length;  
    Console.WriteLine("Read oscilloscope setup ({0} bytes).",  
        nBytes);  
  
    // Write setup string to file.  
    File.WriteAllBytes("c:\\scope\\config\\setup.dat",  
        ResultsArray);  
    Console.WriteLine("Wrote setup string ({0} bytes) to file.",  
        nBytes);  
  
    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup  
     * string to the oscilloscope.  
     */  
    byte[] DataArray;  
  
    // Read setup string from file.
```

```

dataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
Console.WriteLine("Read setup string ({0} bytes) from file.",
    dataArray.Length);

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETUp", dataArray);
Console.WriteLine("Restored setup string.");

/* IMAGE_TRANSFER - In this example, we query for the screen
 * data with the ":DISPLAY:DATA?" query. The .png format
 * data is saved to a file in the local file system.
 */
Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
ResultsArray = myScope.DoQueryIEEEBlock(
    ":DISPLAY:DATA? PNG, SCReen, COLOR");
nBytes = ResultsArray.Length;
Console.WriteLine("Read screen image ({0} bytes).", nBytes);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
    ResultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.",
    nBytes);

// Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */
Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

// Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. To
 * obtain waveform data, you must specify the WAVEFORM
 * parameters for the waveform data prior to sending the
 * ":WAVEFORM:DATA?" query.
 */

```

## 12 Programming Examples

```
/*
 * Once these parameters have been sent, the
 * ":WAVEFORM:PREAMBLE?" query provides information concerning
 * the vertical and horizontal scaling of the waveform data.
 *
 * With the preamble information you can then use the
 * ":WAVEFORM:DATA?" query and read the data block in the
 * correct format.
 */

/* WAVE_FORMAT - Sets the data transmission mode for waveform
 * data output. This command controls how the data is
 * formatted when sent from the oscilloscope and can be set
 * to WORD or BYTE format.
 */

// Set waveform format to BYTE.
myScope.DoCommand(":WAVEform:FORMAT BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
 * The number of time points available is returned by the
 * "ACQUIRE:POINTS?" query. This can be set to any binary
 * fraction of the total time points available.
 */
myScope.DoCommand(":WAVEform:POINTS 1000");

/* GET_PREAMBLE - The preamble contains all of the current
 * WAVEFORM settings returned in the form <preamble block><NL>
 * where the <preamble block> is:
 *   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
 *   YINCREMENT : float32 - voltage difference between data
 *                       points.
 *   YORIGIN    : float32 - value of the voltage at center
 *                       screen.
 *   YREFERENCE : int32 - data point where y-origin occurs.
 */
Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = myScope.DoQueryValues(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMAT: {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: {0:e}", fType);

double fPoints = fResultsArray[2];
```

```

Console.WriteLine("Preamble POINTs: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: {0:e}", fCount);

double fxincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fxincrement);

double fxorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fxorigin);

double fxreference = fResultsArray[6];
Console.WriteLine("Preamble XREFERENCE: {0:e}", fxreference);

double fyincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fyincrement);

double fyorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fyorigin);

double fyreference = fResultsArray[9];
Console.WriteLine("Preamble YREFERENCE: {0:e}", fyreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEform:SOURce" command.
 */

/* 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 block><NL>
 *
 * Where:
 *
 *      <header> = #800002048    (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 which can be set using the
 * ":WAVEFORM:POINTS" command. You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?");
nBytes = ResultsArray.Length;
Console.WriteLine("Read waveform data ({0} bytes).", nBytes);

// Make some calculations from the preamble data.
double fVdiv = 32 * fyincrement;
double fOffset = fyorigin;
double fSdiv = fPoints * fxincrement / 10;

```

## 12 Programming Examples

```
double fDelay = (fPoints / 2) * fxincrement + fxorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fvdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fsdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < nBytes; i = i + (nBytes / 20))
{
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at "
        + "{2:f10} Seconds", i,
        ((float)ResultsArray[i] - fyreference) * fyincrement +
        fyorigin,
        ((float)i - fxreference) * fxincrement + fxorigin);
}

/* SAVE_WAVE_DATA - saves the waveform data to a CSV format
 * file named "waveform.csv".
 */
if (File.Exists("c:\\scope\\data\\waveform.csv"))
    File.Delete("c:\\scope\\data\\waveform.csv");

StreamWriter writer =
    File.CreateText("c:\\scope\\data\\waveform.csv");
for (int i = 0; i < nBytes; i++)
{
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fxreference) * fxincrement + fxorigin,
        ((float)ResultsArray[i] - fyreference) * fyincrement +
        fyorigin);
}
writer.Close();
Console.WriteLine("Waveform data ({0} points) written to " +
    "c:\\scope\\data\\waveform.csv.", nBytes);
}

}

class VisaComInstrument
{
    private ResourceManagerClass mResourceManager;
    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 instrument errors.
    CheckForInstrumentErrors(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 instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return results string.
    return strResults;
}

public double DoQueryValue(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result number.
    double fResult;
    fResult = (double)m_IoObject.ReadNumber(
        IEEEASCIIType.ASCIIType_R8, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return result number.
    return fResult;
}

public double[] DoQueryValues(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 instrument errors.
    CheckForInstrumentErrors(strQuery);
}

```

## 12 Programming Examples

```
// Return result numbers.  
    return fResultsArray;  
}  
  
public byte[] DoQueryIEEEBlock(string strQuery)  
{  
    // Send the query.  
    m_IoObject.WriteString(strQuery, true);  
  
    // Get the results array.  
    byte[] ResultsArray;  
    ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(  
        IEEEBinaryType.BinaryType_UI1, false, true);  
  
    // Check for instrument errors.  
    CheckForInstrumentErrors(strQuery);  
  
    // Return results array.  
    return ResultsArray;  
}  
  
public void DoCommandIEEEBlock(string strCommand,  
    byte[] DataArray)  
{  
    // Send the command.  
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);  
  
    // Check for instrument errors.  
    CheckForInstrumentErrors(strCommand);  
}  
  
private void CheckForInstrumentErrors(string strCommand)  
{  
    string strInstrumentError;  
    bool bFirstError = true;  
  
    // Repeat until all errors are displayed.  
    do  
    {  
        // Send the ":SYSTem:ERRor?" query, and get the result string.  
        m_IoObject.WriteString(":SYSTem:ERRor?", true);  
        strInstrumentError = m_IoObject.ReadString();  
  
        // If there is an error, print it.  
        if (strInstrumentError.ToString() != "+0,\"No error\"\n")  
        {  
            if (bFirstError)  
            {  
                // Print the command that caused the error.  
                Console.WriteLine("ERROR(s) for command '{0}': ",  
                    strCommand);  
                bFirstError = false;  
            }  
            Console.Write(strInstrumentError);  
        }  
    } while (strInstrumentError.ToString() != "+0,\"No error\"\n");  
}
```

```
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 2005:

- 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 most of the commonly used programming  
' features of your Agilent oscilloscopes.  
'-----  
  
Imports System  
Imports System.IO  
Imports System.Text  
Imports InfiniiVision.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::2391::5957::MY47250010::0::INSTR")  
  
                Initialize()  
  
                ' The extras function contains miscellaneous commands that  
                ' do not need to be executed for the proper operation of  
                ' this example. The commands in the extras function are  
                ' shown for reference purposes only.  
  
                ' Extra(); // Uncomment to execute the extra function.  
                Capture()  
                Analyze()  
            Catch err As System.ApplicationException  
                Console.WriteLine("**** VISA Error Message : " + err.Message)  
            End Try  
        End Sub  
    End Class  
End Namespace
```

```

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()
' -----
' This function initializes both the interface and the
' oscilloscope to a known state.

Private Shared Sub Initialize()
    Dim strResults As String

    ' 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.

    ' Reset to the defaults.
    myScope.DoCommand("*RST")

    ' Clear the status data structures.
    myScope.DoCommand("*CLS")

    ' IDN - Ask for the device's *IDN string.
    strResults = myScope.DoQueryString("*IDN?")

    ' Display results.
    Console.Write("Result is: {0}", strResults)

    ' AUTOSCALE - This command evaluates all the input signals
    ' and sets the correct conditions to display all of the
    ' active signals.
    myScope.DoCommand(":AUToscale")

    ' CHANNEL_PROBE - Sets the probe attenuation factor for the
    ' selected channel. The probe attenuation factor may be from
    ' 0.1 to 1000.
    myScope.DoCommand(":CHANnel1:PROBe 10")

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
    ' The range value is eight times the volts per division.
    myScope.DoCommand(":CHANnel1:RANGE 8")

    ' TIME_RANGE - Sets the full scale horizontal time in seconds.
    ' The range value is ten times the time per division.
    myScope.DoCommand(":TIMEbase:RANGE 2e-3")

    ' TIME_REFERENCE - Possible values are LEFT and CENTER:
    ' - LEFT sets the display reference one time division from

```

## 12 Programming Examples

```
'      the left.  
'      - CENTER sets the display reference to the center of the  
'      screen.  
myScope.DoCommand(":TIMEbase:REFERENCE CENTER")  
  
' TRIGGER_SOURCE - Selects the channel that actually produces  
' the TV trigger. Any channel can be selected.  
myScope.DoCommand(":TRIGGER:TV:SOURCe CHANnel1")  
  
' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH,  
' PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,  
' UART, or USB.  
myScope.DoCommand(":TRIGGER:MODE EDGE")  
  
' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the  
' trigger to either POSITIVE or NEGATIVE.  
myScope.DoCommand(":TRIGGER:EDGE:SLOPe POSitive")  
  
End Sub  
  
'  
' Extra()  
' -----  
' The commands in this function are not executed and are shown  
' for reference purposes only. To execute these commands, call  
' this function from main.  
'  
  
Private Shared Sub Extra()  
    ' RUN_STOP (not executed in this example):  
    ' - RUN starts the acquisition of data for the active  
    ' waveform display.  
    ' - STOP stops the data acquisition and turns off AUTOSTORE.  
    '  
    myScope.DoCommand(":RUN")  
    myScope.DoCommand(":STOP")  
  
    ' VIEW_BLANK (not executed in this example):  
    ' - VIEW turns on (starts displaying) an active channel or  
    ' pixel memory.  
    ' - BLANK turns off (stops displaying) a specified channel or  
    ' pixel memory.  
    '  
    myScope.DoCommand(":BLANK CHANnel1")  
    myScope.DoCommand(":VIEW CHANnel1")  
  
    ' TIME_MODE (not executed in this example) - Set the time base  
    ' mode to MAIN, DELAYED, XY or ROLL.  
    '  
    myScope.DoCommand(":TIMEbase:MODE MAIN")  
End Sub  
  
' Capture()  
' -----
```

```

' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.

