

# Agilent InfiniiVision 2000 X-Series Oscilloscopes

	<p>205 Westwood Ave Long Branch, NJ 07740 1-877-742-TEST (8378) Fax: (732) 222-7088 salesteam@Tequipment.NET</p>
--------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------

## Programmer's Guide



Agilent Technologies

# Notices

© Agilent Technologies, Inc. 2005-2011

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 01.10.0000

## Edition

February 15, 2011

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 sub-contract, 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 2000 X-Series oscilloscopes:

**Table 1** InfiniiVision 2000 X-Series Oscilloscope Models

Channels	Input Bandwidth		
	70 MHz	100 MHz	200 MHz
4 analog + 8 digital (mixed signal)	MSO-X 2004A	MSO-X 2014A	MSO-X 2024A
2 analog + 8 digital (mixed signal)	MSO-X 2002A	MSO-X 2012A	MSO-X 2022A
4 analog	DSO-X 2004A	DSO-X 2014A	DSO-X 2024A
2 analog	DSO-X 2002A	DSO-X 2012A	DSO-X 2022A

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

- [Chapter 1](#), “What’s New,” starting on page 21, describes programming command changes in the latest version of oscilloscope software.
- [Chapter 2](#), “Setting Up,” starting on page 27, describes the steps you must take before you can program the oscilloscope.
- [Chapter 3](#), “Getting Started,” starting on page 37, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- [Chapter 4](#), “Commands Quick Reference,” starting on page 51, is a brief listing of the 2000 X-Series oscilloscope commands and syntax.

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

- [Chapter 5](#), “Common (\*) Commands,” starting on page 95, describes commands defined by the IEEE 488.2 standard that are common to all instruments.
- [Chapter 6](#), “Root (: ) Commands,” starting on page 121, describes commands that reside at the root level of the command tree and control many of the basic functions of the oscilloscope.
- [Chapter 7](#), “:ACQUIRE Commands,” starting on page 157, describes commands for setting the parameters used when acquiring and storing data.
- [Chapter 8](#), “:BUS<n> Commands,” starting on page 171, describes commands that control all oscilloscope functions associated with the digital channels bus display.
- [Chapter 9](#), “:CALIBRATE Commands,” starting on page 181, describes utility commands for determining the state of the calibration factor protection button.

- [Chapter 10](#), “:CHANnel<n> Commands,” starting on page 191, describes commands that control all oscilloscope functions associated with individual analog channels or groups of channels.
- [Chapter 11](#), “:DEMO Commands,” starting on page 211, describes commands that control the education kit (Option EDU) demonstration signals that can be output on the oscilloscope's Demo 1 and Demo 2 terminals.
- [Chapter 12](#), “:DIGital<d> Commands,” starting on page 217, describes commands that control all oscilloscope functions associated with individual digital channels.
- [Chapter 13](#), “:DISPlay Commands,” starting on page 225, describes commands that control how waveforms, graticule, and text are displayed and written on the screen.
- [Chapter 14](#), “:EXTernal Trigger Commands,” starting on page 233, describes commands that control the input characteristics of the external trigger input.
- [Chapter 15](#), “:FUNCTion Commands,” starting on page 239, describes commands that control math waveforms.
- [Chapter 16](#), “:HARDcopy Commands,” starting on page 257, describes commands that set and query the selection of hardcopy device and formatting options.
- [Chapter 17](#), “:MARKer Commands,” starting on page 275, describes commands that set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
- [Chapter 18](#), “:MEASure Commands,” starting on page 287, describes commands that select automatic measurements (and control markers).
- [Chapter 19](#), “:MTEST Commands,” starting on page 331, describes commands that control the mask test features provided with Option LMT.
- [Chapter 20](#), “:POD Commands,” starting on page 365, describes commands that control all oscilloscope functions associated with groups of digital channels.
- [Chapter 21](#), “:RECall Commands,” starting on page 371, describes commands that recall previously saved oscilloscope setups, reference waveforms, or masks.
- [Chapter 22](#), “:SAVE Commands,” starting on page 379, describes commands that save oscilloscope setups, screen images, and data.
- [Chapter 23](#), “:SYSTem Commands,” starting on page 397, describes commands that control basic system functions of the oscilloscope.
- [Chapter 24](#), “:TIMEbase Commands,” starting on page 411, describes commands that control all horizontal sweep functions.

- [Chapter 25](#), “:TRIGger Commands,” starting on page 423, describes commands that control the trigger modes and parameters for each trigger type.
- [Chapter 26](#), “:WAVEform Commands,” starting on page 459, describes commands that provide access to waveform data.
- [Chapter 27](#), “:WGEN Commands,” starting on page 495, describes commands that control waveform generator (Option WGN) functions and parameters.
- [Chapter 28](#), “:WMEMory<r> Commands,” starting on page 511, describes commands that control reference waveforms.
- [Chapter 29](#), “Obsolete and Discontinued Commands,” starting on page 521, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- [Chapter 30](#), “Error Messages,” starting on page 569, 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 31](#), “Status Reporting,” starting on page 577, describes the oscilloscope's status registers and how to check the status of the instrument.
- [Chapter 32](#), “Synchronizing Acquisitions,” starting on page 597, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- [Chapter 33](#), “More About Oscilloscope Commands,” starting on page 607, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- [Chapter 34](#), “Programming Examples,” starting on page 617.

### Mixed-Signal Oscilloscope Channel Differences

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

### See Also

- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Agilent IO Libraries Suite.

- For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Agilent 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/DSO-X2000manual>"

# Contents

In This Book 3

## 1 What's New

What's New in Version 1.10 22  
Version 1.00 at Introduction 23  
Command Differences From 7000B Series Oscilloscopes 24

## 2 Setting Up

Step 1. Install Agilent IO Libraries Suite software 28  
Step 2. Connect and set up the oscilloscope 29  
    Using the USB (Device) Interface 29  
    Using the LAN Interface 29  
    Using the GPIB Interface 30  
Step 3. Verify the oscilloscope connection 31

## 3 Getting Started

Basic Oscilloscope Program Structure 38  
    Initializing 38  
    Capturing Data 38  
    Analyzing Captured Data 39  
Programming the Oscilloscope 40  
    Referencing the IO Library 40  
    Opening the Oscilloscope Connection via the IO Library 41  
    Initializing the Interface and the Oscilloscope 41  
    Using :AUToscale to Automate Oscilloscope Setup 42  
    Using Other Oscilloscope Setup Commands 42  
    Capturing Data with the :DIGitize Command 43  
    Reading Query Responses from the Oscilloscope 45  
    Reading Query Results into String Variables 46  
    Reading Query Results into Numeric Variables 46  
    Reading Definite-Length Block Query Response Data 46  
    Sending Multiple Queries and Reading Results 47  
    Checking Instrument Status 48

Other Ways of Sending Commands	49
Telnet Sockets	49
Sending SCPI Commands Using Browser Web Control	49

## 4 Commands Quick Reference

Command Summary	52
Syntax Elements	91
Number Format	91
<NL> (Line Terminator)	91
[ ] (Optional Syntax Terms)	91
{ } (Braces)	91
::= (Defined As)	91
<> (Angle Brackets)	92
... (Ellipsis)	92
n,...,p (Value Ranges)	92
d (Digits)	92
Quoted ASCII String	92
Definite-Length Block Response Data	92

## 5 Common (\*) Commands

*CLS (Clear Status)	99
*ESE (Standard Event Status Enable)	100
*ESR (Standard Event Status Register)	102
*IDN (Identification Number)	104
*LRN (Learn Device Setup)	105
*OPC (Operation Complete)	106
*OPT (Option Identification)	107
*RCL (Recall)	108
*RST (Reset)	109
*SAV (Save)	112
*SRE (Service Request Enable)	113
*STB (Read Status Byte)	115
*TRG (Trigger)	117
*TST (Self Test)	118
*WAI (Wait To Continue)	119

## 6 Root (:) Commands

:ACTivity	125
:AER (Arm Event Register)	126
:AUToscale	127
:AUToscale:AMODE	129



- :AUToscale:CHANnels 130
- :AUToscale:FDEBug 131
- :BLANK 132
- :DIGitize 133
- :MTEenable (Mask Test Event Enable Register) 135
- :MTERegister[:EVENT] (Mask Test Event Event Register) 137
- :OPEE (Operation Status Enable Register) 139
- :OPERRegister:CONDition (Operation Status Condition Register) 141
- :OPERRegister[:EVENT] (Operation Status Event Register) 143
- :OVLenable (Overload Event Enable Register) 145
- :OVLRegister (Overload Event Register) 147
- :PRINT 149
- :RUN 150
- :SERial 151
- :SINGLE 152
- :STATus 153
- :STOP 154
- :TER (Trigger Event Register) 155
- :VIEW 156

## 7 :ACQUIRE Commands

- :ACQUIRE:COMPLETE 159
- :ACQUIRE:COUNT 160
- :ACQUIRE:MODE 161
- :ACQUIRE:POINTS 162
- :ACQUIRE:SEGMENTed:ANALyze 163
- :ACQUIRE:SEGMENTed:COUNT 164
- :ACQUIRE:SEGMENTed:INDEX 165
- :ACQUIRE:SRATE 168
- :ACQUIRE:TYPE 169

## 8 :BUS<n> Commands

- :BUS<n>:BIT<m> 173
- :BUS<n>:BITS 174
- :BUS<n>:CLEar 176
- :BUS<n>:DISPlay 177
- :BUS<n>:LABel 178
- :BUS<n>:MASK 179

## 9 :CALIBRATE Commands

- :CALIBRATE:DATE 183

- :CALibrate:LABel 184
- :CALibrate:OUTPut 185
- :CALibrate:PROTeCted 186
- :CALibrate:STARt 187
- :CALibrate:STATus 188
- :CALibrate:TEMPerature 189
- :CALibrate:TIME 190

## 10 :CHANnel<n> Commands

- :CHANnel<n>:BWLimit 194
- :CHANnel<n>:COUPling 195
- :CHANnel<n>:DISPlay 196
- :CHANnel<n>:IMPedance 197
- :CHANnel<n>:INVert 198
- :CHANnel<n>:LABel 199
- :CHANnel<n>:OFFSet 200
- :CHANnel<n>:PROBe 201
- :CHANnel<n>:PROBe:HEAD[:TYPE] 202
- :CHANnel<n>:PROBe:ID 203
- :CHANnel<n>:PROBe:SKEW 204
- :CHANnel<n>:PROBe:STYPe 205
- :CHANnel<n>:PROTection 206
- :CHANnel<n>:RANGe 207
- :CHANnel<n>:SCALE 208
- :CHANnel<n>:UNITs 209
- :CHANnel<n>:VERNier 210

## 11 :DEMO Commands

- :DEMO:FUNcTion 212
- :DEMO:FUNcTion:PHASe:PHASe 214
- :DEMO:OUTPut 215

## 12 :DIGital<d> Commands

- :DIGital<d>:DISPlay 219
- :DIGital<d>:LABel 220
- :DIGital<d>:POSition 221
- :DIGital<d>:SIZE 222
- :DIGital<d>:THReshold 223

## 13 :DISPlay Commands

- :DISPlay:CLEar 227

:DISPlay:DATA 228  
:DISPlay:LABel 229  
:DISPlay:LABList 230  
:DISPlay:PERsistence 231  
:DISPlay:VECTors 232

## 14 :EXtErnal Trigger Commands

:EXtErnal:BWLimit 234  
:EXtErnal:PROBe 235  
:EXtErnal:RANGe 236  
:EXtErnal:UNITs 237

## 15 :FUNction Commands

:FUNction:CENTer 242  
:FUNction:DISPlay 243  
:FUNction:GOFT:OPERation 244  
:FUNction:GOFT:SOURce1 245  
:FUNction:GOFT:SOURce2 246  
:FUNction:OFFSet 247  
:FUNction:OPERation 248  
:FUNction:RANGe 249  
:FUNction:REFerence 250  
:FUNction:SCALe 251  
:FUNction:SOURce1 252  
:FUNction:SOURce2 253  
:FUNction:SPAN 254  
:FUNction:WINDow 255

## 16 :HARDcopy Commands

:HARDcopy:AREA 259  
:HARDcopy:APRinter 260  
:HARDcopy:FACTors 261  
:HARDcopy:FFEed 262  
:HARDcopy:INKSaver 263  
:HARDcopy:LAYout 264  
:HARDcopy:NETWork:ADDRes 265  
:HARDcopy:NETWork:APPLY 266  
:HARDcopy:NETWork:DOMain 267  
:HARDcopy:NETWork:PASSword 268  
:HARDcopy:NETWork:SLOT 269  
:HARDcopy:NETWork:USERname 270

:HARDcopy:PALETTE 271  
:HARDcopy:PRINter:LIST 272  
:HARDcopy:STARt 273

## 17 :MARKer Commands

:MARKer:MODE 277  
:MARKer:X1Position 278  
:MARKer:X1Y1source 279  
:MARKer:X2Position 280  
:MARKer:X2Y2source 281  
:MARKer:XDELta 282  
:MARKer:Y1Position 283  
:MARKer:Y2Position 284  
:MARKer:YDELta 285

## 18 :MEASure Commands

:MEASure:ALL 296  
:MEASure:CLEar 297  
:MEASure:DEFine 298  
:MEASure:DELay 301  
:MEASure:DUTYcycle 303  
:MEASure:FALLtime 304  
:MEASure:FREQuency 305  
:MEASure:NWIDth 306  
:MEASure:OVERshoot 307  
:MEASure:PERiod 309  
:MEASure:PHASe 310  
:MEASure:PREShoot 311  
:MEASure:PWIDth 312  
:MEASure:RISetime 313  
:MEASure:SHOW 314  
:MEASure:SOURce 315  
:MEASure:TEDGe 317  
:MEASure:TVALue 319  
:MEASure:VAMPLitude 321  
:MEASure:VAverage 322  
:MEASure:VBASe 323  
:MEASure:VMAX 324  
:MEASure:VMIN 325  
:MEASure:VPP 326  
:MEASure:VRMS 327

:MEASure:VTIMe 328  
:MEASure:VTOp 329  
:MEASure:WINDow 330

## 19 :MTESt Commands

:MTESt:ALL 336  
:MTESt:AMASk:CREate 337  
:MTESt:AMASk:SOURce 338  
:MTESt:AMASk:UNITs 339  
:MTESt:AMASk:XDELta 340  
:MTESt:AMASk:YDELta 341  
:MTESt:COUNt:FWAVEforms 342  
:MTESt:COUNt:RESet 343  
:MTESt:COUNt:TIME 344  
:MTESt:COUNt:WAVEforms 345  
:MTESt:DATA 346  
:MTESt:DELeTe 347  
:MTESt:ENABle 348  
:MTESt:LOCK 349  
:MTESt:RMODE 350  
:MTESt:RMODE:FACTion:MEASure 351  
:MTESt:RMODE:FACTion:PRINt 352  
:MTESt:RMODE:FACTion:SAVE 353  
:MTESt:RMODE:FACTion:STOP 354  
:MTESt:RMODE:SIGMa 355  
:MTESt:RMODE:TIME 356  
:MTESt:RMODE:WAVEforms 357  
:MTESt:SCALe:BIND 358  
:MTESt:SCALe:X1 359  
:MTESt:SCALe:XDELta 360  
:MTESt:SCALe:Y1 361  
:MTESt:SCALe:Y2 362  
:MTESt:SOURce 363  
:MTESt:TITLe 364

## 20 :POD Commands

:POD<n>:DISPlay 366  
:POD<n>:SIZE 367  
:POD<n>:THReshold 368

## 21 :RECall Commands

:RECall:FIlename	373
:RECall:MASK[:START]	374
:RECall:PWD	375
:RECall:SETup[:START]	376
:RECall:WMEMory<r>[:START]	377

## 22 :SAVE Commands

:SAVE:FIlename	382
:SAVE:IMAGe[:START]	383
:SAVE:IMAGe:FACTors	384
:SAVE:IMAGe:FORMat	385
:SAVE:IMAGe:INKSaver	386
:SAVE:IMAGe:PALette	387
:SAVE:MASK[:START]	388
:SAVE:PWD	389
:SAVE:SETup[:START]	390
:SAVE:WAVeform[:START]	391
:SAVE:WAVeform:FORMat	392
:SAVE:WAVeform:LENGth	393
:SAVE:WAVeform:SEGMented	394
:SAVE:WMEMory:SOURce	395
:SAVE:WMEMory[:START]	396

## 23 :SYSTem Commands

:SYSTem:DATE	399
:SYSTem:DSP	400
:SYSTem:ERRor	401
:SYSTem:LOCK	402
:SYSTem:PRESet	403
:SYSTem:PROTection:LOCK	406
:SYSTem:SETup	407
:SYSTem:TIME	409

## 24 :TIMebase Commands

:TIMebase:MODE	413
:TIMebase:POSition	414
:TIMebase:RANGe	415
:TIMebase:REFerence	416
:TIMebase:SCALE	417
:TIMebase:VERNier	418

:TIMebase:WINDow:POSition	419
:TIMebase:WINDow:RANGe	420
:TIMebase:WINDow:SCALe	421

## 25 :TRIGger Commands

General :TRIGger Commands	425
:TRIGger:HFReject	426
:TRIGger:HOLDoff	427
:TRIGger:LEVel:HIGH	428
:TRIGger:LEVel:LOW	429
:TRIGger:MODE	430
:TRIGger:NREject	431
:TRIGger:SWEep	432
:TRIGger[:EDGE] Commands	433
:TRIGger[:EDGE]:COUpling	434
:TRIGger[:EDGE]:LEVel	435
:TRIGger[:EDGE]:REject	436
:TRIGger[:EDGE]:SLOPe	437
:TRIGger[:EDGE]:SOURce	438
:TRIGger:GLITch Commands	439
:TRIGger:GLITch:GREaterthan	441
:TRIGger:GLITch:LESSthan	442
:TRIGger:GLITch:LEVel	443
:TRIGger:GLITch:POLarity	444
:TRIGger:GLITch:QUALifier	445
:TRIGger:GLITch:RANGe	446
:TRIGger:GLITch:SOURce	447
:TRIGger:PATTern Commands	448
:TRIGger:PATTern	449
:TRIGger:PATTern:FORMat	451
:TRIGger:PATTern:QUALifier	452
:TRIGger:TV Commands	453
:TRIGger:TV:LINE	454
:TRIGger:TV:MODE	455
:TRIGger:TV:POLarity	456
:TRIGger:TV:SOURce	457
:TRIGger:TV:STANdard	458

## 26 :WAVEform Commands

:WAVEform:BYTeorder	467
---------------------	-----

- :WAVeform:COUNT 468
- :WAVeform:DATA 469
- :WAVeform:FORMat 471
- :WAVeform:POINts 472
- :WAVeform:POINts:MODE 474
- :WAVeform:PREamble 476
- :WAVeform:SEGmented:COUNT 479
- :WAVeform:SEGmented:TTAG 480
- :WAVeform:SOURce 481
- :WAVeform:SOURce:SUBSource 485
- :WAVeform:TYPE 486
- :WAVeform:UNSigned 487
- :WAVeform:VIEW 488
- :WAVeform:XINCrement 489
- :WAVeform:XORigin 490
- :WAVeform:XREFerence 491
- :WAVeform:YINCrement 492
- :WAVeform:YORigin 493
- :WAVeform:YREFerence 494

## 27 :WGEN Commands

- :WGEN:FREQuency 497
- :WGEN:FUNcTion 498
- :WGEN:FUNcTion:PULSe:WIDTh 500
- :WGEN:FUNcTion:RAMP:SYMMetry 501
- :WGEN:FUNcTion:SQUare:DCYCLE 502
- :WGEN:OUTPut 503
- :WGEN:OUTPut:LOAD 504
- :WGEN:PERiod 505
- :WGEN:RST 506
- :WGEN:VOLTage 507
- :WGEN:VOLTage:HIGH 508
- :WGEN:VOLTage:LOW 509
- :WGEN:VOLTage:OFFSet 510

## 28 :WMEMory<r> Commands

- :WMEMory<r>:CLEar 513
- :WMEMory<r>:DISPlay 514
- :WMEMory<r>:LABel 515
- :WMEMory<r>:SAVE 516
- :WMEMory<r>:SKEW 517



:WMEMory<r>:YOFFset 518  
:WMEMory<r>:YRANge 519  
:WMEMory<r>:YSCale 520

## 29 Obsolete and Discontinued Commands

:CHANnel:ACTivity 526  
:CHANnel:LABel 527  
:CHANnel:THReshold 528  
:CHANnel2:SKEW 529  
:CHANnel<n>:INPut 530  
:CHANnel<n>:PMODE 531  
:DISPlay:CONNect 532  
:DISPlay:ORDer 533  
:ERASe 534  
:EXTernal:PMODE 535  
:FUNction:SOURce 536  
:FUNction:VIEW 537  
:HARDcopy:DESTination 538  
:HARDcopy:FILename 539  
:HARDcopy:GRAYscale 540  
:HARDcopy:IGColors 541  
:HARDcopy:PDRiver 542  
:MEASure:LOWer 543  
:MEASure:SCRatch 544  
:MEASure:TDELta 545  
:MEASure:THResholds 546  
:MEASure:TSTArt 547  
:MEASure:TSTOp 548  
:MEASure:TVOLt 549  
:MEASure:UPPer 551  
:MEASure:VDELta 552  
:MEASure:VSTArt 553  
:MEASure:VSTOp 554  
:MTESt:AMASK:{SAVE | STORe} 555  
:MTESt:AVERAge 556  
:MTESt:AVERAge:COUNT 557  
:MTESt:LOAD 558  
:MTESt:RUMode 559  
:MTESt:RUMode:SOFailure 560  
:MTESt:{STARt | STOp} 561  
:MTESt:TRIGger:SOURce 562

:PRINt? 563  
:SAVE:IMAGe:AREA 565  
:TIMebase:DELay 566  
:TRIGger:THReshold 567  
:TRIGger:TV:TVMode 568

## 30 Error Messages

## 31 Status Reporting

Status Reporting Data Structures 579  
Status Byte Register (STB) 581  
Service Request Enable Register (SRE) 583  
Trigger Event Register (TER) 584  
Output Queue 585  
Message Queue 586  
(Standard) Event Status Register (ESR) 587  
(Standard) Event Status Enable Register (ESE) 588  
Error Queue 589  
Operation Status Event Register (:OPERegister[:EVENT]) 590  
Operation Status Condition Register (:OPERegister:CONDition) 591  
Arm Event Register (AER) 592  
Overload Event Register (:OVLRegister) 593  
Mask Test Event Event Register (:MTERegister[:EVENT]) 594  
Clearing Registers and Queues 595  
Status Reporting Decision Chart 596

## 32 Synchronizing Acquisitions

Synchronization in the Programming Flow 598  
    Set Up the Oscilloscope 598  
    Acquire a Waveform 598  
    Retrieve Results 598  
Blocking Synchronization 599  
Polling Synchronization With Timeout 600  
Synchronizing with a Single-Shot Device Under Test (DUT) 602  
Synchronization with an Averaging Acquisition 604

### 33 More About Oscilloscope Commands

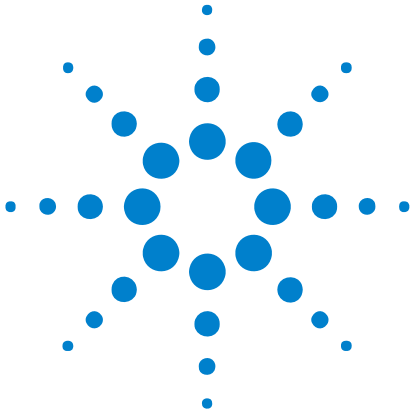
Command Classifications	608
Core Commands	608
Non-Core Commands	608
Obsolete Commands	608
Valid Command/Query Strings	609
Program Message Syntax	609
Duplicate Mnemonics	613
Tree Traversal Rules and Multiple Commands	613
Query Return Values	615
All Oscilloscope Commands Are Sequential	616

### 34 Programming Examples

VISA COM Examples	618
VISA COM Example in Visual Basic	618
VISA COM Example in C#	627
VISA COM Example in Visual Basic .NET	636
VISA Examples	645
VISA Example in C	645
VISA Example in Visual Basic	654
VISA Example in C#	664
VISA Example in Visual Basic .NET	675
SICL Examples	686
SICL Example in C	686
SICL Example in Visual Basic	695

## Index





# 1

## What's New

What's New in Version 1.10 [22](#)

Version 1.00 at Introduction [23](#)

Command Differences From 7000B Series Oscilloscopes [24](#)



## What's New in Version 1.10

New command descriptions for Version 1.10 of the InfiniiVision 2000 X-Series oscilloscope software appear below.

### New Commands

Command	Description
:SYSTem:PRESet (see <a href="#">page 403</a> )	Now documented, this command is equivalent to the front panel <b>[Default Setup]</b> key which leaves some user settings, like preferences, unchanged. The *RST command is equivalent to a factory default setup where no user settings are left unchanged.

## Version 1.00 at Introduction

The Agilent InfiniiVision 2000 X-Series oscilloscopes were introduced with version 1.00 of oscilloscope operating software.

The command set is most closely related to the InfiniiVision 7000B Series oscilloscopes (and the 7000A Series, 6000 Series, and 54620/54640 Series oscilloscopes before them). For more information, see [“Command Differences From 7000B Series Oscilloscopes”](#) on page 24.

## Command Differences From 7000B Series Oscilloscopes

The Agilent InfiniiVision 2000 X-Series oscilloscopes command set is most closely related to the InfiniiVision 7000B Series oscilloscopes (and the 7000A Series, 6000 Series, and 54620/54640 Series oscilloscopes before them).