Private Shared Sub Capture()
    ' AQUIRE_TYPE - Sets the acquisition mode. There are three
    ' acquisition types NORMAL, PEAK, or AVERAGE.
    myScope.DoCommand(":ACQuire:TYPE NORMAL")

    ' 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.DoCommand(":ACQuire:COMPlete 100")

    ' DIGITIZE - Used to acquire the waveform data for transfer
    ' over the interface. Sending this command causes an
    ' acquisition to take place with the resulting data being
    ' placed in the buffer.

    ' NOTE! The use of the DIGITIZE command is highly recommended
    ' as it will ensure that sufficient data is available for
    ' measurement. Keep in mind when the oscilloscope is running,
    ' communication with the computer interrupts data acquisition.
    ' Setting up the oscilloscope over the bus causes the data
    ' buffers to be cleared and internal hardware to be
    ' reconfigured.
    ' If a measurement is immediately requested there may not have
    ' been enough time for the data acquisition process to collect
    ' data and the results may not be accurate. An error value of
    ' 9.9E+37 may be returned over the bus in this situation.
    myScope.DoCommand(":DIGItize CHANnel1")

End Sub

' Analyze()
' -----
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
'   time.
' - Save the oscilloscope display to a file which can be
'   printed.
' - Make single channel measurements.

Private Shared Sub Analyze()
    ' Results array.
    Dim ResultsArray As Byte()

    ' Number of bytes returned from instrument.
    Dim nBytes As Integer

    ' 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,
    '     #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    Console.WriteLine("Saving oscilloscope setup to " + _

```

## 12 Programming Examples

```
"c:\scope\config\setup.dat")
If File.Exists("c:\scope\config\setup.dat") Then
    File.Delete("c:\scope\config\setup.dat")
End If

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETUp?")
nBytes = ResultsArray.Length
Console.WriteLine("Read oscilloscope setup ({0} bytes).", nBytes)

' Write setup string to file.
File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
    nBytes)

' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
' string to the oscilloscope.
Dim DataArray As Byte()

' Read setup string from file.
DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
Console.WriteLine("Read setup string ({0} bytes) from file.", _
    DataArray.Length)

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETUp", DataArray)
Console.WriteLine("Restored setup string.")

' IMAGE_TRANSFER - In this example, we query for the screen
' data with the ":DISPLAY:DATA?" query. The .png format
' data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to " + _
    "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
ResultsArray = _
    myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, SCReen, COlor")
nBytes = ResultsArray.Length
Console.WriteLine("Read screen image ({0} bytes).", nBytes)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
    nBytes)

' Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to
' make measurements on displayed waveforms.
```

```

' Set source to measure.
myScope.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = myScope.DoQueryValue(":MEASure:FREQuency?")
Console.WriteLine("The frequency is: {0:F4} kHz", _
    fResults / 1000)

' Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?")
Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
    fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. 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:PREAMBLE?" query provides information concerning
' the vertical and horizontal scaling of the waveform data.
'
' With the preamble information you can then use the
' ":WAVEFORM:DATA?" query and read the data block in the
' correct format.

' WAVE_FORMAT - Sets the data transmission mode for waveform
' data output. This command controls how the data is
' formatted when sent from the oscilloscope and can be set
' to WORD or BYTE format.

' Set waveform format to BYTE.
myScope.DoCommand(":WAveform:FORMAT BYTE")

' WAVE_POINTS - Sets the number of points to be transferred.
' The number of time points available is returned by the
' "ACQUIRE:POINTS?" query. This can be set to any binary
' fraction of the total time points available.
myScope.DoCommand(":WAveform:POINTS 1000")

' GET_PREAMBLE - The preamble contains all of the current
' WAVEFORM settings returned in the form <preamble block><NL>
' where the <preamble block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = 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 the x-origin.
'   YINCREMENT : float32 - voltage difference between data
'                     points.

```

## 12 Programming Examples

```
'      YORIGIN      : float32 - value of the voltage at center
'                           screen.
'      YREFERENCE : int32 - data point where y-origin occurs.

Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryValues(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMAT: {0:e}", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: {0:e}", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINTs: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNT: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

Dim fxorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORigin: {0:e}", fxorigin)

Dim fxreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREFerence: {0:e}", fxreference)

Dim fyincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCrement: {0:e}", fyincrement)

Dim fyorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORigin: {0:e}", fyorigin)

Dim fyreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREFerence: {0:e}", fyreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVEform:SOURce" command.

' 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 block><NL>
'
' Where:
'
'      <header> = #800002048      (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 which can be set using the
```

```

' " :WAVEFORM:POINTS" command. You may then read that number
' of bytes from the oscilloscope; then, read the following NL
' character to terminate the query.

' Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?")
nBytes = ResultsArray.Length
Console.WriteLine("Read waveform data ({0} bytes).", nBytes)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < nBytes
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at " + _
        "{2:f10} Seconds", i, _
        (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, (CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + (nBytes / 20)
End While

' SAVE_WAVE_DATA - saves the waveform data to a CSV format
' file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter = _
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To nBytes - 1
    writer.WriteLine("{0:E}, {1:f6}", _
        (CSng(index) - fXreference) * fXincrement + fXorigin, _
        (CSng(ResultsArray(index)) - fYreference) * fYincrement + fYorigin)
Next
writer.Close()
Console.WriteLine("Waveform data ({0} points) written to " + _
    "c:\scope\data\waveform.csv.", nBytes)
End Sub
End Class

Class VisaComInstrument
    Private m_Manager As ResourceManagerClass
    Private m_IoObject As FormattedIO488Class
    Private m_strVisaAddress As String

```

## 12 Programming Examples

```
' Constructor.
Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA COM IO object.
    OpenIo()

    ' Clear the interface.
    m_IoObject.IO.Clear()
End Sub

Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    m_IoObject.WriteString(strCommand, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(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 instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return results string.
    Return strResults
End Function

Public Function DoQueryValue(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 instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return result number.
    Return fResult
End Function

Public Function DoQueryValues(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 instrument errors.
CheckForInstrumentErrors(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.
Dim ResultsArray As Byte()
ResultsArray = _
    m_IoObject.ReadIEEEBlock(IEEETweetType.BinaryType_UI1, _
    False, True)

' Check for instrument errors.
CheckForInstrumentErrors(strQuery)

' Return results array.
Return ResultsArray
End Function

Public _
Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte())
' Send the command.
m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

' Check for instrument errors.
CheckForInstrumentErrors(strCommand)
End Sub

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
Dim strInstrumentError As String
Dim bFirstError As Boolean = True

' Repeat until all errors are displayed.
Do
    ' Send the ":SYSTem:ERRor?" query, and get the result string.
    m_IoObject.WriteString(":SYSTem:ERRor?", True)
    strInstrumentError = m_IoObject.ReadString()

    ' If there is an error, print it.
    If strInstrumentError.ToString() <> "+0,""No error"" _ 
        & Chr(10) & "" Then
        If bFirstError Then
            ' Print the command that caused the error.
            Console.WriteLine("ERROR(s) for command '{0}': ", _
                strCommand)
            bFirstError = False
        End If
    End If
End Sub

```

## 12 Programming Examples

```
        Console.WriteLine(strInstrumentError)
    End If
Loop While strInstrumentError.ToString() <> "+0, ""No error"" _ 
    & Chr(10) & ""
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
```

# Index

## Symbols

+9.9E+37, infinity representation, 677  
+9.9E+37, measurement error, 272

## Numerics

0 (zero) values in waveform data, 509  
1 (one) values in waveform data, 509  
82350A GPIB interface, 4

## A

AC coupling, trigger edge, 440  
AC input coupling for specified channel, 193  
acknowledge, 610  
ACQuire commands, 163  
acquire data, 132, 177  
acquire mode on autoscale, 128  
acquire reset conditions, 111  
acquire sample rate, 176  
ACQuire subsystem, 47  
acquired data points, 170  
acquisition anti-alias control, 165  
acquisition count, 167  
acquisition mode, 163, 169, 526  
acquisition type, 163, 177  
active printer, 248  
add function, 521  
add math function, 237  
add math function as g(t) source, 233  
address field size, IIC serial decode, 381  
address, IIC trigger pattern, 454  
Addresses softkey, 34  
AER (Arm Event Register), 125, 147, 149, 640  
Agilent Connection Expert, 35  
Agilent Interactive IO application, 39  
Agilent IO Control icon, 35  
Agilent IO Libraries Suite, 4, 31, 44, 46  
Agilent IO Libraries Suite, installing, 32  
ALB waveform data format, 370  
ALL segments waveform save option, 372  
alphabetical list of commands, 535  
AMASK commands, 536  
amplitude, vertical, 306  
analog channel coupling, 193  
analog channel display, 194  
analog channel impedance, 195  
analog channel input, 570  
analog channel inversion, 196  
analog channel labels, 197, 214  
analog channel offset, 198  
analog channel protection lock, 396

analog channel range, 204  
analog channel scale, 205  
analog channel source for glitch, 452  
analog channel units, 206  
analog probe attenuation, 199  
analog probe sensing, 571  
analog probe skew, 201, 569  
analyzing captured data, 43  
angle brackets, 93  
annotate channels, 197  
anti-alias control, 165  
AREA commands, 536  
area for hardcopy print, 247  
area for saved image, 361  
Arm Event Register (AER), 125, 147, 149, 640  
ASCII format, 511  
ASCII format for data transfer, 502  
ASCII string, quoted, 93  
ASCIix waveform data format, 370  
assign channel names, 197  
attenuation factor (external trigger) probe, 222  
attenuation for oscilloscope probe, 199  
AUT option for probe sense, 571, 575  
auto trigger sweep mode, 411  
automask create, 323  
automask source, 324  
automask units, 325  
automatic measurements constants, 199  
automatic probe type detection, 571, 575  
Automation-Ready CD, 32  
autoscale, 126  
autoscale acquire mode, 128  
autoscale channels, 129  
AUToscale command, 46  
AVERage commands, 536  
average value measurement, 307  
averaging acquisition type, 164, 502  
averaging, synchronizing with, 654

## B

bandwidth filter limits, 220  
bandwidth filter limits to 20 MHz, 192  
BASE commands, 536  
base value measurement, 308  
base, UART trigger, 487  
basic instrument functions, 99  
baud rate, 429, 465, 488  
BAUDrate commands, 536  
begin acquisition, 132, 156, 158  
BHARris window for minimal spectral leakage, 244  
binary block data, 93, 397, 509

BINary waveform data format, 370  
bind levels for masks, 344  
bit order, 489  
bit weights, 104  
bitmap display, 211  
bits in Service Request Enable Register, 116  
bits in Standard Event Status Enable Register, 103  
bits in Status Byte Register, 118  
blank, 130  
block data, 93, 107, 211, 397  
block response data, 50  
blocking synchronization, 649  
blocking wait, 648  
BMP (bitmap) hardcopy format, 581  
braces, 92  
built-in measurements, 43  
button disable, 395  
BWLimit commands, 537  
byte format for data transfer, 502, 511  
BYTeorder, 507

## C

C#, VISA COM example, 754  
C#, VISA example, 717  
C, SICL library example, 680  
C, VISA library example, 698  
CAL PROTECT switch, 179, 186  
calculating preshoot of waveform, 289  
calculating the waveform overshoot, 285  
calibrate, 181, 182, 186, 188  
CALibrate commands, 179  
calibrate date, 181  
calibrate introduction, 179  
calibrate label, 182  
calibrate output, 183  
calibrate start, 184  
calibrate status, 185  
calibrate switch, 186  
calibrate temperature, 187  
calibrate time, 188  
CAN, 424  
CAN acknowledge, 428, 610  
CAN baud rate, 429  
CAN commands, 537  
CAN frame counters, reset, 377  
CAN id pattern, 426  
CAN signal definition, 611  
CAN source, 430  
CAN trigger, 425, 431  
CAN trigger commands, 422  
CAN trigger pattern id mode, 427

- capture data, 132  
capturing data, 42  
CDISplay, 131  
center frequency set, 230, 231  
center of screen, 534  
center reference, 405  
center screen, vertical value at, 236, 239  
channel, 162, 197  
CHANnel commands, 189, 190  
channel coupling, 193  
channel display, 194  
channel input impedance, 195  
channel inversion, 196  
channel label, 197, 568  
channel labels, 213, 214  
channel overload, 203  
channel probe ID, 223  
channel protection, 203  
channel reset conditions, 111  
channel selected to produce trigger, 452, 483  
channel signal type, 202  
channel skew for oscilloscope probe, 201, 569  
channel status, 159  
channel vernier, 207  
channel, stop displaying, 130  
channels to autoscale, 129  
channels, how autoscale affects, 126  
characters to display, 393  
classes of input signals, 244  
classifications, command, 658  
clear, 210  
CLEar commands, 538  
clear display, 131  
clear markers, 274, 586  
clear measurement, 274, 586  
clear message queue, 101  
Clear method, 45  
clear screen, 573  
clear status, 101  
clear waveform area, 208  
clipped high waveform data value, 509  
clipped low waveform data value, 509  
clock, 457, 471, 472, 476  
CLOCK commands, 538  
CLS (Clear Status), 101  
CME (Command Error) status bit, 103, 105  
code, \*RST, 113  
code, :ACQuire:COMPLETE, 166  
code, :ACQuire:SEGMENTed, 173  
code, :ACQuire:TYPE, 178  
code, :AUToscale, 127  
code, :CHANnel:LABel, 197  
code, :CHANnel:PROBe, 199  
code, :CHANnel:RANGE, 204  
code, :DIGItize, 132  
code, :DISPLAY:DATA, 212  
code, :DISPLAY:LABel, 213  
code, :MEASure:PERiod, 297  
code, :MEASure:REsults, 291  
code, :MEASure:TEDGe, 303  
code, :MTESt, 320  
code, :RUN/:STOP, 156  
code, :SYSTem:SETup, 397  
code, :TIMEbase:DElay, 609  
code, :TIMEbase:MODE, 402  
code, :TIMEbase:RANGE, 404  
code, :TIMEbase:REFerence, 405  
code, :TRIGger:MODE, 417  
code, :TRIGger:SLOPe, 443  
code, :TRIGger:SOURce, 444  
code, :VIEW and :BLANK, 162  
code, :WAVeform, 521  
code, :WAVeform:DATA, 509  
code, :WAVeform:POINTs, 513  
code, :WAVeform:PREamble, 517  
code, :WAVeform:SEGMENTed, 173  
code, SICL library example in C, 680  
code, SICL library example in Visual Basic, 689  
code, VISA COM library example in C#, 754  
code, VISA COM library example in Visual Basic, 744  
code, VISA COM library example in Visual Basic .NET, 765  
code, VISA library example in C, 698  
code, VISA library example in C#, 717  
code, VISA library example in Visual Basic, 707  
code, VISA library example in Visual Basic .NET, 731  
colon, root commands prefixed by, 124  
color palette for hardcopy, 253  
color palette for image, 365  
Comma Separated Values (CSV) hardcopy format, 581  
Comma Separated Values (CSV) waveform data format, 370  
command classifications, 658  
command errors detected in Standard Event Status, 105  
command header, 660  
command headers, common, 662  
command headers, compound, 661  
command headers, simple, 661  
command strings, valid, 659  
command tree, 663  
commands by subsystem, 95  
commands in alphabetical order, 535  
commands quick reference, 55  
commands sent over interface, 99  
commands, more about, 657  
commands, obsolete and discontinued, 563  
common (\*) commands, 96, 97, 99  
common command headers, 662  
completion criteria for an acquisition, 166, 167  
compound command headers, 661  
compound header, 675  
computer control examples, 679  
conditions for external trigger, 218  
conditions, reset, 111  
configurations, oscilloscope, 107, 110, 114, 397  
Configure softkey, 34  
connect oscilloscope, 33  
connect sampled data points, 572  
constants for making automatic measurements, 199  
constants for scaling display factors, 199  
constants for setting trigger levels, 199  
Control softkey, 33, 34  
controller initialization, 42  
copy display, 155  
core commands, 658  
count, 508  
COUNT commands, 538  
count values, 167  
counter, 275  
coupling, 440  
COUPling commands, 539  
coupling for channels, 193  
create automask, 323  
CSV (Comma Separated Values) hardcopy format, 581  
CSV (Comma Separated Values) waveform data format, 370  
current oscilloscope configuration, 107, 110, 114, 397  
current probe, 206, 227  
CURRent segment waveform save option, 372  
cursor mode, 258  
cursor position, 259, 261, 263, 264, 266  
cursor readout, 587, 591, 592  
cursor reset conditions, 111  
cursor source, 260, 262  
cursor time, 587, 591, 592  
cursors track measurements, 296  
cursors, how autoscale affects, 126  
cursors, X1, X2, Y1, Y2, 257  
cycle measured, 281  
cycle time, 287

**D**

- data, 424, 455, 458, 474, 477, 509  
data 2, 456  
DATA commands, 539  
data conversion, 502  
data displayed, 211  
data format for transfer, 502  
data output order, 507  
data pattern length, 425  
data pattern width, 475  