The main differences between the version 1.00 programming command set for the InfiniiVision 2000 X-Series oscilloscopes and the 6.10 programming command set for the InfiniiVision 7000B Series oscilloscopes are related to:

- Built-in waveform generator (with Option WGN license).
- Built-in demo signals (with Option EDU license that comes with the N6455A Education Kit).
- Reference waveforms (in place of trace memory).
- Serial decode is not supported.
- Waveform event search is not supported.
- Smaller set of trigger types.
- Fewer measurements.
- Different path name format for internal and USB storage device locations.

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

### New Commands

Command	Description
:DEMO Commands (see <a href="#">page 211</a> )	Commands for using built-in demo signals (with the Option EDU license that comes with the N6455A Education Kit).
:HARDcopy:NETWork Commands (see <a href="#">page 257</a> )	For accessing network printers.
:MEASure:WINDow (see <a href="#">page 330</a> )	When the zoomed time base is on, specifies whether the measurement window is the zoomed time base or the main time base.
:MTESt:ALL (see <a href="#">page 336</a> )	Specifies whether all channels are included in the mask test.
:RECall:WMEMory<r>[:START] (see <a href="#">page 377</a> )	Recalls reference waveforms.
:SAVE:WMEMory:SOURce (see <a href="#">page 395</a> )	Selects the source for saving a reference waveform.
:SAVE:WMEMory[:START] (see <a href="#">page 396</a> )	Saves reference waveforms.



Command	Description
:TRIGger:LEVel:HIGH (see <a href="#">page 428</a> )	Sets runt and transition (rise/fall time) trigger high level.
:TRIGger:LEVel:LOW (see <a href="#">page 429</a> )	Sets runt and transition (rise/fall time) trigger low level.
:TRIGger:PATtern Commands (see <a href="#">page 448</a> )	This subsystem contains commands/functions that are in the 7000B Series oscilloscope's :TRIGger:DURation subsystem.
:WGEN Commands (see <a href="#">page 495</a> )	Commands for controlling the built-in waveform generator (with Option WGN license).
:WMEMory<r> Commands (see <a href="#">page 511</a> )	Commands for reference waveforms.

**Changed Commands**

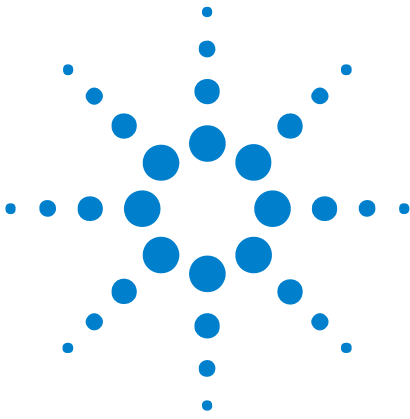
Command	Differences From InfiniiVision 7000B Series Oscilloscopes
:ACQuire:MODE (see <a href="#">page 161</a> )	There is no ETIME parameter with the 2000 X-Series oscilloscopes.
:CALibrate:OUTPut (see <a href="#">page 185</a> )	The TRIG OUT signal can be a trigger output, mask test failure, or waveform generator sync pulse.
:DISPlay:DATA (see <a href="#">page 228</a> )	Monochrome TIFF images of the graticule cannot be saved or restored.
:DISPlay:LABList (see <a href="#">page 230</a> )	The label list contains up to 77, 10-character labels (instead of 75).
:DISPlay:VECTors (see <a href="#">page 232</a> )	Always ON with 2000 X-Series oscilloscopes.
:MARKer Commands (see <a href="#">page 275</a> )	Can select reference waveforms as marker source.
:MEASure Commands (see <a href="#">page 287</a> )	Can select reference waveforms as the source for many measurements.
:SAVE:IMAGe[:START] (see <a href="#">page 383</a> )	Cannot save images to internal locations.
:TRIGger:PATtern (see <a href="#">page 449</a> )	Takes <string> parameter instead of <value>,<mask> parameters.
:WAVEform:SOURce (see <a href="#">page 481</a> )	Can select reference waveforms as the source.
:VIEW (see <a href="#">page 156</a> )	PMEMory (pixel memory) locations are not present.

**Obsolete Commands**

Obsolete Command	Current Command Equivalent	Behavior Differences

**Discontinued  
Commands**

Command	Description
:ACQuire:RSIGnal	The 2000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.
:CALibrate:SWITch?	Replaced by :CALibrate:PROTected? (see <a href="#">page 186</a> ). The oscilloscope has a protection button instead of a switch.
:DISPlay:SOURce	PMEMory (pixel memory) locations are not present.
:EXTernal:IMPedance	External TRIG IN connector is now fixed at 1 MOhm.
:EXTernal:PROBe:ID	Not supported on external TRIG IN connector.
:EXTernal:PROBe:STYPe	Not supported on external TRIG IN connector.
:EXTernal:PROTectiOn	Not supported on external TRIG IN connector.
:HARDcopy:DEVice, :HARDcopy:FORMat	Use the :SAVE:IMAGe:FORMat, :SAVE:WAVEform:FORMat, and :HARDcopy:APRinter commands instead.
:MERGe	Waveform traces have been replaced by reference waveforms.
:RECall:IMAGe[:START]	Waveform traces have been replaced by reference waveforms.
:SYSTem:PRECision	The 2000 X-Series oscilloscopes' measurement record, and maximum record size, is 62,500 points, and there is no need for a special precision mode.
:TIMebase:REFClock	The 2000 X-Series oscilloscope does not have a 10 MHz REF BNC connector.



## 2 Setting Up

- Step 1. Install Agilent IO Libraries Suite software [28](#)
- Step 2. Connect and set up the oscilloscope [29](#)
- Step 3. Verify the oscilloscope connection [31](#)

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



## Step 1. Install Agilent IO Libraries Suite software

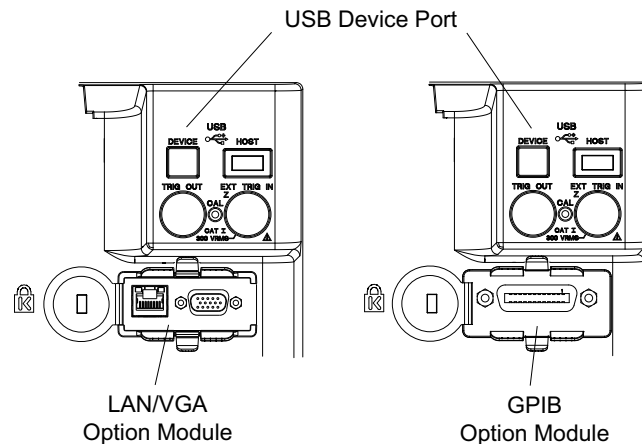
- 1 Download the Agilent IO Libraries Suite software from the Agilent web site at:
  - "<http://www.agilent.com/find/iolib>"
- 2 Run the setup file, and follow its installation instructions.

## Step 2. Connect and set up the oscilloscope

The 2000 X-Series oscilloscope has three different interfaces you can use for programming:

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

When installed, these interfaces are always active.



**Figure 1** Control Connectors on Rear Panel

### Using the USB (Device) Interface

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

This is a USB 2.0 high-speed port.

### Using the LAN Interface

- 1 If the controller PC is not already connected to the local area network (LAN), do that first.
- 2 Contact your network administrator about adding the oscilloscope to the network.

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

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

- 3 Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the LAN/VGA option module.
- 4 Configure the oscilloscope's LAN interface:
  - a Press the **Configure** softkey until "LAN" is selected.
  - b Press the **LAN Settings** softkey.
  - c Press the **Config** softkey, and enable all the configuration options supported by your network.
  - d If automatic configuration is not supported, press the **Addresses** softkey.

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

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

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

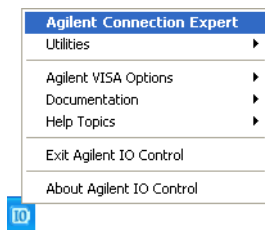
When you are done, press the **[Back up]** key.

### Using the GPIB Interface

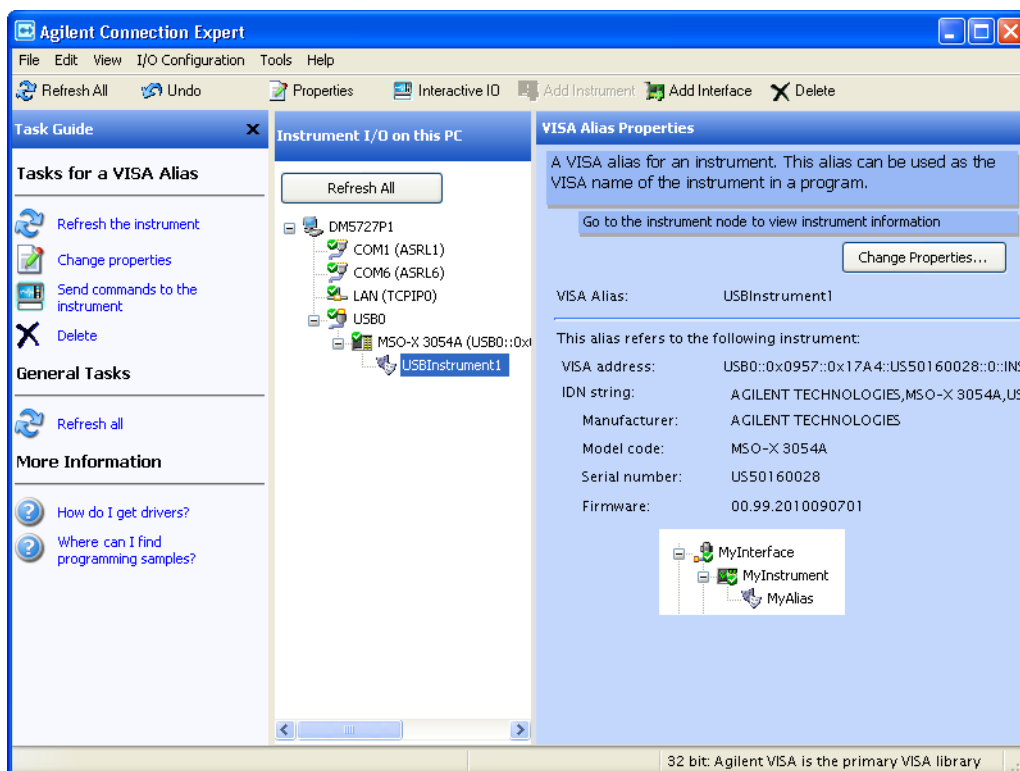
- 1 Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the GPIB option module.
- 2 Configure the oscilloscope's GPIB interface:
  - a Press the **Configure** softkey until "GPIB" is selected.
  - b Use the Entry knob to select the **Address** value.

## Step 3. Verify the oscilloscope connection

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



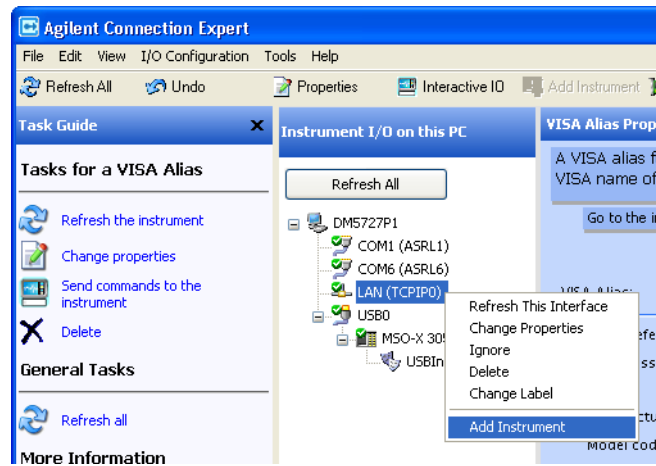
- 2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