data point index, 531  
data points, 170  
data required to fill time buckets, 166  
data structures, status reporting, 626  
data transfer, 211  
data, erasing, 131  
data, saving and recalling, 208  
DATE commands, 539  
date, calibration, 181  
date, system, 392  
dB versus frequency, 230  
DC coupling for edge trigger, 440  
DC input coupling for specified channel, 193  
dc RMS measured on waveform, 313

DDE (Device Dependent Error) status bit, [103](#), [105](#)  
 decision chart, status reporting, [646](#)  
 default conditions, [111](#)  
 define channel labels, [197](#)  
 define glitch trigger, [450](#)  
 define measurement, [277](#)  
 define measurement source, [297](#)  
 define trigger, [419](#), [434](#), [435](#), [436](#), [438](#), [451](#)  
 defined as, [92](#)  
 definite-length block query response, [50](#)  
 definite-length block response data, [93](#)  
 DELay commands, [539](#)  
 delay measured to calculate phase, [288](#)  
 delay measurement, [277](#)  
 delay measurements, [302](#)  
 delay parameters for measurement, [279](#)  
 delay, how autoscale affects, [126](#)  
 delayed time base, [402](#)  
 delayed time base mode, how autoscale affects, [126](#)  
 delayed window horizontal scale, [410](#)  
 delete mask, [333](#)  
 delta time, [587](#)  
 delta voltage measurement, [596](#)  
 delta X cursor, [257](#)  
 delta Y cursor, [257](#)  
 DeskJet, [579](#)  
 destination, [216](#)  
 detecting probe types, [571](#), [575](#)  
 device for hardcopy, [579](#)  
 device-defined error queue clear, [101](#)  
 differential signal type, [202](#), [224](#)  
 differentiate math function, [168](#), [230](#), [237](#), [521](#)  
 DIFFerentiate source for function, [242](#), [576](#)  
 digital channel data, [502](#)  
 digital channel labels, [214](#)  
 digital pod, stop displaying, [130](#)  
 digitize channels, [132](#)  
 DIGItize command, [43](#), [47](#)  
 digits, [93](#)  
 disable anti-alias mode, [168](#)  
 disable front panel, [395](#)  
 disable function, [577](#)  
 disabling calibration, [186](#)  
 disabling channel display, [194](#)  
 disabling status register bits, [102](#), [115](#)  
 discontinued and obsolete commands, [563](#)  
 display clear, [210](#)  
 DISPLAY commands, [208](#), [540](#)  
 display commands introduction, [208](#)  
 display connect, [572](#)  
 display data, [211](#)  
 display date, [392](#)  
 display factors scaling, [199](#)  
 display for channels, [194](#)  
 display frequency span, [243](#)  
 display measurements, [272](#), [296](#)  
 display persistence, [215](#)  
 display reference, [403](#), [405](#)  
 display reset conditions, [111](#)  
 display serial number, [157](#)

display source, [216](#)  
 display vectors, [217](#)  
 display, clearing, [131](#)  
 display, oscilloscope, [215](#), [216](#), [217](#), [232](#), [393](#)  
 display, serial decode bus, [380](#)  
 displaying a baseline, [421](#)  
 displaying unsynchronized signal, [421](#)  
 DNS IP, [33](#)  
 domain, [33](#)  
 Domain softkey, [34](#)  
 driver, printer, [584](#)  
 duplicate mnemonics, [674](#)  
 duration, [434](#), [435](#), [438](#)  
 duration for glitch trigger, [446](#), [447](#), [451](#)  
 duration pattern, [436](#)  
 duration qualifier, trigger, [434](#), [435](#), [437](#)  
 DURation trigger commands, [433](#)  
 duration triggering, [411](#)  
 duty cycle measurement, [43](#), [272](#), [281](#)

**E**

edge coupling, [440](#)  
 edge define, [419](#)  
 edge fall time, [282](#)  
 edge parameter for delay measurement, [279](#)  
 edge preshoot measured, [289](#)  
 edge rise time, [294](#)  
 edge slope, [443](#)  
 edge source, [444](#)  
 EDGE trigger commands, [439](#)  
 edge triggering, [411](#)  
 edges in measurement, [277](#)  
 elapsed time in mask test, [330](#)  
 ellipsis, [93](#)  
 enable channel labels, [213](#)  
 enabling calibration, [186](#)  
 enabling channel display, [194](#)  
 enabling status register bits, [102](#), [115](#)  
 end of string (EOS) terminator, [660](#)  
 end of text (EOT) terminator, [660](#)  
 end or identify (EOI), [660](#)  
 enter pattern, [419](#)  
 EOI (end or identify), [660](#)  
 EOS (end of string) terminator, [660](#)  
 EOT (end of text) terminator, [660](#)  
 Epson, [579](#)  
 equivalent-time acquisition mode, [164](#), [169](#)  
 erase data, [131](#), [210](#)  
 erase functions, [131](#)  
 erase measurements, [586](#)  
 erase screen, [573](#)  
 ERRor commands, [540](#)  
 error frame count (CAN), [375](#)  
 error frame count (UART), [386](#)  
 error messages, [394](#), [615](#)  
 error number, [394](#)  
 error queue, [394](#), [637](#)  
 error, measurement, [272](#)  
 ESB (Event Status Bit), [116](#), [118](#)  
 ESE (Standard Event Status Enable Register), [102](#), [636](#)

ESR (Standard Event Status Register), [104](#), [635](#)  
 EVENT commands, [540](#)  
 event status conditions occurred, [118](#)  
 Event Status Enable Register (ESE), [102](#), [636](#)  
 Event Status Register (ESR), [104](#), [161](#), [635](#)  
 example code, \*RST, [113](#)  
 example code, :ACQuire:COMplete, [166](#)  
 example code, :ACQuire:SEGmented, [173](#)  
 example code, :ACQuire:TYPE, [178](#)  
 example code, :AUToscale, [127](#)  
 example code, :CHANnel:LABel, [197](#)  
 example code, :CHANnel:PROBe, [199](#)  
 example code, :CHANnel:RANGE, [204](#)  
 example code, :DIGitize, [132](#)  
 example code, :DISPlay:DATA, [212](#)  
 example code, :DISPlay:LABel, [213](#)  
 example code, :MEASure:PERiod, [297](#)  
 example code, :MEASure:RESults, [291](#)  
 example code, :MEASure:TEDGE, [303](#)  
 example code, :MTEST, [320](#)  
 example code, :RUN/STOP, [156](#)  
 example code, :SYSTem:SETup, [397](#)  
 example code, :TIMEbase:DELay, [609](#)  
 example code, :TIMEbase:MODE, [402](#)  
 example code, :TIMEbase:RANGE, [404](#)  
 example code, :TIMEbase:REFERENCE, [405](#)  
 example code, :TRIGger:MODE, [417](#)  
 example code, :TRIGger:SLOPe, [443](#)  
 example code, :TRIGger:SOURce, [444](#)  
 example code, :VIEW and :BLANK, [162](#)  
 example code, :WAVeform, [521](#)  
 example code, :WAVeform:DATA, [509](#)  
 example code, :WAVeform:POINTS, [513](#)  
 example code, :WAVeform:PREamble, [517](#)  
 example code, :WAVeform:SEGmented, [173](#)  
 example programs, [4](#), [679](#)  
 EXE (Execution Error) status bit, [103](#), [105](#)  
 execution error detected in Standard Event Status, [105](#)  
 exponential notation, [92](#)  
 external glitch trigger source, [452](#)  
 external range, [226](#)  
 external trigger, [218](#), [221](#), [222](#), [444](#), [574](#)  
 EXternal trigger commands, [218](#)  
 external trigger input impedance, [221](#), [574](#)  
 EXternal trigger level, [441](#)  
 external trigger overload, [225](#)  
 external trigger probe attenuation factor, [222](#)  
 external trigger probe ID, [223](#)  
 external trigger probe sensing, [575](#)  
 external trigger protection, [225](#)  
 external trigger signal type, [224](#)  
 EXternal trigger source, [444](#)  
 external trigger units, [227](#)

**F**

FACTion commands, [541](#)  
 FACTors commands, [541](#)  
 fail (mask test) output, [336](#)  
 failed waveforms in mask test, [328](#)  
 failure, self test, [120](#)

fall time measurement, 272, 282  
falling edge, 419  
Fast Fourier Transform (FFT) functions, 230, 231, 242, 243, 244, 576  
FF values in waveform data, 509  
FFT (Fast Fourier Transform) functions, 230, 231, 242, 243, 244, 576  
FFT (Fast Fourier Transform) operation, 237, 521  
FFT math function, 168  
fifty ohm impedance, disable setting, 396  
Filename commands, 541  
filename for hardcopy, 580  
filename for recall, 352  
filename for save, 359  
filter for frequency reject, 442  
filter for high frequency reject, 415  
filter for noise reject, 418  
filter used to limit bandwidth, 192, 220  
filters to Fast Fourier Transforms, 244  
fine horizontal adjustment (vernier), 407  
fine vertical adjustment (vernier), 207  
finish pending device operations, 108  
first point displayed, 531  
FLATtop window for amplitude measurements, 244  
format, 511, 516  
FORMAT commands, 541  
format for block data, 107  
format for generic video, 480, 484  
format for hardcopy, 578, 581  
format for image, 363  
format for waveform data, 370  
FormattedIO488 object, 45  
formfeed for hardcopy, 246, 250  
frame, 478  
frame counters (CAN), error, 375  
frame counters (CAN), overload, 376  
frame counters (CAN), reset, 377  
frame counters (CAN), total, 378  
frame counters (UART), error, 386  
frame counters (UART), reset, 387  
frame counters (UART), Rx frames, 388  
frame counters (UART), Tx frames, 389  
framing, 473  
FRAMING commands, 541  
frequency measurement, 43, 272, 283  
frequency resolution, 244  
frequency span of display, 243  
frequency versus dB, 230  
front panel mode, 421  
front panel Single key, 158  
front panel Stop key, 160  
front-panel lock, 395  
full-scale horizontal time, 404, 409  
full-scale vertical axis defined, 238  
function, 162, 231, 232, 236, 237, 238, 239, 240, 242, 243, 244, 576, 577  
FUNCTION commands, 228  
function memory, 159  
function turned on or off, 577  
functions, 521

functions, erasing, 131

## G

g(t) source, first input channel, 234  
g(t) source, math operation, 233  
g(t) source, second input channel, 235  
gateway IP, 33  