## 2 Setting Up

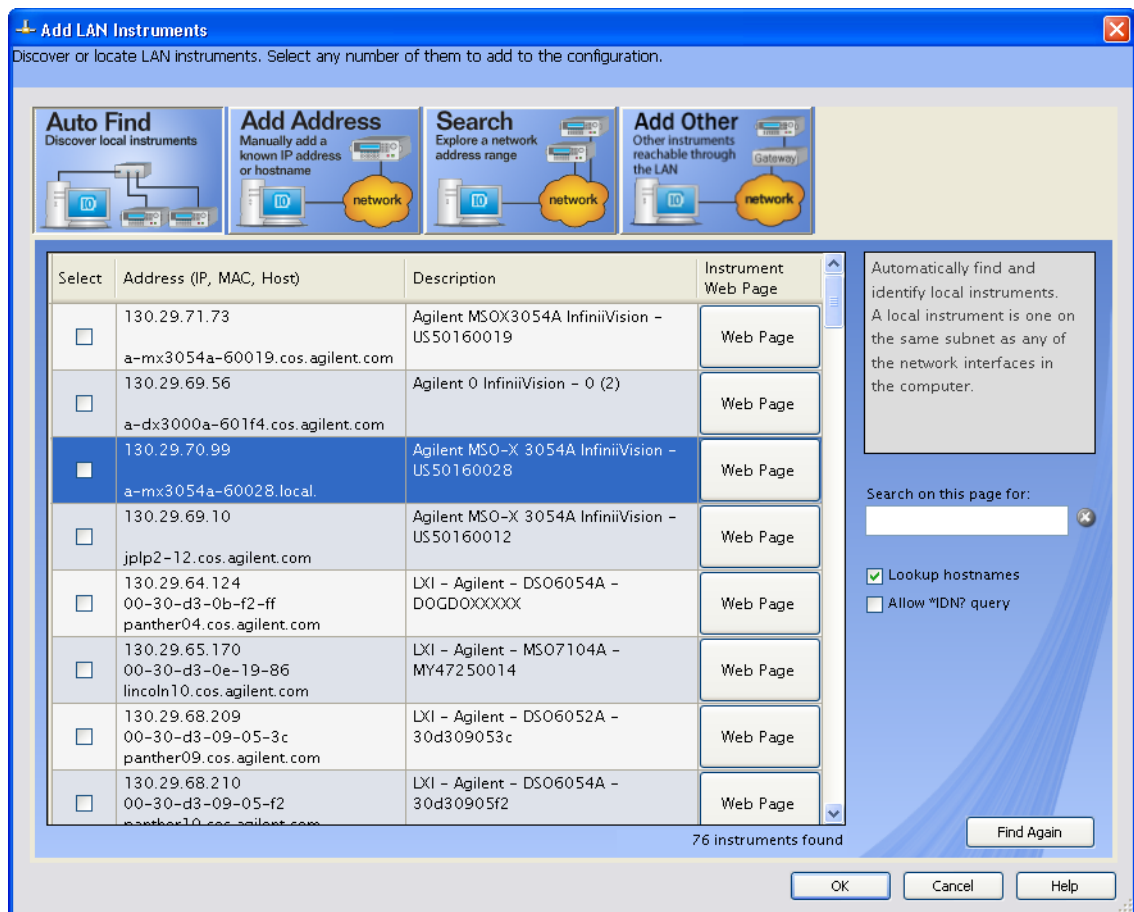
You must manually add instruments on LAN interfaces:

- 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**.

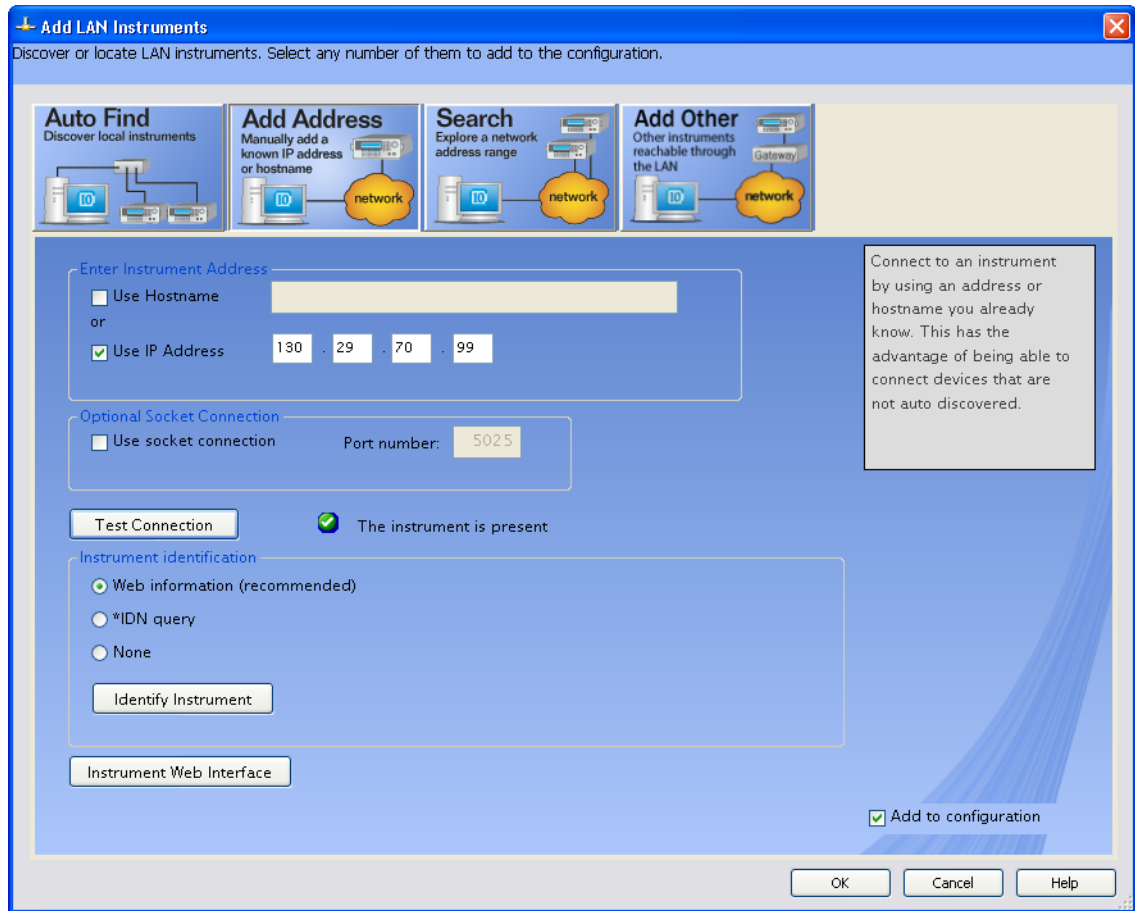




Otherwise, if the instrument is not on the same subnet, click **Add Address**.

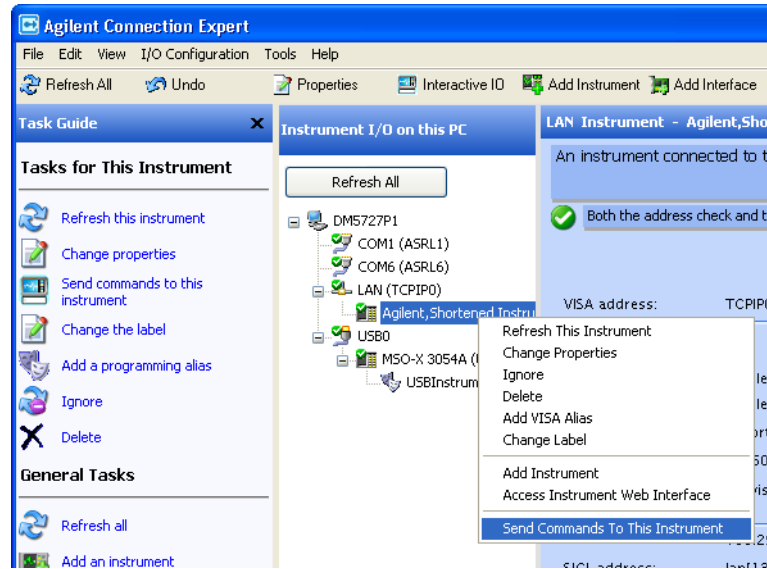
- i In the next dialog, select either **Hostname** or **IP address**, and enter the oscilloscope's hostname or IP address.
- ii Click **Test Connection**.

## 2 Setting Up

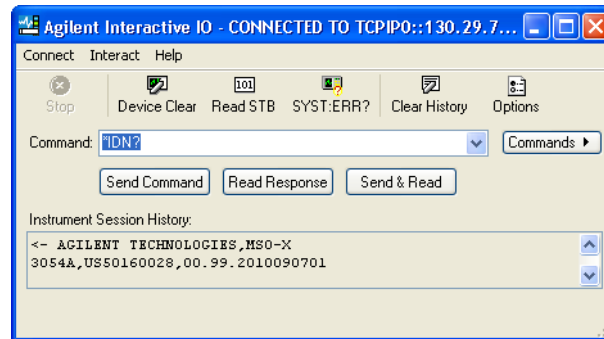


- iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

- 3 Test some commands on the instrument:
  - a Right-click on the instrument and choose **Send Commands To This Instrument** from the popup menu.

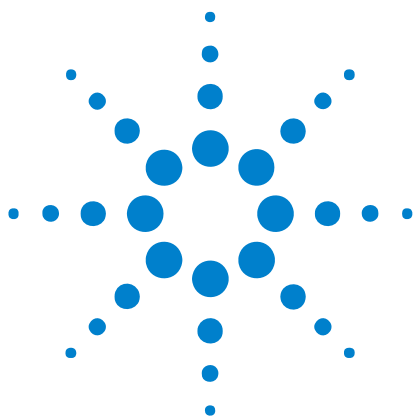


- b In the Agilent Interactive IO application, enter commands in the **Command** field and press **Send Command**, **Read Response**, or **Send&Read**.



- 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	38
Programming the Oscilloscope	40
Other Ways of Sending Commands	49

This chapter gives you an overview of programming the 2000 X-Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

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

### NOTE

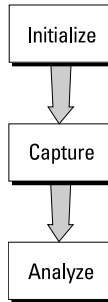
#### Language for Program Examples

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



## Basic Oscilloscope Program Structure

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



### Initializing

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

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

### Capturing Data

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

To collect data, you use the `:DIGitize` command. This command clears the waveform buffers and starts the acquisition process. Acquisition continues until acquisition memory is full, then stops. The acquired data is displayed by the oscilloscope, and the captured data can be measured, stored in

acquisition memory in the oscilloscope, or transferred to the controller for further analysis. Any additional commands sent while :DIGitize is working are buffered until :DIGitize is complete.

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

## Analyzing Captured Data

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

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

## Programming the Oscilloscope

- "Referencing the IO Library" on page 40
- "Opening the Oscilloscope Connection via the IO Library" on page 41
- "Using :AUToscale to Automate Oscilloscope Setup" on page 42
- "Using Other Oscilloscope Setup Commands" on page 42
- "Capturing Data with the :DIGitize Command" on page 43
- "Reading Query Responses from the Oscilloscope" on page 45
- "Reading Query Results into String Variables" on page 46
- "Reading Query Results into Numeric Variables" on page 46
- "Reading Definite-Length Block Query Response Data" on page 46
- "Sending Multiple Queries and Reading Results" on page 47
- "Checking Instrument Status" on page 48

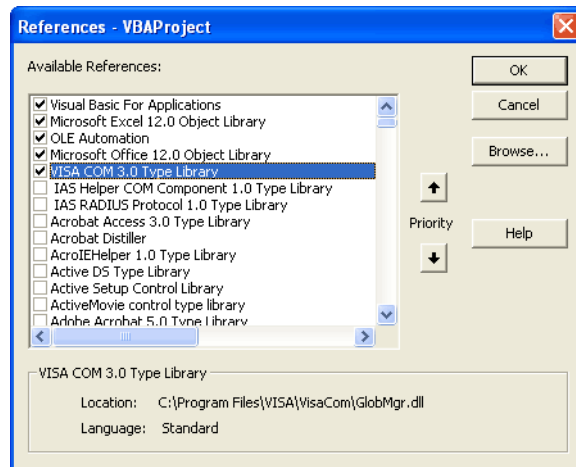
## Referencing the IO Library

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 programming 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 609.