general trigger commands, 414  
GENeric, 480, 484  
generic video format, 480, 484  
glitch duration, 451  
glitch qualifier, 450  
glitch source, 452  
GLITCH trigger commands, 445  
glitch trigger duration, 446  
glitch trigger polarity, 449  
glitch trigger source, 446  
GOFT commands, 542  
GPIB interface, 33, 34  
graphics, 211  
graticule area for hardcopy print, 247  
graticule area for saved image, 361  
graticule colors, invert for hardcopy, 251, 583  
graticule colors, invert for image, 364  
graticule data, 211  
grayscale palette for hardcopy, 253  
grayscale palette for image, 365  
grayscaling on hardcopy, 582  
greater than qualifier, 450  
greater than time, 434, 438, 446, 451  
GREaterthan commands, 542

## H

HANNing window for frequency resolution, 244  
hardcopy, 155, 246  
HARDcopy commands, 245  
hardcopy device, 579  
hardcopy factors, 249, 362  
hardcopy filename, 580  
hardcopy format, 578, 581  
hardcopy formfeed, 250  
hardcopy grayscale, 582  
hardcopy invert graticule colors, 251, 583  
hardcopy layout, 252  
hardcopy palette, 253  
hardcopy print, area, 247  
hardcopy printer driver, 584  
hardware event condition register, 136  
Hardware Event Condition Register (:HWEregister:CONDition), 136  
Hardware Event Condition Register (:OPERegister:CONDition), 643  
Hardware Event Enable Register (HWEenable), 134  
hardware event event register, 138  
Hardware Event Event Register (:HWEregister[:EVENT]), 138, 642  
header, 660

high-frequency reject filter, 415, 442  
high-resolution acquisition type, 164  
hold until operation complete, 108  
holdoff time, 416  
holes in waveform data, 509  
horizontal adjustment, fine (vernier), 407  
horizontal position, 408  
horizontal scale, 406, 410  
horizontal scaling, 516  
horizontal time, 404, 409, 587  
hostname, 33  
HWEenable (Hardware Event Enable Register), 134  
HWEregister:CONDition (Hardware Event Condition Register), 136, 643  
HWEregister[:EVENT] (Hardware Event Event Register), 138, 642

## I

I/O softkey, 33, 34  
I1080L50HZ, 480, 484  
I1080L60HZ, 480, 484  
ID commands, 543  
id mode, 427  
identification number, 106  
identification of options, 109  
identifier, 426  
identifier, LIN, 463  
idle until operation complete, 108  
IDN (Identification Number), 106  
IEEE 488.2 standard, 99  
IGColors commands, 543  
IIC address, 454  
IIC clock, 457  
IIC commands, 543  
IIC data, 455, 458  
IIC data 2, 456  
IIC serial decode address field size, 381  
IIC trigger commands, 453  
IIC trigger qualifier, 459  
IIC trigger type, 460  
IMAGE commands, 543  
image format, 363  
image invert graticule colors, 364  
image memory, 159, 216  
image palette, 365  
image, recall, 353  
image, save, 360  
image, save with inksaver, 364  
impedance, 195  
IMPedance commands, 543  
impedance for external trigger input, 221, 574  
infinity representation, 677  
initialization, 42, 45  
initialize, 111  
initialize label list, 214  
initiate acquisition, 132  
inksaver, save image with, 364  
input, 221, 574  
input coupling for channels, 193  
input impedance for channels, 195, 570

input impedance for external trigger, 221, 574  
 input inversion for specified channel, 196  
 insert label, 197  
 installed options identified, 109  
 instruction header, 660  
 instrument number, 106  
 instrument options identified, 109  
 instrument requests service, 118  
 instrument serial number, 157  
 instrument settings, 246  
 instrument status, 52  
 instrument type, 106  
 integrate math function, 230, 237, 521  
 INTegrate source for function, 242, 576  
 INTERN files, 216  
 internal low-pass filter, 192, 220  
 introduction to :ACQuire commands, 163  
 introduction to :CALibrate commands, 179  
 introduction to :CHANnel commands, 190  
 introduction to :DISPlay commands, 208  
 introduction to :EXTernal commands, 218  
 introduction to :FUNCtion commands, 230  
 introduction to :HARDcopy commands, 246  
 introduction to :MARKer commands, 257  
 introduction to :MEASure commands, 272  
 introduction to :RECall commands, 351  
 introduction to :SAVE commands, 358  
 introduction to :SBUS commands, 374  
 introduction to :SYSTem commands, 391  
 introduction to :TIMEbase commands, 400  
 introduction to :TRIGger commands, 411  
 introduction to :WAVeform commands, 502  
 introduction to common (\*) commands, 99  
 introduction to root (: commands, 124  
 invert graticule colors for hardcopy, 251, 583  
 invert graticule colors for image, 364  
 inverted masks, bind levels, 344  
 inverting input for channels, 196  
 IO library, referencing, 44  
 IP address, 33  
 IP Options softkey, 34

## K

key disable, 395  
 key press detected in Standard Event Status Register, 105  
 knob disable, 395  
 known state, 111

## L

label, 568  
 LABel commands, 543  
 label list, 197, 214  
 labels, 197, 213, 214  
 labels to store calibration information, 182  
 labels, specifying, 208  
 LAN interface, 33, 36  
 LAN Settings softkey, 34  
 landscape layout for hardcopy, 252

language for program examples, 41  
 LaserJet, 579  
 layout for hardcopy, 252  
 leakage into peak spectrum, 244  
 learn string, 107, 397  
 least significant byte first, 507  
 left reference, 405  
 legal values for channel offset, 198  
 legal values for frequency span, 243  
 legal values for offset, 236, 239  
 LENGTH commands, 544  
 length for waveform data, 371  
 less than qualifier, 450  
 less than time, 435, 438, 447, 451  
 LESSthan commands, 544  
 LEVel commands, 544  
 level for trigger voltage, 441, 448  
 LF coupling, 440  
 license information, 109  
 limits for line number, 480  
 LIN acknowledge, 464  
 LIN baud rate, 465  
 LIN identifier, 463  
 LIN serial decode bus parity bits, 382  
 LIN source, 466  
 LIN standard, 467  
 LIN sync break, 468  
 LIN trigger, 469  
 LIN trigger commands, 462  
 LIN trigger definition, 612  
 line glitch trigger source, 452  
 line number for TV trigger, 480  
 line terminator, 92  
 LINE trigger level, 441  
 LINE trigger source, 444  
 list of channel labels, 214  
 load utilization (CAN), 379  
 local lockout, 395  
 lock, 395  
 LOCK commands, 544  
 lock mask to signal, 335  
 lock, analog channel protection, 396  
 lockout message, 395  
 long form, 660  
 lower threshold, 287  
 lower threshold voltage for measurement, 585  
 lowercase characters in commands, 659  
 low-frequency reject filter, 442  
 low-pass filter used to limit bandwidth, 192, 220  
 LRN (Learn Device Setup), 107  
 lsbfirst, 507

## M

magnitude of occurrence, 304  
 main sweep range, 408  
 main time base, 609  
 main time base mode, 402  
 making measurements, 272  
 MAN option for probe sense, 571, 575  
 manual cursor mode, 258

MARKer commands, 256  
 marker mode, 264  
 marker position, 265  
 marker readout, 591, 592  
 marker set for voltage measurement, 597, 598  
 marker sets start time, 588  
 marker time, 587  
 markers for delta voltage measurement, 596  
 markers track measurements, 296  
 markers, command overview, 257  
 markers, mode, 258  
 markers, time at start, 592  
 markers, time at stop, 591  
 markers, X delta, 263  
 markers, X1 position, 259  
 markers, X1Y1 source, 260  
 markers, X2 position, 261  
 markers, X2Y2 source, 262  
 markers, Y delta, 266  
 markers, Y1 position, 264  
 markers, Y2 position, 265  
 mask, 102, 115, 419, 436  
 MASK commands, 544  
 mask statistics, reset, 329  
 mask test commands, 318  
 Mask Test Event Enable Register (MTEenable), 141  
 mask test event event register, 143  
 Mask Test Event Event Register (:MTERegister[:EVENT]), 143, 644  
 mask test output, 336  
 mask test run mode, 337  
 mask test termination conditions, 337  
 mask test, enable/disable, 334  
 mask, delete, 333  
 mask, get as binary block data, 332  
 mask, load from binary block data, 332  
 mask, lock to signal, 335  
 mask, recall, 354  
 mask, save, 366  
 masks, bind levels, 344  
 master summary status bit, 118  
 math function, stop displaying, 130  
 math operations, 230  
 MAV (Message Available), 101, 116, 118  
 maximum duration, 434, 435, 447  
 maximum position, 403  
 maximum range for zoomed window, 409  
 maximum scale for zoomed window, 410  
 maximum vertical value measurement, 309  
 maximum vertical value, time of, 316, 589  
 MEASure commands, 267  
 measure overshoot, 285  
 measure period, 287  
 measure phase between channels, 288  
 measure preshoot, 289  
 measure start voltage, 597  
 measure stop voltage, 598  
 measure value at a specified time, 314  
 measure value at top of waveform, 315  
 measurement error, 272  
 measurement setup, 272, 297

- measurement source, 297  
measurement statistics results, 291  
measurements, average value, 307  
measurements, base value, 308  
measurements, built-in, 43  
measurements, clear, 274, 586  
measurements, command overview, 272  
measurements, counter, 275  
measurements, dc RMS, 313  
measurements, definition setup, 277  
measurements, delay, 279  
measurements, duty cycle, 281  
measurements, fall time, 282  
measurements, frequency, 283  
measurements, how autoscale affects, 126  
measurements, lower threshold level, 585  
measurements, maximum vertical value, 309  
measurements, maximum vertical value, time of, 316, 589  
measurements, minimum vertical value, 310  
measurements, minimum vertical value, time of, 317, 590  
measurements, overshoot, 285  