## Initializing the Interface and the Oscilloscope

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. When using the Agilent VISA COM library, you can use the resource session object's Clear method to clear the interface buffer:

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

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

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

' Clear the interface buffer and set the interface timeout to 10 seconds
.
myScope.IO.Clear
myScope.IO.Timeout = 10000
```

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

After clearing the interface, initialize the instrument to a preset state:

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

#### NOTE

#### Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in [Chapter 5](#), "Common (\*) Commands," starting on page 95.

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
myScope.IO.Timeout = 10000    ' Set interface timeout to 10 seconds.

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

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

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

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

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

## Capturing Data with the :DIGitize Command

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

#### NOTE

#### Ensure New Data is Collected

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

---

When you send the :DIGitize command to the oscilloscope, the specified channel signal is digitized with the current :ACQUIRE parameters. To obtain waveform data, you must specify the :WAVEFORM parameters for the SOURCE channel, the FORMAT type, and the number of POINTs prior to sending the :WAVEFORM:DATA? query.

#### NOTE

#### Set :TIMEbase:MODE to MAIN when using :DIGitize

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

---

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

The following program example shows a typical setup:

```
myScope.WriteString ":ACQUIRE:TYPE AVERAGE"  
myScope.WriteString ":ACQUIRE:COMPLETE 100"  
myScope.WriteString ":ACQUIRE:COUNT 8"  
myScope.WriteString ":DIGITIZE CHANNEL1"  
myScope.WriteString ":WAVEFORM:SOURCE CHANNEL1"  
myScope.WriteString ":WAVEFORM:FORMAT BYTE"  
myScope.WriteString ":WAVEFORM:POINTS 500"  
myScope.WriteString ":WAVEFORM:DATA?"
```

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

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

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

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

For more information, see the waveform subsystem commands and corresponding program code examples in [Chapter 26](#), “:WAVeform Commands,” starting on page 459.

**NOTE****Aborting a Digitize Operation Over the Programming Interface**

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

**Reading Query Responses from the Oscilloscope**

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

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

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

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

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

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

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

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

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

## Reading Query Results into String Variables

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

### NOTE

#### Express String Variables Using Exact Syntax

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

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

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

After running this program, the controller displays:

**Range (string): +40.0E+00**

## Reading Query Results into Numeric Variables

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

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

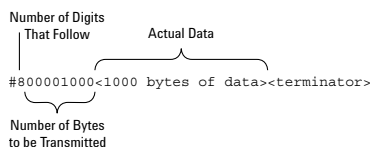
After running this program, the controller displays:

**Range (variant): 40**

## Reading Definite-Length Block Query Response Data

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

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



**Figure 2** Definite-length block response data

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

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

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

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

## Sending Multiple Queries and Reading Results

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

For example, to read the `:TIMEbase:RANGE?;DElay?` query result into a single string variable, you could use the commands:

```
myScope.WriteString ":TIMEbase:RANGE?;DElay?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
MsgBox "Timebase range; delay:" + strQueryResult
```

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

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

To read the `:TIMEbase:RANGE?;DElay?` query result into multiple string variables, you could use the `ReadList` method to read the query results into a string array variable using the commands:

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

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

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

## Checking Instrument Status

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

For more information, see [Chapter 31](#), “Status Reporting,” starting on page 577 which explains how to check the status of the instrument.



## Other Ways of Sending Commands

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

- "Telnet Sockets" on page 49
- "Sending SCPI Commands Using Browser Web Control" on page 49

### Telnet Sockets

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

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

```
telnet <hostname> 5024
```

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

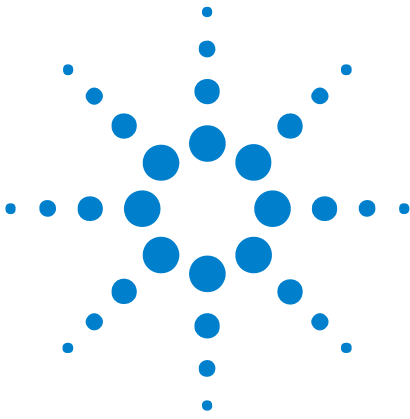
For a command line without a prompt, use port 5025. For example:

```
telnet <hostname> 5025
```

### Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the *InfiniiVision 2000 X-Series Oscilloscopes User's Guide*. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the **Browser Web Control** tab, then select the **Remote Programming** link.

### **3 Getting Started**



## 4 Commands Quick Reference

Command Summary 52

Syntax Elements 91



### Command Summary

- Common (\*) Commands Summary (see [page 53](#))
- Root (:) Commands Summary (see [page 55](#))
- :ACQuire Commands Summary (see [page 58](#))
- :BUS<n> Commands Summary (see [page 59](#))
- :CALibrate Commands Summary (see [page 60](#))
- :CHANnel<n> Commands Summary (see [page 61](#))
- :DEMO Commands Summary (see [page 63](#))
- :DIGital<n> Commands Summary (see [page 63](#))
- :DISPlay Commands Summary (see [page 64](#))
- :EXTErnal Trigger Commands Summary (see [page 64](#))
- :FUNCTion Commands Summary (see [page 65](#))
- :HARDcopy Commands Summary (see [page 66](#))
- :MARKer Commands Summary (see [page 68](#))
- :MEASure Commands Summary (see [page 69](#))
- :MTEST Commands Summary (see [page 76](#))
- :POD<n> Commands Summary (see [page 78](#))
- :RECall Commands Summary (see [page 78](#))
- :SAVE Commands Summary (see [page 79](#))
- :SYSTEM Commands Summary (see [page 81](#))
- :TIMEbase Commands Summary (see [page 81](#))
- General :TRIGger Commands Summary (see [page 82](#))
- :TRIGger[:EDGE] Commands Summary (see [page 83](#))
- :TRIGger:GLITCh Commands Summary (see [page 84](#))
- :TRIGger:PATTern Commands Summary (see [page 85](#))
- :TRIGger:TV Commands Summary (see [page 86](#))
- :WAVEform Commands Summary (see [page 86](#))
- :WGEN Commands Summary (see [page 89](#))
- :WMEMory<r> Commands Summary (see [page 90](#))

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

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 99</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 100</a> )	*ESE? (see <a href="#">page 100</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 102</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 102</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 105</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 106</a> )	*OPC? (see <a href="#">page 106</a> )	ASCII "1" is placed in the output queue when all pending device operations have completed.

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

Command	Query	Options and Query Returns
n/a	*OPT? (see <a href="#">page 107</a> )	<pre>&lt;return_value&gt; ::= 0,0,&lt;license info&gt; &lt;license info&gt; ::= &lt;All field&gt;, &lt;reserved&gt;, &lt;MSO&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;Power Measurements&gt;, &lt;reserved&gt;, &lt;Segmented Memory&gt;, &lt;Mask Test&gt;, &lt;reserved&gt;, &lt;Bandwidth&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;Waveform Generator&gt;, &lt;reserved&gt;, &lt;reserved&gt; &lt;All field&gt; ::= {0   All} &lt;reserved&gt; ::= 0 &lt;MSO&gt; ::= {0   MSO} &lt;Power Measurements&gt; ::= {0   PWR} &lt;Segmented Memory&gt; ::= {0   SGM} &lt;Mask Test&gt; ::= {0   MASK} &lt;Bandwidth&gt; ::= {0   BW10   BW20} &lt;Waveform Generator&gt; ::= {0   WAVEGEN}</pre>
*RCL <value> (see <a href="#">page 108</a> )	n/a	<value> ::= {0   1   4   5   6   7   8   9}
*RST (see <a href="#">page 109</a> )	n/a	See *RST (Reset) (see <a href="#">page 109</a> )
*SAV <value> (see <a href="#">page 112</a> )	n/a	<value> ::= {0   1   4   5   6   7   8   9}
*SRE <mask> (see <a href="#">page 113</a> )	*SRE? (see <a href="#">page 114</a> )	<pre>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= 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</pre>

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

Command	Query	Options and Query Returns
n/a	*STB? (see <a href="#">page 115</a> )	<p>&lt;value&gt; ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <p>Bit Weight Name "1" Indicates</p> <pre> ----- 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.           </pre>
*TRG (see <a href="#">page 117</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 118</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 119</a> )	n/a	n/a

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

Command	Query	Options and Query Returns
:ACTivity (see <a href="#">page 125</a> )	:ACTivity? (see <a href="#">page 125</a> )	<p>&lt;return value&gt; ::= &lt;edges&gt;,&lt;levels&gt;</p> <p>&lt;edges&gt; ::= presence of edges (32-bit integer in NR1 format)</p> <p>&lt;levels&gt; ::= logical highs or lows (32-bit integer in NR1 format)</p>
n/a	:AER? (see <a href="#">page 126</a> )	{0   1}; an integer in NR1 format

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:AUToscale [<source>[,...,<source>]] (see <a href="#">page 127</a> )	n/a	<source> ::= CHANNEL<n> for DSO models <source> ::= {CHANNEL<n>   DIGital<d>   POD1   POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODE <value> (see <a href="#">page 129</a> )	:AUToscale:AMODE? (see <a href="#">page 129</a> )	<value> ::= {NORMAL   CURRENT}}
:AUToscale:CHANnels <value> (see <a href="#">page 130</a> )	:AUToscale:CHANnels? (see <a href="#">page 130</a> )	<value> ::= {ALL   DISPLAYed}}
:AUToscale:FDEBug {{0   OFF}   {1   ON}} (see <a href="#">page 131</a> )	:AUToscale:FDEBug? (see <a href="#">page 131</a> )	{0   1}
:BLANK [<source>] (see <a href="#">page 132</a> )	n/a	<source> ::= {CHANNEL<n>}   FUNCTION   MATH} for DSO models <source> ::= {CHANNEL<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNCTION   MATH} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:DIGitize [<source>[,...,<source>]] (see <a href="#">page 133</a> )	n/a	<source> ::= {CHANNEL<n>   FUNCTION   MATH} for DSO models <source> ::= {CHANNEL<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNCTION   MATH} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:MTEenable <n> (see <a href="#">page 135</a> )	:MTEenable? (see <a href="#">page 135</a> )	<n> ::= 16-bit integer in NR1 format



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

Command	Query	Options and Query Returns
n/a	:MTERegister[:EVENT]? (see <a href="#">page 137</a> )	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see <a href="#">page 139</a> )	:OPEE? (see <a href="#">page 139</a> )	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDition? (see <a href="#">page 141</a> )	<n> ::= 15-bit integer in NR1 format
n/a	:OPERRegister[:EVENT]? (see <a href="#">page 143</a> )	<n> ::= 15-bit integer in NR1 format
:OVLenable <mask> (see <a href="#">page 145</a> )	:OVLenable? (see <a href="#">page 146</a> )	<mask> ::= 16-bit integer in NR1 format as shown:  Bit Weight Input --- ---- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see <a href="#">page 147</a> )	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINT [<options>] (see <a href="#">page 149</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 150</a> )	n/a	n/a
n/a	:SERial (see <a href="#">page 151</a> )	<return value> ::= unquoted string containing serial number
:SINGle (see <a href="#">page 152</a> )	n/a	n/a

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
n/a	:STATus? <display> (see <a href="#">page 153</a> )	{0   1} <display> ::= {CHANnel<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNCTION   MATH} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:STOP (see <a href="#">page 154</a> )	n/a	n/a
n/a	:TER? (see <a href="#">page 155</a> )	{0   1}
:VIEW <source> (see <a href="#">page 156</a> )	n/a	<source> ::= {CHANnel<n>   FUNCTION   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNCTION   MATH} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

**Table 4** :ACQUIRE Commands Summary

Command	Query	Options and Query Returns
:ACQUIRE:COMPLETE <complete> (see <a href="#">page 159</a> )	:ACQUIRE:COMPLETE? (see <a href="#">page 159</a> )	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNT <count> (see <a href="#">page 160</a> )	:ACQUIRE:COUNT? (see <a href="#">page 160</a> )	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQUIRE:MODE <mode> (see <a href="#">page 161</a> )	:ACQUIRE:MODE? (see <a href="#">page 161</a> )	<mode> ::= {RTIME   SEGmented}
n/a	:ACQUIRE:POINTS? (see <a href="#">page 162</a> )	<# points> ::= an integer in NR1 format
:ACQUIRE:SEGmented:ANALyze (see <a href="#">page 163</a> )	n/a	n/a (with Option SGM)
:ACQUIRE:SEGmented:COUNT <count> (see <a href="#">page 164</a> )	:ACQUIRE:SEGmented:COUNT? (see <a href="#">page 164</a> )	<count> ::= an integer from 2 to 25 in NR1 format (with Option SGM)

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

Command	Query	Options and Query Returns
:ACQUIRE:SEGMENTED:IN Dex <index> (see page 165)	:ACQUIRE:SEGMENTED:IN Dex? (see page 165)	<index> ::= an integer from 1 to 25 in NR1 format (with Option SGM)
n/a	:ACQUIRE:SRATE? (see page 168)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQUIRE:TYPE <type> (see page 169)	:ACQUIRE:TYPE? (see page 169)	<type> ::= {NORMAL   AVERAGE   HRESOLUTION   PEAK}

**Table 5** :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0   OFF}   {1   ON}} (see page 173)	:BUS<n>:BIT<m>? (see page 173)	{0   1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0   OFF}   {1   ON}} (see page 174)	:BUS<n>:BITS? (see page 174)	<channel_list>, {0   1} <channel_list> ::= (@<m>, <m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:CLEAR (see page 176)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPLAY {{0   OFF}   {1   ON}} (see page 177)	:BUS<n>:DISPLAY? (see page 177)	{0   1} <n> ::= 1 or 2; an integer in NR1 format

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:BUS<n>:LAbel <string> (see page 178)	:BUS<n>:LAbel? (see page 178)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 179)	:BUS<n>:MASK? (see page 179)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

**Table 6** :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 183)	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LAbel <string> (see page 184)	:CALibrate:LAbel? (see page 184)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see page 185)	:CALibrate:OUTPut? (see page 185)	<signal> ::= {TRIGgers   MASK   WAVEgen}
n/a	:CALibrate:PROTected? (see page 186)	{PROTected   UNPROTected}
:CALibrate:START (see page 187)	n/a	n/a
n/a	:CALibrate:STATus? (see page 188)	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string

**Table 6** :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPerature? (see <a href="#">page 189</a> )	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see <a href="#">page 190</a> )	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

**Table 7** :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {0   OFF}   {1   ON} (see <a href="#">page 194</a> )	:CHANnel<n>:BWLimit? (see <a href="#">page 194</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUpling <coupling> (see <a href="#">page 195</a> )	:CHANnel<n>:COUpling? (see <a href="#">page 195</a> )	<coupling> ::= {AC   DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay {0   OFF}   {1   ON} (see <a href="#">page 196</a> )	:CHANnel<n>:DISPlay? (see <a href="#">page 196</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see <a href="#">page 197</a> )	:CHANnel<n>:IMPedance? (see <a href="#">page 197</a> )	<impedance> ::= ONEMeg <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert {0   OFF}   {1   ON} (see <a href="#">page 198</a> )	:CHANnel<n>:INVert? (see <a href="#">page 198</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see <a href="#">page 199</a> )	:CHANnel<n>:LABel? (see <a href="#">page 199</a> )	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see <a href="#">page 200</a> )	:CHANnel<n>:OFFSet? (see <a href="#">page 200</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 201</a> )	:CHANnel<n>:PROBe? (see <a href="#">page 201</a> )	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see <a href="#">page 202</a> )	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see <a href="#">page 202</a> )	<head_param> ::= {SEND0   SEND6   SEND12   SEND20   DIFF0   DIFF6   DIFF12   DIFF20   NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see <a href="#">page 203</a> )	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see <a href="#">page 204</a> )	:CHANnel<n>:PROBe:SKEW? (see <a href="#">page 204</a> )	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see <a href="#">page 205</a> )	:CHANnel<n>:PROBe:STYPe? (see <a href="#">page 205</a> )	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see <a href="#">page 206</a> )	:CHANnel<n>:PROTection? (see <a href="#">page 206</a> )	NORM <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see <a href="#">page 207</a> )	:CHANnel<n>:RANGe? (see <a href="#">page 207</a> )	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALE <scale>[suffix] (see <a href="#">page 208</a> )	:CHANnel<n>:SCALE? (see <a href="#">page 208</a> )	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITs <units> (see <a href="#">page 209</a> )	:CHANnel<n>:UNITs? (see <a href="#">page 209</a> )	<units> ::= {VOLT   AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 210</a> )	:CHANnel<n>:VERNier? (see <a href="#">page 210</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format

**Table 8** :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCTION <signal> (see page 212)	:DEMO:FUNCTION? (see page 213)	<signal> ::= {SINusoid   NOISy   PHASe   RINGing   SINGle   AM   CLK   GLITch   BURSt   MSO   RFBurst   LFSine}
:DEMO:FUNCTION:PHASe: PHASe <angle> (see page 214)	:DEMO:FUNCTION:PHASe: PHASe? (see page 214)	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0   OFF}   {1   ON}} (see page 215)	:DEMO:OUTPut? (see page 215)	{0   1}

**Table 9** :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital<d>:DISPlay {{0   OFF}   {1   ON}} (see page 219)	:DIGital<d>:DISPlay? (see page 219)	<d> ::= 0 to (# digital channels - 1) in NR1 format {0   1}
:DIGital<d>:LABel <string> (see page 220)	:DIGital<d>:LABel? (see page 220)	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGital<d>:POSition <position> (see page 221)	:DIGital<d>:POSition? (see page 221)	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.
:DIGital<d>:SIZE <value> (see page 222)	:DIGital<d>:SIZE? (see page 222)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl   MEDium   LARGE}
:DIGital<d>:THReshold <value>[suffix] (see page 223)	:DIGital<d>:THReshold ? (see page 223)	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V   mV   uV}

## 4 Commands Quick Reference

**Table 10** :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLEar (see <a href="#">page 227</a> )	n/a	n/a
n/a	:DISPlay:DATA? [ <format>] [,][&lt;palette&gt;] (see <a href="#">page 228</a>)</format>	<format> ::= {BMP   BMP8bit   PNG} <palette> ::= {COLor   GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0   OFF}   {1   ON}} (see <a href="#">page 229</a> )	:DISPlay:LABel? (see <a href="#">page 229</a> )	{0   1}
:DISPlay:LABList <binary block> (see <a href="#">page 230</a> )	:DISPlay:LABList? (see <a href="#">page 230</a> )	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see <a href="#">page 231</a> )	:DISPlay:PERsistence? (see <a href="#">page 231</a> )	<value> ::= {MINimum   INFinite   <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1   ON} (see <a href="#">page 232</a> )	:DISPlay:VECTors? (see <a href="#">page 232</a> )	1

**Table 11** :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see <a href="#">page 234</a> )	:EXTernal:BWLimit? (see <a href="#">page 234</a> )	<bwlimit> ::= {0   OFF}
:EXTernal:PROBe <attenuation> (see <a href="#">page 235</a> )	:EXTernal:PROBe? (see <a href="#">page 235</a> )	<attenuation> ::= probe attenuation ratio in NR3 format
:EXTernal:RANGe <range> [<suffix>] (see <a href="#">page 236</a> )	:EXTernal:RANGe? (see <a href="#">page 236</a> )	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXTernal:UNITs <units> (see <a href="#">page 237</a> )	:EXTernal:UNITs? (see <a href="#">page 237</a> )	<units> ::= {VOLT   AMPere}



**Table 12** :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTer <frequency> (see page 242)	:FUNCTION:CENTer? (see page 242)	<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 page 243)	:FUNCTION:DISPlay? (see page 243)	{0   1}
:FUNCTION:GOFT:OPERat ion <operation> (see page 244)	:FUNCTION:GOFT:OPERat ion? (see page 244)	<operation> ::= {ADD   SUBTract   MULTiPly}
:FUNCTION:GOFT:SOURce 1 <source> (see page 245)	:FUNCTION:GOFT:SOURce 1? (see page 245)	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 246)	:FUNCTION:GOFT:SOURce 2? (see page 246)	<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 247)	:FUNCTION:OFFSet? (see page 247)	<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 248)	:FUNCTION:OPERation? (see page 248)	<operation> ::= {ADD   SUBTract   MULTiPly   FFT}
:FUNCTION:RANGe <range> (see page 249)	:FUNCTION:RANGe? (see page 249)	<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 FFT function is 8 to 800 dBV.
:FUNCTION:REFerence <level> (see page 250)	:FUNCTION:REFerence? (see page 250)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:FUNCTION:SCALE <scale value>[<suffix>] (see page 251)	:FUNCTION:SCALE? (see page 251)	<scale value> ::= integer in NR1 format <suffix> ::= {V   dB}
:FUNCTION:SOURCE1 <source> (see page 252)	:FUNCTION:SOURCE1? (see page 252)	<source> ::= {CHANNEL<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models GOFT is only for FFT operation.
:FUNCTION:SOURCE2 <source> (see page 253)	:FUNCTION:SOURCE2? (see page 253)	<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 page 254)	:FUNCTION:SPAN? (see page 254)	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCTION:WINDOW <window> (see page 255)	:FUNCTION:WINDOW? (see page 255)	<window> ::= {RECTangular   HANNing   FLATtop   BHARRis}

**Table 13** :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 259)	:HARDcopy:AREA? (see page 259)	<area> ::= SCREEN
:HARDcopy:APRinter <active_printer> (see page 260)	:HARDcopy:APRinter? (see page 260)	<active_printer> ::= {<index>   <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0   OFF}   {1   ON}} (see page 261)	:HARDcopy:FACTors? (see page 261)	{0   1}
:HARDcopy:FFEed {{0   OFF}   {1   ON}} (see page 262)	:HARDcopy:FFEed? (see page 262)	{0   1}
:HARDcopy:INKSaver {{0   OFF}   {1   ON}} (see page 263)	:HARDcopy:INKSaver? (see page 263)	{0   1}

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

Command	Query	Options and Query Returns
:HARDcopy:LAYout <layout> (see page 264)	:HARDcopy:LAYout? (see page 264)	<layout> ::= {LANDscape   PORTRait}
:HARDcopy:NETWork:ADD Res <address> (see page 265)	:HARDcopy:NETWork:ADD Res? (see page 265)	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see page 266)	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see page 267)	:HARDcopy:NETWork:DOM ain? (see page 267)	<domain> ::= quoted ASCII string
:HARDcopy:NETWork:PAS Sword <password> (see page 268)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLO T <slot> (see page 269)	:HARDcopy:NETWork:SLO T? (see page 269)	<slot> ::= {NET0   NET1}
:HARDcopy:NETWork:USE Rname <username> (see page 270)	:HARDcopy:NETWork:USE Rname? (see page 270)	<username> ::= quoted ASCII string
:HARDcopy:PALette <palette> (see page 271)	:HARDcopy:PALette? (see page 271)	<palette> ::= {COLor   GRAYscale   NONE}
n/a	:HARDcopy:PRINter:LIS T? (see page 272)	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y   N} <name> ::= name of printer
:HARDcopy:STArt (see page 273)	n/a	n/a

**Table 14** :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see <a href="#">page 277</a> )	:MARKer:MODE? (see <a href="#">page 277</a> )	<mode> ::= {OFF   MEASurement   MANual   WAVEform}
:MARKer:X1Position <position>[suffix] (see <a href="#">page 278</a> )	:MARKer:X1Position? (see <a href="#">page 278</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 279</a> )	:MARKer:X1Y1source? (see <a href="#">page 279</a> )	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see <a href="#">page 280</a> )	:MARKer:X2Position? (see <a href="#">page 280</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 281</a> )	:MARKer:X2Y2source? (see <a href="#">page 281</a> )	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see <a href="#">page 282</a> )	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see <a href="#">page 283</a> )	:MARKer:Y1Position? (see <a href="#">page 283</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 284</a> )	:MARKer:Y2Position? (see <a href="#">page 284</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 285</a> )	<return_value> ::= Y cursors delta value in NR3 format

**Table 15** :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see page 296)	n/a	n/a
:MEASure:CLear (see page 297)	n/a	n/a
:MEASure:DEFine DELay, <delay spec> (see page 298)	:MEASure:DEFine? DELay (see page 299)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+   -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 298)	:MEASure:DEFine? THResholds (see page 299)	<threshold spec> ::= {STANdard}   {<threshold mode>,<upper>,<middle>,<lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 301)	:MEASure:DELay? [<source1>] [,<source2>] (see page 301)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 303)	:MEASure:DUTYcycle? [<source>] (see page 303)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format

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

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 304)	:MEASure:FALLtime? [<source>] (see page 304)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREquency [<source>] (see page 305)	:MEASure:FREquency? [<source>] (see page 305)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 306)	:MEASure:NWIDth? [<source>] (see page 306)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format

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

Command	Query	Options and Query Returns
:MEASure:OVERshoot [<source>] (see page 307)	:MEASure:OVERshoot? [<source>] (see page 307)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 309)	:MEASure:PERiod? [<source>] (see page 309)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 310)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 310)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 311)	:MEASure:PREShoot? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format

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

Command	Query	Options and Query Returns
:MEASure:PWIDth [<source>] (see page 312)	:MEASure:PWIDth? [<source>] (see page 312)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISetime [<source>] (see page 313)	:MEASure:RISetime? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SHOW {1   ON} (see page 314)	:MEASure:SHOW? (see page 314)	{1}
:MEASure:SOURce <source1> [,<source2>] (see page 315)	:MEASure:SOURce? (see page 315)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>   EXTernal} for DSO models <source1,2> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>   EXTernal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source>   NONE}



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

Command	Query	Options and Query Returns
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see <a href="#">page 317</a> )	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNction   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see <a href="#">page 319</a> )	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNction   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format