measurements, period, 287  
measurements, phase, 288  
measurements, preshoot, 289  
measurements, pulse width, negative, 284  
measurements, pulse width, positive, 290  
measurements, ratio of AC RMS values, 312  
measurements, resetting, 131  
measurements, rise time, 294  
measurements, show, 296  
measurements, source channel, 297  
measurements, standard deviation, 295  
measurements, start marker time, 591  
measurements, stop marker time, 592  
measurements, thresholds, 588  
measurements, time between start and stop markers, 587  
measurements, time between trigger and edge, 302  
measurements, time between trigger and vertical value, 304  
measurements, time between trigger and voltage level, 593  
measurements, upper threshold value, 595  
measurements, vertical amplitude, 306  
measurements, vertical peak-to-peak, 311  
measurements, voltage difference, 596  
memory setup, 114, 397  
merge, 140  
message available bit, 118  
message available bit clear, 101  
message displayed, 118  
message error, 615  
message queue, 634  
messages ready, 118  
midpoint of thresholds, 287  
minimum duration, 434, 435, 438, 446  
minimum vertical value measurement, 310  
minimum vertical value, time of, 317, 590  
mnemonics, duplicate, 674  
mode, 169, 177, 258, 402, 481  
MODE commands, 546  
mode, serial decode, 383  
model number, 106  
models, oscilloscope, 3  
modes for triggering, 417  
Modify softkey, 34  
monochrome palette for image, 365  
most significant byte first, 507  
move, 230  
move cursors, 591, 592  
msbfirst, 507  
MSG (Message), 116, 118  
MSS (Master Summary Status), 118  
MTEvent (Mask Test Event Enable Register), 141  
MTERegister[:EVENTn] (Mask Test Event Event Register), 143, 644  
MTEst commands, 318  
multiple commands, 675  
multiple queries, 51  
multiply math function, 230, 237, 521  
multiply math function as g(t) source, 233
- ## N
- name channels, 197  
name list, 214  
negative glitch trigger polarity, 449  
negative pulse width, 284  
negative pulse width measurement, 43  
negative slope, 443, 471  
negative TV trigger polarity, 482  
new line (NL) terminator, 92, 660  
NL (new line) terminator, 92, 660  
noise reject filter, 418  
non-core commands, 658  
non-interlaced GENeric mode, 484  
non-volatile memory, label list, 214  
normal acquisition type, 163, 502  
normal trigger sweep mode, 411  
notices, 2  
NR1 number format, 92  
NR3 number format, 92  
NTSC, 480, 484  
NULL string, 393  
number format, 92  
number of points, 170, 512, 514  
number of time buckets, 512, 514  
numeric variables, 50  
numeric variables, reading query results into multiple, 52  
nwidth, 284
- ## O
- obsolete and discontinued commands, 563  
obsolete commands, 658  
occurrence reported by magnitude, 593  
offset, 230  
OFFSet commands, 547
- offset value for channel voltage, 198  
offset value for selected function, 236, 239  
one values in waveform data, 509  
OPC (Operation Complete) command, 108  
OPC (Operation Complete) status bit, 103, 105  
OPEE (Operation Status Enable Register), 145  
Open method, 45  
operating configuration, 107, 397  
operating state, 114  
OPERation commands, 548  
operation complete, 108  
operation status condition register, 147  
Operation Status Condition Register (:OPERRegister:CONDition), 147, 639  
operation status conditions occurred, 118  
Operation Status Enable Register (OPEE), 145  
operation status event register, 149  
Operation Status Event Register (:OPERRegister[:EVENTn]), 149, 638  
operation, math, 230  
operations for function, 237  
OPERRegister:CONDition (Operation Status Condition Register), 147, 639  
OPERRegister[:EVENTn] (Operation Status Event Register), 149, 638  
OPT (Option Identification), 109  
optional syntax terms, 92  
options, 109  
order of output, 507  
oscilloscope connection, opening, 45  
oscilloscope connection, verifying, 35  
oscilloscope external trigger, 218  
oscilloscope models, 3  
oscilloscope rate, 176  
oscilloscope, connecting, 33  
oscilloscope, initialization, 42  
oscilloscope, operation, 4  
oscilloscope, program structure, 42  
oscilloscope, setting up, 33  
oscilloscope, setup, 46  
OUTPut commands, 548  
output messages ready, 118  
output queue, 108, 633  
output queue clear, 101  
output sequence, 507  
output, mask test, 336  
overlapped commands, 678  
overload, 203, 225  
Overload Event Enable Register (OVL), 151  
Overload Event Register (:OVLRegister), 641  
Overload Event Register (OVLR), 153  
overload frame count (CAN), 376  
overload protection, 151, 153  
overshoot of waveform, 285  
overvoltage, 203, 225  
OVL (Overload Event Enable Register), 151  
OVLR (Overload Event Register), 153  
OVLR bit, 138, 147, 149  
OVLRegister (Overload Event Register), 641

**P**

P1080L24HZ, 480, 484  
 P1080L25HZ, 480, 484  
 P480L60HZ, 480, 484  
 P720L60HZ, 480, 484  
 PAL, 480, 484  
 PALETTE commands, 548  
 palette for hardcopy, 253  
 palette for image, 365  
 PAL-M, 480, 484  
 parameters for delay measurement, 279  
 parametric measurements, 272  
 parity, 493  
 parity bits, LIN serial decode bus, 382  
 PARity commands, 548  
 parser, 124, 675  
 pass (mask test) output, 336  
 pass, self test, 120  
 path information, recall, 355  
 path information, save, 367  
 pattern, 419, 424, 426, 436, 454, 455, 456, 474  
 pattern and edge, 419  
 PATTern commands, 548  
 pattern duration, 434, 435, 446, 447  
 pattern length, 425  
 pattern trigger, 419  
 pattern triggering, 411  
 pattern width, 475  
 peak detect, 177  
 peak detect acquisition type, 164, 502  
 peaks, 230  
 peak-to-peak vertical value measurement, 311  
 pending operations, 108  
 percent of waveform overshoot, 285  
 percent thresholds, 277  
 period measured to calculate phase, 288  
 period measurement, 43, 272, 287  
 persistence, waveform, 208, 215  
 phase measured between channels, 288  
 phase measurements, 302  
 pixel memory, 216  
 pixel memory, saving display to, 140  
 PLL Locked bit, 136, 147  
 pod, 521  
 pod, stop displaying, 130  
 points, 170, 512, 514  
 POINts commands, 549  
 points in waveform data, 502  
 polarity, 482, 494  
 POLarity commands, 549  
 polarity for glitch trigger, 449  
 polling synchronization with timeout, 650  
 polling wait, 648  
 PON (Power On) status bit, 103, 105  
 portrait layout for hardcopy, 252  
 position, 261, 403, 408  
 POSition commands, 549  
 position cursors, 591, 592  
 position in zoomed view, 408  
 positive glitch trigger polarity, 449

positive pulse width, 290  
 positive pulse width measurement, 43  
 positive slope, 443, 471  
 positive TV trigger polarity, 482  
 positive width, 290  
 preamble data, 502, 516  
 present working directory, recall operations, 355  
 present working directory, save operations, 367  
 preset conditions, 111  
 preshoot measured on waveform, 289  
 previously stored configuration, 110  
 print command, 155  
 print job, start, 255  
 print mask test failures, 338  
 print query, 607  
 printer, 579  
 printer driver for hardcopy, 584  
 printer hardcopy format, 581  
 printer, active, 248  
 printing, 246  
 printing in grayscale, 582  
 probe, 441  
 probe attenuation affects channel voltage range, 204  
 probe attenuation factor (external trigger), 222  
 probe attenuation factor for selected channel, 199  
 PROBe commands, 549  
 probe ID, 200, 223  
 probe sense for oscilloscope, 571, 575  
 probe skew value, 201, 569  
 process sigma, mask test run, 341  
 program data, 660  
 program data syntax rules, 662  
 program initialization, 42  
 program message, 45, 99  
 program message syntax, 659  
 program message terminator, 660  
 program structure, 42  
 programming examples, 4, 679  
 protecting against calibration, 186  
 protection, 151, 153, 203, 225  
 PROTection commands, 549  
 protection lock, 396  
 pulse width, 284, 290  
 pulse width duration trigger, 446, 447, 451  
 pulse width measurement, 43, 272  
 pulse width trigger, 418  
 pulse width trigger level, 448  
 pulse width triggering, 411  
 PWD commands, 549  
 pwidht, 290

**Q**

qualifier, 451  
 QUALifier commands, 549  
 qualifier, trigger duration, 434, 435, 437  
 queries, multiple, 51  
 query error detected in Standard Event Status, 105

query responses, block data, 50  
 query responses, reading, 49  
 query results, reading into numeric variables, 50  
 query results, reading into string variables, 50  
 query return values, 677  
 query setup, 246, 257, 272, 397  
 query subsystem, 230  
 querying setup, 190  
 querying the subsystem, 411  
 queues, clearing, 645  
 quick reference, commands, 55  
 quoted ASCII string, 93  
 QYE (Query Error) status bit, 103, 105

**R**

range, 230, 409  
 RANGE commands, 550  
 range for channels, 204  
 range for duration trigger, 438  
 range for external trigger, 226  
 range for full-scale vertical axis, 238  
 range for glitch trigger, 451  
 range for time base, 404  
 range of offset values, 198  
 range qualifier, 450  
 ranges, value, 93  
 rate, 176  
 ratio of AC RMS values measured between channels, 312  
 RCL (Recall), 110  
 read configuration, 107  
 read trace memory, 211  
 ReadIEEEBlock method, 45, 49, 51  
 ReadList method, 45, 49  
 ReadNumber method, 45, 49  
 readout, 587  
 ReadString method, 45, 49  
 real-time acquisition mode, 164, 169  
 recall, 110, 351, 397  
 RECall commands, 351  
 recall filename, 352  
 recall image, 353  
 recall mask, 354  
 recall path information, 355  
 recall setup, 356  
 recalling and saving data, 208  
 RECTangular window for transient signals, 244  
 reference, 230, 405  
 REFERENCE commands, 550  
 reference for time base, 609  
 registers, 104, 110, 114, 125, 134, 136, 138, 141, 143, 145, 147, 149, 151, 153  
 registers, clearing, 645  
 reject filter, 442  
 reject high frequency, 415  
 reject noise, 418  
 remote control examples, 679  
 remove cursor information, 258  
 remove labels, 213  
 remove message from display, 393

- reorder channels, 126  
repetitive acquisitions, 156  
report errors, 394  
report transition, 302, 304  
reporting status, 623  
reporting the setup, 411  
request service, 118  
Request-for-OPC flag clear, 101  
reset, 111  
RESet commands, 550  
reset conditions, 111  
reset mask statistics, 329  
reset measurements, 131, 210  
resolution of printed copy, 582  
resource session object, 45  
ResourceManager object, 45  
restore configurations, 107, 110, 114, 397  
restore labels, 213  
restore setup, 110  
return values, query, 677  
returning acquisition type, 177  
returning number of data points, 170  
right reference, 405  
rise time measurement, 272  
rise time of positive edge, 294  
rising edge, 419  
RMODE commands, 550  
RMS value measurement, 313  
roll time base mode, 402  
root (:) commands, 122, 124  
root level commands, 96  
RQL (Request Control) status bit, 103, 105  
ROS (Request Service), 118  
RST (Reset), 111  
rules, tree traversal, 675  
rules, truncation, 660  
RUMode commands, 551  
run, 119, 156  
Run bit, 147, 149  
run mode, mask test, 337  
running configuration, 114, 397  
Rx frame count (UART), 388  
Rx source, 496
- S**
- sample rate, 176  
sampled data, 572  
sampled data points, 509  
SAMPLEpoint commands, 551  
SAV (Save), 114  
save, 114, 358  
SAVE commands, 357, 551  
save filename, 359  
save image, 360  
save image with inksaver, 364  
save mask, 366  
save mask test failures, 339  
save path information, 367  
save setup, 368  
SAVE TO INTERN, 140  
save waveform data, 369  
save waveforms to pixel memory, 140  
saved image, area, 361  
saving and recalling data, 208  
SBUS commands, 373  
scale, 240, 406, 410  
SCALe commands, 552  
scale factors output on hardcopy, 249, 362  
scale for channels, 205  
scale units for channels, 206  
scale units for external trigger, 227  
scaling display factors, 199  
SCPI commands, 53  
scratch measurements, 586  
screen area for hardcopy print, 247  
screen area for saved image, 361  
screen data, 211  
SECAM, 480, 484  
seconds per division, 406  
SEGmented commands, 552  
segmented waveform save option, 372  
segments, analyze, 171  
segments, count of waveform, 519  
segments, setting number of memory, 172  
segments, setting the index, 173  
segments, time tag, 520  
select measurement channel, 297  
self-test, 120  
sensing a channel probe, 571  
sensing a external trigger probe, 575  
sensitivity of oscilloscope input, 199  
sequence triggering, 411  
sequential commands, 678  
serial clock, 457, 476  
serial data, 458, 477  
serial decode bus, 374  
serial decode bus display, 380  
serial decode mode, 383  
serial frame, 478  
serial number, 157  
service request, 118  
Service Request Enable Register (SRE), 116, 631  
set, 111  
set center frequency, 231  
set conditions, 126  
set cursors, 591, 592  
set date, 392  
set delay, 126  
set thresholds, 126  
set time, 399  
set time/div, 126  
set up oscilloscope, 33  
setting display, 232  
setting external trigger level, 218  
setting impedance for channels, 195  
setting inversion for channels, 196  
settings, 110, 114  
settings, instrument, 246  
setup, 164, 190, 208, 230, 246, 397  
SETup commands, 552  
setup configuration, 110, 114, 397  
setup defaults, 111  
setup memory, 110  
setup reported, 411  
setup, recall, 356  
setup, save, 368  
short form, 3, 660  
show channel labels, 213  
show measurements, 272, 296  
SICL example in C, 680  
SICL example in Visual Basic, 689  
SICL examples, 680  
sigma, mask test run, 341  
SIGNal commands, 553  
signal type, 202, 224  
simple command headers, 661  
single acquisition, 158  
single-ended signal type, 202, 224  
single-shot DUT, synchronizing with, 652  
skew, 201, 569  
slope, 443, 471  
slope (direction) of waveform, 593  
SLOPe commands, 553  
slope not valid in TV trigger mode, 443  
slope parameter for delay measurement, 279  
software version, 106  
source, 216, 230, 297, 430, 466, 521  
SOURce commands, 553  
source for function, 241, 242, 576  
source for trigger, 444  
source for TV trigger, 483  
source, automask, 324  
source, mask test, 349  
SOURce1 commands, 553  
SOURce2 commands, 553  
span, 230  
span of frequency on display, 243  
specify measurement, 297  
SPI, 471, 472, 474  
SPI commands, 554  
SPI decode word width, 384  
SPI trigger, 473, 475  
SPI trigger clock, 476  
SPI trigger commands, 470  
SPI trigger data, 477  
SPI trigger frame, 478  
square root math function, 237  
SRE (Service Request Enable Register), 116, 631  
SRQ (Service Request interrupt), 134, 141, 145  
STANDARD commands, 554  
standard deviation measured on waveform, 295  
Standard Event Status Enable Register (ESE), 102, 636  
Standard Event Status Register (ESR), 104, 635  
standard for video, 484  
standard, LIN, 467  
start acquisition, 119, 132, 156, 158  
start and stop edges, 277  
STARt commands, 554  
start cursor, 591  
start measurement, 272  
start print job, 255

start time, 451, 591  
 start time marker, 588  
 state memory, 114  
 state of instrument, 107, 397  
 STATistics commands, 554  
 statistics increment, 300  
 statistics reset, 301  
 statistics results, 291  
 statistics, type of, 299  
 status, 117, 159, 161  
 Status Byte Register (STB), 115, 117, 118, 629  
 STA tus commands, 554  
 status data structure clear, 101  
 status registers, 52  
 status reporting, 623  
 STB (Status Byte Register), 115, 117, 118, 629  
 step size for frequency span, 243  
 stop, 132, 160  
 stop acquisition, 160  
 STOP commands, 554  
 stop cursor, 592  
 stop displaying channel, 130  
 stop displaying math function, 130  
 stop displaying pod, 130  
 stop on mask test failure, 340  
 stop time, 451, 592  
 storage, 114  
 store instrument setup, 107, 114  
 store setup, 114  
 store waveforms to pixel memory, 140  
 storing calibration information, 182  
 string variables, 50  
 string variables, reading multiple query results into, 51  
 string variables, reading query results into multiple, 51  
 string, quoted ASCII, 93  
 subnet mask, 33  
 subsource, waveform source, 525  
 subsystem commands, 96, 675  
 subtract math function, 230, 237, 521  
 subtract math function as g(t) source, 233  
 sweep mode, trigger, 411, 421  
 sweep speed set to fast to measure fall time, 282  
 sweep speed set to fast to measure rise time, 294  
 switch disable, 395  
 switch, calibration protect, 186  
 sync break, LIN, 468  
 syntax elements, 92  
 syntax rules, program data, 662  
 syntax, optional terms, 92  
 syntax, program message, 659  
 SYStem commands, 391  
 system commands, 392, 393, 394, 395, 397, 399  
 system commands introduction, 391

**T**

tdelta, 587

tedge, 302  
 telnet ports 5024 and 5025, 509  
 Telnet sockets, 53  
 temporary message, 393  
 TER (Trigger Event Register), 161, 632  
 termination conditions, mask test, 337  
 test sigma, mask test run, 341  
 test, self, 120  
 text, writing to display, 393  
 threshold voltage (lower) for measurement, 585  
 threshold voltage (upper) for measurement, 595  
 thresholds, 277, 588  
 thresholds used to measure period, 287  
 thresholds, how autoscale affects, 126  
 TIFF image format, 363  
 time base, 402, 403, 404, 405, 406, 609  
 time base commands introduction, 400  
 time base reset conditions, 111  
 time base window, 408, 409, 410  
 time between points, 587  
 time buckets, 166, 167  
 TIME commands, 555  
 time delay, 609  
 time delta, 587  
 time difference between data points, 529  
 time duration, 434, 435, 438, 451  
 time holdoff for trigger, 416  
 time interval, 302, 304, 587  
 time interval between trigger and occurrence, 593  
 time marker sets start time, 588  
 time per division, 404  
 time record, 244  
 time specified, 314  
 time, calibration, 188  
 time, mask test run, 342  
 time, start marker, 591  
 time, stop marker, 592  
 time, system, 399  
 time/div, how autoscale affects, 126  
 time-at-max measurement, 589  
 time-at-min measurement, 590  
 TIMEbase commands, 400  
 timebase vernier, 407  
 TIMEbase:MODE, 48  
 time-ordered label list, 214  
 timeout, 472  
 timing measurement, 272  
 title channels, 197  
 title, mask test, 350  
 tolerance, automask, 326, 327  
 top of waveform value measured, 315  
 total frame count (CAN), 378  
 total waveforms in mask test, 331  
 trace memories, how autoscale affects, 126  
 trace memory, 159, 162  
 trace memory data, 211  
 track measurements, 296  
 trademarks, 2  
 transfer instrument state, 107, 397

transmit, 211  
 tree traversal rules, 675  
 tree, command, 663  
 TRG (Trigger), 116, 118, 119  
 TRIG OUT BNC, 183  
 trigger (external) input impedance, 221, 574  
 trigger armed event register, 147, 149  
 trigger burst, UART, 490  
 TRIGger CAN commands, 422  
 trigger channel source, 452, 483  
 TRIGger commands, 411, 556  
 TRIGger commands, general, 414  
 trigger data, UART, 491  
 trigger duration, 434, 435  
 TRIGger DURation commands, 433  
 TRIGger EDGE commands, 439  
 trigger edge coupling, 440  
 trigger edge slope, 443  
 trigger event bit, 161  
 Trigger Event Register (TER), 632  
 TRIGger GLITCH commands, 445  
 trigger holdoff, 416  
 trigger idle, UART, 492  
 TRIGger IIC commands, 453  
 trigger level constants, 199  
 trigger level voltage, 441  
 TRIGger LIN commands, 462  
 trigger occurred, 118  
 trigger pattern, 419, 436  
 trigger qualifier, 437  
 trigger qualifier, UART, 495  
 trigger reset conditions, 111  
 trigger SPI clock slope, 471  
 TRIGger SPI commands, 470  
 trigger status bit, 161  
 trigger sweep mode, 411  
 TRIGger TV commands, 479  
 trigger type, UART, 498  
 TRIGger UART commands, 485  
 trigger, CAN, 431  
 trigger, CAN acknowledge, 610  
 trigger, CAN pattern data, 424  
 trigger, CAN pattern data length, 425  
 trigger, CAN pattern ID, 426  
 trigger, CAN pattern ID mode, 427  
 trigger, CAN sample point, 428  
 trigger, CAN signal baudrate, 429  
 trigger, CAN signal definition, 611  
 trigger, CAN source, 430  
 trigger, duration greater than, 434  
 trigger, duration less than, 435  
 trigger, duration pattern, 436  
 trigger, duration qualifier, 437  
 trigger, duration range, 438  
 trigger, edge coupling, 440  
 trigger, edge level, 441  
 trigger, edge reject, 442  
 trigger, edge slope, 443  
 trigger, edge source, 444  
 trigger, glitch greater than, 446  
 trigger, glitch less than, 447  
 trigger, glitch level, 448

- trigger, glitch polarity, 449  
trigger, glitch qualifier, 450  
trigger, glitch range, 451  
trigger, glitch source, 452  
trigger, high frequency reject filter, 415  
trigger, holdoff, 416  
trigger, IIC clock source, 457  
trigger, IIC data source, 458  
trigger, IIC pattern address, 454  
trigger, IIC pattern data, 455  
trigger, IIC pattern data 2, 456  
trigger, IIC qualifier, 459  
trigger, IIC signal baudrate, 465  
trigger, IIC type, 460  
trigger, LIN, 469  
trigger, LIN sample point, 464  
trigger, LIN signal definition, 612  
trigger, LIN source, 466  
trigger, mode, 417  
trigger, noise reject filter, 418  
trigger, pattern, 419  
trigger, SPI clock slope, 471  
trigger, SPI clock source, 476  
trigger, SPI clock timeout, 472  
trigger, SPI data source, 477  
trigger, SPI frame source, 478  
trigger, SPI framing, 473  
trigger, SPI pattern data, 474  
trigger, SPI pattern width, 475  
trigger, sweep, 421  
trigger, TV line, 480  
trigger, TV mode, 481, 613  
trigger, TV polarity, 482  
trigger, TV source, 483  
trigger, TV standard, 484  
trigger, UART base, 487  
trigger, UART baudrate, 488  
trigger, UART bit order, 489  
trigger, UART parity, 493  
trigger, UART polarity, 494  
trigger, UART Rx source, 496  
trigger, UART Tx source, 497  
trigger, UART width, 499  
truncation rules, 660  
TST (Self Test), 120  
tstart, 591  
tstop, 592  
turn function on or off, 577  
turn off channel, 130  
turn off channel labels, 213  
turn off cursors, 126  
turn off digital pod, 130  
turn off math function, 130  
turn off measurements, 126  
turn off trace memories, 126  
turn off zoomed time base mode, 126  
turn on channel labels, 213  
turn on channels, 126  
turning channel display on and off, 194  
turning off/on function calculation, 232  
turning vectors on or off, 572  
TV mode, 481, 613  
TV trigger commands, 479  
TV trigger line number setting, 480  
TV trigger mode, 483  
TV trigger polarity, 482  
TV trigger standard setting, 484  
TV triggering, 411  
tvmode, 613  
Tx data, UART, 525  
Tx frame count (UART), 389  
Tx source, 497  
type, 177, 526  
TYPE commands, 558
- U**
- UART base, 487  
UART baud rate, 488  
UART bit order, 489  
UART commands, 558  
UART frame counters, reset, 387  
UART parity, 493  
UART polarity, 494  
UART Rx source, 496  
UART trigger burst, 490  
UART trigger commands, 485  
UART trigger data, 491  
UART trigger idle, 492  
UART trigger qualifier, 495  
UART trigger type, 498  
UART Tx data, 525  
UART Tx source, 497  
UART width, 499  
UNITS commands, 559  
units per division, 205, 206, 227, 406  
units per division (vertical) for function, 205, 240  
units, automask, 325  
unsigned mode, 527  
upper threshold, 287  
upper threshold voltage for measurement, 595  
uppercase characters in commands, 659  
URQ (User Request) status bit, 103, 105  
USB (Device) interface, 33  
user defined channel labels, 197  
user event conditions occurred, 118  
User's Guide, 4  
USR (User Event bit), 116, 118  
Utility button, 33, 34  
utilization, CAN bus, 379
- V**
- valid command strings, 659  
valid pattern time, 434, 435  
value, 304  
value measured at base of waveform, 308  
value measured at specified time, 314  
value measured at top of waveform, 315  
value ranges, 93  
values required to fill time buckets, 167  
VBA, 44, 744
- vectors, 217  
vectors turned on or off, 572  
vectors, turning on or off, 208  
vernier, channel, 207  
vernier, horizontal, 407  
vertical adjustment, fine (vernier), 207  
vertical amplitude measurement, 306  
vertical axis defined by RANGE, 238  
vertical axis range for channels, 204  
vertical offset for channels, 198  
vertical peak-to-peak measured on waveform, 311  
vertical scale, 205, 240  
vertical scaling, 516  
vertical value at center screen, 236, 239  
vertical value maximum measured on waveform, 309  
vertical value measurements to calculate overshoot, 285  
vertical value minimum measured on waveform, 310  
video line to trigger on, 480  
video standard selection, 484  
view, 162, 230, 528  
view turns function on or off, 577  
VISA COM example in C#, 754  
VISA COM example in Visual Basic, 744  
VISA COM example in Visual Basic .NET, 765  
VISA example in C, 698  
VISA example in C#, 717  
VISA example in Visual Basic, 707  
VISA example in Visual Basic .NET, 731  
VISA examples, 698, 744  
Visual Basic .NET, VISA COM example, 765  
Visual Basic .NET, VISA example, 731  
Visual Basic 6.0, 45  
Visual Basic for Applications, 44, 744  
Visual Basic, SICL library example, 689  
Visual Basic, VISA COM example, 744  
Visual Basic, VISA example, 707  
voltage crossing reported or not found, 593  
voltage difference between data points, 532  
voltage difference measured, 596  
voltage level for active trigger, 441  
voltage marker used to measure waveform, 597, 598  
voltage offset value for channels, 198  
voltage probe, 206, 227  
voltage ranges for channels, 204  
voltage ranges for external trigger, 226  
voltage threshold, 277
- W**
- WAI (Wait To Continue), 121  
wait, 121  
wait for operation complete, 108  
Wait Trig bit, 147, 149  
waveform base value measured, 308  
Waveform command, 43  
Waveform commands, 500, 560  
waveform data, 502

waveform data format, 370  
 waveform data length, 371  
 waveform data, save, 369  
 waveform introduction, 502  
 waveform maximum vertical value measured, 309  
 waveform minimum vertical value measured, 310  
 waveform must cross voltage level to be an occurrence, 593  
 WAVEform parameters, 48  
 waveform peak-to-peak vertical value measured, 311  
 waveform period, 287  
 waveform persistence, 208  
 waveform RMS value measured, 313  
 waveform save option for segments, 372  
 waveform source channels, 521  
 waveform source subsource, 525  
 waveform standard deviation value measured, 295  
 waveform vertical amplitude, 306  
 waveform voltage measured at marker, 597, 598  
 waveform, byte order, 507  
 waveform, count, 508  
 waveform, data, 509  
 waveform, format, 511  
 waveform, points, 512, 514  
 waveform, preamble, 516  
 waveform, source, 521  
 waveform, type, 526  
 waveform, unsigned, 527  
 waveform, view, 528  
 waveform, X increment, 529  
 waveform, X origin, 530  
 waveform, X reference, 531  
 waveform, Y increment, 532  
 waveform, Y origin, 533  
 waveform, Y reference, 534  
 WAVEform:FORMat, 48  
 WAVEforms commands, 560  
 waveforms, mask test run, 343  
 Web control, 53  
 what's new, 19  
 width, 451, 499  
 WIDTh commands, 560  
 window, 408, 409, 410  
 window time, 404  
 window time base mode, 402  
 windows, 244  
 windows as filters to Fast Fourier Transforms, 244  
 windows for Fast Fourier Transform functions, 244  
 word format, 511  
 word format for data transfer, 502  
 word width, SPI decode, 384  
 write text to display, 393  
 write trace memory, 211  
 WriteIEEEBlock method, 45, 51  
 WriteList method, 45

WriteNumber method, 45  
 WriteString method, 45

**X**

X axis markers, 257  
 X delta, 263  
 X delta, mask scaling, 346  
 X1 and X2 cursor value difference, 263  
 X1 cursor, 257, 259, 260  
 X1, mask scaling, 345  
 X2 cursor, 257, 261, 262  
 X-axis functions, 400  
 XDELta commands, 561  
 X-increment, 529  
 X-of-max measurement, 316  
 X-of-min measurement, 317  
 X-origin, 530  
 X-reference, 531  
 X-Y mode, 400, 402

**Y**

Y axis markers, 257  
 Y1 and Y2 cursor value difference, 266  
 Y1 cursor, 257, 260, 264, 266  
 Y1, mask scaling, 347  
 Y2 cursor, 257, 262, 265, 266  
 Y2, mask scaling, 348  
 Y-axis value, 533  
 YDELta commands, 561  
 Y-increment, 532  
 Y-origin, 533, 534  
 Y-reference, 534

**Z**

zero values in waveform data, 509  
 zoomed time base, 402  
 zoomed time base mode, how autoscale affects, 126  
 zoomed window horizontal scale, 410