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

Command	Query	Options and Query Returns
:MEASure:VAMplitude [<source>] (see page 321)	:MEASure:VAMplitude? [<source>] (see page 321)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<interval>][,][<source>] (see page 322)	:MEASure:VAverage? [<interval>][,][<source>] (see page 322)	<interval> ::= {CYCLe   DISPlay} <source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 323)	:MEASure:VBASe? [<source>] (see page 323)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 324)	:MEASure:VMAX? [<source>] (see page 324)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 325)	:MEASure:VMIN? [<source>] (see page 325)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format

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

Command	Query	Options and Query Returns
:MEASure:VPP [<source>] (see page 326)	:MEASure:VPP? [<source>] (see page 326)	<source> ::= {CHANnel<n>   FUNCTION   MATH   WMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 327)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 327)	<interval> ::= {CYCLe   DISPlay} <type> ::= {AC   DC} <source> ::= {CHANnel<n>   FUNCTION   MATH   WMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTime? <vtime>[,<source>] (see page 328)	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n>   FUNCTION   MATH   WMemory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNCTION   MATH   WMemory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format
:MEASure:VTOP [<source>] (see page 329)	:MEASure:VTOP? [<source>] (see page 329)	<source> ::= {CHANnel<n>   FUNCTION   MATH   WMemory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see page 330)	:MEASure:WINDow? (see page 330)	<type> ::= {MAIN   ZOOM   AUTO}

## 4 Commands Quick Reference

**Table 16** :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0   OFF}   {1   ON}} (see <a href="#">page 336</a> )	:MTESt:ALL? (see <a href="#">page 336</a> )	{0   1}
:MTESt:AMASk:CREate (see <a href="#">page 337</a> )	n/a	n/a
:MTESt:AMASk:SOURce <source> (see <a href="#">page 338</a> )	:MTESt:AMASk:SOURce? (see <a href="#">page 338</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 339</a> )	:MTESt:AMASk:UNITs? (see <a href="#">page 339</a> )	<units> ::= {CURRENT   DIVISIONs}
:MTESt:AMASk:XDELta <value> (see <a href="#">page 340</a> )	:MTESt:AMASk:XDELta? (see <a href="#">page 340</a> )	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see <a href="#">page 341</a> )	:MTESt:AMASk:YDELta? (see <a href="#">page 341</a> )	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVEfor ms? [CHANnel<n>] (see <a href="#">page 342</a> )	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see <a href="#">page 343</a> )	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see <a href="#">page 344</a> )	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVEform s? (see <a href="#">page 345</a> )	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see <a href="#">page 346</a> )	:MTESt:DATA? (see <a href="#">page 346</a> )	<mask> ::= data in IEEE 488.2 # format.
:MTESt:DELeTe (see <a href="#">page 347</a> )	n/a	n/a
:MTESt:ENABle {{0   OFF}   {1   ON}} (see <a href="#">page 348</a> )	:MTESt:ENABle? (see <a href="#">page 348</a> )	{0   1}
:MTESt:LOCK {{0   OFF}   {1   ON}} (see <a href="#">page 349</a> )	:MTESt:LOCK? (see <a href="#">page 349</a> )	{0   1}
:MTESt:RMODE <rmode> (see <a href="#">page 350</a> )	:MTESt:RMODE? (see <a href="#">page 350</a> )	<rmode> ::= {FOREver   TIME   SIGMa   WAVEforms}

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

Command	Query	Options and Query Returns
:MTESt:RMODE:FACTION:MEASure {{0   OFF}   {1   ON}} (see <a href="#">page 351</a> )	:MTESt:RMODE:FACTION:MEASure? (see <a href="#">page 351</a> )	{0   1}
:MTESt:RMODE:FACTION:PRINT {{0   OFF}   {1   ON}} (see <a href="#">page 352</a> )	:MTESt:RMODE:FACTION:PRINT? (see <a href="#">page 352</a> )	{0   1}
:MTESt:RMODE:FACTION:SAVE {{0   OFF}   {1   ON}} (see <a href="#">page 353</a> )	:MTESt:RMODE:FACTION:SAVE? (see <a href="#">page 353</a> )	{0   1}
:MTESt:RMODE:FACTION:STOP {{0   OFF}   {1   ON}} (see <a href="#">page 354</a> )	:MTESt:RMODE:FACTION:STOP? (see <a href="#">page 354</a> )	{0   1}
:MTESt:RMODE:SIGMa <level> (see <a href="#">page 355</a> )	:MTESt:RMODE:SIGMa? (see <a href="#">page 355</a> )	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODE:TIME <seconds> (see <a href="#">page 356</a> )	:MTESt:RMODE:TIME? (see <a href="#">page 356</a> )	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODE:WAVeform s <count> (see <a href="#">page 357</a> )	:MTESt:RMODE:WAVeform s? (see <a href="#">page 357</a> )	<count> ::= number of waveforms in NR1 format
:MTESt:SCALE:BIND {{0   OFF}   {1   ON}} (see <a href="#">page 358</a> )	:MTESt:SCALE:BIND? (see <a href="#">page 358</a> )	{0   1}
:MTESt:SCALE:X1 <x1_value> (see <a href="#">page 359</a> )	:MTESt:SCALE:X1? (see <a href="#">page 359</a> )	<x1_value> ::= X1 value in NR3 format
:MTESt:SCALE:XDELta <xdelta_value> (see <a href="#">page 360</a> )	:MTESt:SCALE:XDELta? (see <a href="#">page 360</a> )	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALE:Y1 <y1_value> (see <a href="#">page 361</a> )	:MTESt:SCALE:Y1? (see <a href="#">page 361</a> )	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALE:Y2 <y2_value> (see <a href="#">page 362</a> )	:MTESt:SCALE:Y2? (see <a href="#">page 362</a> )	<y2_value> ::= Y2 value in NR3 format

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:MTESt:SOURce <source> (see page 363)	:MTESt:SOURce? (see page 363)	<source> ::= {CHANnel<n>   NONE} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
n/a	:MTESt:TITLe? (see page 364)	<title> ::= a string of up to 128 ASCII characters

**Table 17** :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0   OFF}   {1   ON}} (see page 366)	:POD<n>:DISPlay? (see page 366)	{0   1} <n> ::= 1 in NR1 format
:POD<n>:SIZE <value> (see page 367)	:POD<n>:SIZE? (see page 367)	<value> ::= {SMALl   MEDium   LARGe}
:POD<n>:THReshold <type>[suffix] (see page 368)	:POD<n>:THReshold? (see page 368)	<n> ::= 1 in NR1 format <type> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V   mV   uV }

**Table 18** :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FIleName <base_name> (see page 373)	:RECall:FIleName? (see page 373)	<base_name> ::= quoted ASCII string
:RECall:MASk[:START] [<file_spec>] (see page 374)	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 375)	:RECall:PWD? (see page 375)	<path_name> ::= quoted ASCII string

**Table 18** :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:RECall:SETup[:START] [<file_spec>] (see <a href="#">page 376</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMemory<r>[:START] [<file_name>] (see <a href="#">page 377</a> )	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

**Table 19** :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILENAME <base_name> (see <a href="#">page 382</a> )	:SAVE:FILENAME? (see <a href="#">page 382</a> )	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_name>] (see <a href="#">page 383</a> )	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGE:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 384</a> )	:SAVE:IMAGE:FACTors? (see <a href="#">page 384</a> )	{0   1}
:SAVE:IMAGE:FORMat <format> (see <a href="#">page 385</a> )	:SAVE:IMAGE:FORMat? (see <a href="#">page 385</a> )	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGE:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 386</a> )	:SAVE:IMAGE:INKSaver? (see <a href="#">page 386</a> )	{0   1}
:SAVE:IMAGE:PALette <palette> (see <a href="#">page 387</a> )	:SAVE:IMAGE:PALette? (see <a href="#">page 387</a> )	<palette> ::= {COLor   GRAYscale   MONochrome}
:SAVE:MASK[:START] [<file_spec>] (see <a href="#">page 388</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 389</a> )	:SAVE:PWD? (see <a href="#">page 389</a> )	<path_name> ::= quoted ASCII string

**Table 19** :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETup[:START] [ <file_spec>] (see page 390)</file_spec>	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 391)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVEform:FORMat <format> (see page 392)	:SAVE:WAVEform:FORMat ? (see page 392)	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVEform:LENGth <length> (see page 393)	:SAVE:WAVEform:LENGth ? (see page 393)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVEform:SEGMen ted <option> (see page 394)	:SAVE:WAVEform:SEGMen ted? (see page 394)	<option> ::= {ALL   CURRent}
:SAVE:WMEMory:SOURce <source> (see page 395)	:SAVE:WMEMory:SOURce? (see page 395)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:START] [<file_name>] (see page 396)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".



**Table 20** :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see <a href="#">page 399</a> )	:SYSTem:DATE? (see <a href="#">page 399</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 400</a> )	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERROR? (see <a href="#">page 401</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 569</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 402</a> )	:SYSTem:LOCK? (see <a href="#">page 402</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PRESet (see <a href="#">page 403</a> )	n/a	See :SYSTem:PRESet (see <a href="#">page 403</a> )
:SYSTem:PROTection:LOCK <value> (see <a href="#">page 406</a> )	:SYSTem:PROTection:LOCK? (see <a href="#">page 406</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 407</a> )	:SYSTem:SETup? (see <a href="#">page 407</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 409</a> )	:SYSTem:TIME? (see <a href="#">page 409</a> )	<time> ::= hours,minutes,seconds in NR1 format

**Table 21** :TIMEbase Commands Summary

Command	Query	Options and Query Returns
:TIMEbase:MODE <value> (see <a href="#">page 413</a> )	:TIMEbase:MODE? (see <a href="#">page 413</a> )	<value> ::= {MAIN   WINDOW   XY   ROLL}
:TIMEbase:POSition <pos> (see <a href="#">page 414</a> )	:TIMEbase:POSition? (see <a href="#">page 414</a> )	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMEbase:RANGe <range_value> (see <a href="#">page 415</a> )	:TIMEbase:RANGe? (see <a href="#">page 415</a> )	<range_value> ::= 50 ns through 500 s in NR3 format

## 4 Commands Quick Reference

**Table 21** :TIMebase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMebase:REfERENCE {LEFT   CENTer   RIGHT} (see <a href="#">page 416</a> )	:TIMebase:REfERENCE? (see <a href="#">page 416</a> )	<return_value> ::= {LEFT   CENTer   RIGHT}
:TIMebase:SCALe <scale_value> (see <a href="#">page 417</a> )	:TIMebase:SCALe? (see <a href="#">page 417</a> )	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 418</a> )	:TIMebase:VERNier? (see <a href="#">page 418</a> )	{0   1}
:TIMebase:WINDow:POSi tion <pos> (see <a href="#">page 419</a> )	:TIMebase:WINDow:POSi tion? (see <a href="#">page 419</a> )	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMebase:WINDow:RANG e <range_value> (see <a href="#">page 420</a> )	:TIMebase:WINDow:RANG e? (see <a href="#">page 420</a> )	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCAL e <scale_value> (see <a href="#">page 421</a> )	:TIMebase:WINDow:SCAL e? (see <a href="#">page 421</a> )	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window

**Table 22** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}   {1   ON}} (see <a href="#">page 426</a> )	:TRIGger:HFReject? (see <a href="#">page 426</a> )	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see <a href="#">page 427</a> )	:TRIGger:HOLDoff? (see <a href="#">page 427</a> )	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:HIGH <level>, <source> (see <a href="#">page 428</a> )	:TRIGger:LEVel:HIGH? <source> (see <a href="#">page 428</a> )	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see <a href="#">page 429</a> )	:TRIGger:LEVel:LOW? <source> (see <a href="#">page 429</a> )	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format

**Table 22** General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:MODE <mode> (see <a href="#">page 430</a> )	:TRIGger:MODE? (see <a href="#">page 430</a> )	<mode> ::= {EDGE   GLITCh   PATtern   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 431</a> )	:TRIGger:NREJect? (see <a href="#">page 431</a> )	{0   1}
:TRIGger:SWEep <sweep> (see <a href="#">page 432</a> )	:TRIGger:SWEep? (see <a href="#">page 432</a> )	<sweep> ::= {AUTO   NORMal}

**Table 23** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LFReject} (see <a href="#">page 434</a> )	:TRIGger[:EDGE]:COUPling? (see <a href="#">page 434</a> )	{AC   DC   LFReject}
:TRIGger[:EDGE]:LEVel <level> [, <source>] (see <a href="#">page 435</a> )	:TRIGger[:EDGE]:LEVel? [<source>] (see <a href="#">page 435</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. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   EXTernal } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LFReject   HFReject} (see <a href="#">page 436</a> )	:TRIGger[:EDGE]:REJect? (see <a href="#">page 436</a> )	{OFF   LFReject   HFReject}

## 4 Commands Quick Reference

**Table 23** :TRIGger[:EDGE] Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:SLOPe <polarity> (see page 437)	:TRIGger[:EDGE]:SLOPe ? (see page 437)	<polarity> ::= {POSitive   NEGative   EITHER   ALTernate}
:TRIGger[:EDGE]:SOURc e <source> (see page 438)	:TRIGger[:EDGE]:SOURc e? (see page 438)	<source> ::= {CHANnel<n>   EXTernal   LINE   WGEN} for the DSO models <source> ::= {CHANnel<n>   DIGital<d>   EXTernal   LINE   WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

**Table 24** :TRIGger:GLITCh Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITCh:GREAt erthan <greater_than_time>[s uffix] (see page 441)	:TRIGger:GLITCh:GREAt erthan? (see page 441)	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITCh:LESSt han <less_than_time>[suff ix] (see page 442)	:TRIGger:GLITCh:LESSt han? (see page 442)	<less_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITCh:LEVEl <level> [<source>] (see page 443)	:TRIGger:GLITCh:LEVEl ? (see page 443)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

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

Command	Query	Options and Query Returns
:TRIGger:GLITch:POLarity <polarity> (see page 444)	:TRIGger:GLITch:POLarity? (see page 444)	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 445)	:TRIGger:GLITch:QUALifier? (see page 445)	<qualifier> ::= {GREATERthan   LESSthan   RANGE}
:TRIGger:GLITch:RANGE <less_than_time>[suffix], <greater_than_time>[suffix] (see page 446)	:TRIGger:GLITch:RANGE? (see page 446)	<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 447)	:TRIGger:GLITch:SOURce? (see page 447)	<source> ::= {CHANnel<n>   DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

**Table 25** :TRIGger:PATtern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATtern <string>[, <edge_source>, <edge>] (see page 449)	:TRIGger:PATtern? (see page 450)	<string> ::= "nn...n" where n ::= {0   1   X   R   F} when <base> = ASCII <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F   X   \$} when <base> = HEX <edge_source> ::= {CHANnel<n>   NONE} for DSO models <edge_source> ::= {CHANnel<n>   DIGital<d>   NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive   NEGative}
:TRIGger:PATtern:FORM at <base> (see page 451)	:TRIGger:PATtern:FORM at? (see page 451)	<base> ::= {ASCII   HEX}
:TRIGger:PATtern:QUALifier <qualifier> (see page 452)	:TRIGger:PATtern:QUALifier? (see page 452)	<qualifier> ::= ENTERed

## 4 Commands Quick Reference

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

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see <a href="#">page 454</a> )	:TRIGger:TV:LINE? (see <a href="#">page 454</a> )	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see <a href="#">page 455</a> )	:TRIGger:TV:MODE? (see <a href="#">page 455</a> )	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LFIELD1   LFIELD2   LALTERNATE}
:TRIGger:TV:POLarity <polarity> (see <a href="#">page 456</a> )	:TRIGger:TV:POLarity? (see <a href="#">page 456</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:TV:SOURce <source> (see <a href="#">page 457</a> )	:TRIGger:TV:SOURce? (see <a href="#">page 457</a> )	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see <a href="#">page 458</a> )	:TRIGger:TV:STANdard? (see <a href="#">page 458</a> )	<standard> ::= {NTSC   PAL   PALM   SECam}

**Table 27** :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see <a href="#">page 467</a> )	:WAVEform:BYTeorder? (see <a href="#">page 467</a> )	<value> ::= {LSBFirst   MSBFirst}
n/a	:WAVEform:COUNT? (see <a href="#">page 468</a> )	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see <a href="#">page 469</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 471</a> )	:WAVEform:FORMat? (see <a href="#">page 471</a> )	<value> ::= {WORD   BYTE   ASCII}

**Table 27** :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINTs <# points> (see page 472)	:WAVeform:POINTs? (see page 472)	<# 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 474)	:WAVeform:POINTs:MODE ? (see page 474)	<points_mode> ::= {NORMAl   MAXimum   RAW}
n/a	:WAVeform:PREamble? (see page 476)	<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>• 3 for AVERAge type</li> <li>• 4 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 page 479)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGmented:TAG? (see page 480)	<time_tag> ::= in NR3 format (with Option SGM)

## 4 Commands Quick Reference

**Table 27** :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SOURce <source> (see <a href="#">page 481</a> )	:WAVeform:SOURce? (see <a href="#">page 481</a> )	<source> ::= {CHANnel<n>   FUNction   MATH} for DSO models <source> ::= {CHANnel<n>   POD{1   2}   BUS{1   2}   FUNction   MATH} for MSO models <n> ::= 1 to (# analog channels) in NR1 format
:WAVeform:SOURce:SUBS ource <subsource> (see <a href="#">page 485</a> )	:WAVeform:SOURce:SUBS ource? (see <a href="#">page 485</a> )	<subsource> ::= {SUB0   RX   MOSI}
n/a	:WAVeform:TYPE? (see <a href="#">page 486</a> )	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVeform:UNSigned {{0   OFF}   {1   ON}} (see <a href="#">page 487</a> )	:WAVeform:UNSigned? (see <a href="#">page 487</a> )	{0   1}
:WAVeform:VIEW <view> (see <a href="#">page 488</a> )	:WAVeform:VIEW? (see <a href="#">page 488</a> )	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see <a href="#">page 489</a> )	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see <a href="#">page 490</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see <a href="#">page 491</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see <a href="#">page 492</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see <a href="#">page 493</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see <a href="#">page 494</a> )	<return_value> ::= y-reference value in the current preamble in NR1 format



**Table 28** :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:FREQuency <frequency> (see page 497)	:WGEN:FREQuency? (see page 497)	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNcTion <signal> (see page 498)	:WGEN:FUNcTion? (see page 499)	<signal> ::= {SINusoid   SQUare   RAMP   PULSe   NOISe   DC}
:WGEN:FUNcTion:PULSe: WIDTh <width> (see page 500)	:WGEN:FUNcTion:PULSe: WIDTh? (see page 500)	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNcTion:RAMP:S YMMetry <percent> (see page 501)	:WGEN:FUNcTion:RAMP:S YMMetry? (see page 501)	<percent> ::= symmetry percentage from 0% to 100% in NR3 format
:WGEN:FUNcTion:SQUare :DCYcLe <percent> (see page 502)	:WGEN:FUNcTion:SQUare :DCYcLe? (see page 502)	<percent> ::= duty cycle percentage from 20% to 80% in NR3 format
:WGEN:OUTPut {{0   OFF}   {1   ON}} (see page 503)	:WGEN:OUTPut? (see page 503)	{0   1}
:WGEN:OUTPut:LOAD <impedance> (see page 504)	:WGEN:OUTPut:LOAD? (see page 504)	<impedance> ::= {ONEMeg   FIFTy}
:WGEN:PERiod <period> (see page 505)	:WGEN:PERiod? (see page 505)	<period> ::= period in seconds in NR3 format
:WGEN:RST (see page 506)	n/a	n/a
:WGEN:VOLTagE <amplitude> (see page 507)	:WGEN:VOLTagE? (see page 507)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTagE:HIGh <high> (see page 508)	:WGEN:VOLTagE:HIGh? (see page 508)	<high> ::= high-level voltage in volts, in NR3 format
:WGEN:VOLTagE:LOW <low> (see page 509)	:WGEN:VOLTagE:LOW? (see page 509)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTagE:OFFSet <offset> (see page 510)	:WGEN:VOLTagE:OFFSet? (see page 510)	<offset> ::= offset in volts in NR3 format

## 4 Commands Quick Reference

**Table 29** :WMEMemory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEMemory<r>:CLEar (see <a href="#">page 513</a> )	n/a	<r> ::= 1-2 in NR1 format
:WMEMemory<r>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 514</a> )	:WMEMemory<r>:DISPlay? (see <a href="#">page 514</a> )	<r> ::= 1-2 in NR1 format {0   1}
:WMEMemory<r>:LABel <string> (see <a href="#">page 515</a> )	:WMEMemory<r>:LABel? (see <a href="#">page 515</a> )	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEMemory<r>:SAVE <source> (see <a href="#">page 516</a> )	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEMemory<r>:SKEW <skew> (see <a href="#">page 517</a> )	:WMEMemory<r>:SKEW? (see <a href="#">page 517</a> )	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format
:WMEMemory<r>:YOFFset <offset>[suffix] (see <a href="#">page 518</a> )	:WMEMemory<r>:YOFFset? (see <a href="#">page 518</a> )	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V   mV}
:WMEMemory<r>:YRANge <range>[suffix] (see <a href="#">page 519</a> )	:WMEMemory<r>:YRANge? (see <a href="#">page 519</a> )	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V   mV}
:WMEMemory<r>:YSCale <scale>[suffix] (see <a href="#">page 520</a> )	:WMEMemory<r>:YSCale? (see <a href="#">page 520</a> )	<r> ::= 1-2 in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V   mV}

## Syntax Elements

- "Number Format" on page 91
- "<NL> (Line Terminator)" on page 91
- "[ ] (Optional Syntax Terms)" on page 91
- "{ } (Braces)" on page 91
- " ::= (Defined As)" on page 91
- "< > (Angle Brackets)" on page 92
- "... (Ellipsis)" on page 92
- "n,...,p (Value Ranges)" on page 92
- "d (Digits)" on page 92
- "Quoted ASCII String" on page 92
- "Definite-Length Block Response Data" on page 92

### Number Format

NR1 specifies integer data.

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

### <NL> (Line Terminator)

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

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

### [ ] (Optional Syntax Terms)

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

### { } (Braces)

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

### ::= (Defined As)

::= means "defined as".

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

### **< > (Angle Brackets)**

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

### **... (Ellipsis)**

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

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

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

### **d (Digits)**

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

### **Quoted ASCII String**

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

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

has a quoted ASCII string of:

```
'One'
```

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

### **Definite-Length Block Response Data**

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

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

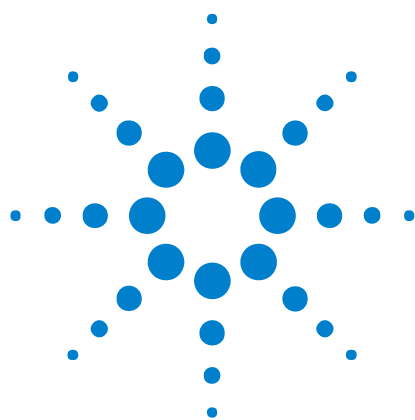
```
#800001000<1000 bytes of data> <NL>
```

**8** is the number of digits that follow

**00001000** is the number of bytes to be transmitted

**<1000 bytes of data>** is the actual data

## 4 Commands Quick Reference



## 5 Common (\*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See "Introduction to Common (\*) Commands" on page 97.

**Table 30** Common (\*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 99</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 100</a> )	*ESE? (see <a href="#">page 100</a> )	<p>&lt;mask&gt; ::= 0 to 255; an integer in NR1 format:</p> <pre> 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 </pre>
n/a	*ESR? (see <a href="#">page 102</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 102</a> )	<p>AGILENT TECHNOLOGIES,&lt;model&gt;,&lt;serial number&gt;,X.XX.XX</p> <p>&lt;model&gt; ::= the model number of the instrument</p> <p>&lt;serial number&gt; ::= the serial number of the instrument</p> <p>&lt;X.XX.XX&gt; ::= the software revision of the instrument</p>
n/a	*LRN? (see <a href="#">page 105</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 106</a> )	*OPC? (see <a href="#">page 106</a> )	ASCII "1" is placed in the output queue when all pending device operations have completed.



## 5 Common (\*) Commands

**Table 30** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see <a href="#">page 107</a> )	<pre>&lt;return_value&gt; ::= 0,0,&lt;license info&gt; &lt;license info&gt; ::= &lt;All field&gt;, &lt;reserved&gt;, &lt;MSO&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;Power Measurements&gt;, &lt;reserved&gt;, &lt;Segmented Memory&gt;, &lt;Mask Test&gt;, &lt;reserved&gt;, &lt;Bandwidth&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;reserved&gt;, &lt;Waveform Generator&gt;, &lt;reserved&gt;, &lt;reserved&gt; &lt;All field&gt; ::= {0   All} &lt;reserved&gt; ::= 0 &lt;MSO&gt; ::= {0   MSO} &lt;Power Measurements&gt; ::= {0   PWR} &lt;Segmented Memory&gt; ::= {0   SGM} &lt;Mask Test&gt; ::= {0   MASK} &lt;Bandwidth&gt; ::= {0   BW10   BW20} &lt;Waveform Generator&gt; ::= {0   WAVEGEN}</pre>
*RCL <value> (see <a href="#">page 108</a> )	n/a	<value> ::= {0   1   4   5   6   7   8   9}
*RST (see <a href="#">page 109</a> )	n/a	See *RST (Reset) (see <a href="#">page 109</a> )
*SAV <value> (see <a href="#">page 112</a> )	n/a	<value> ::= {0   1   4   5   6   7   8   9}
*SRE <mask> (see <a href="#">page 113</a> )	*SRE? (see <a href="#">page 114</a> )	<pre>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= 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</pre>



Table 30 Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see <a href="#">page 115</a> )	<p>&lt;value&gt; ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <pre> 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. </pre>
*TRG (see <a href="#">page 117</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 118</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 119</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 Common (\*) Commands

### NOTE

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

---

## \*CLS (Clear Status)

**C** (see [page 608](#))

### 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 97
  - ["\\*STB \(Read Status Byte\)"](#) on page 115
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 100
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 102
  - ["\\*SRE \(Service Request Enable\)"](#) on page 113
  - [":SYSTem:ERRor"](#) on page 401

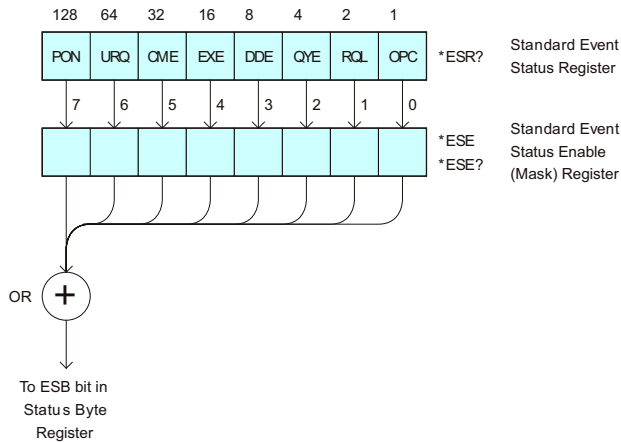
### \*ESE (Standard Event Status Enable)

**C** (see page 608)

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

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

**Query Syntax** \*ESE?

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

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

<mask\_argument> ::= 0,...,255; an integer in NR1 format.

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 97
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 102
  - ["\\*OPC \(Operation Complete\)"](#) on page 106
  - ["\\*CLS \(Clear Status\)"](#) on page 99

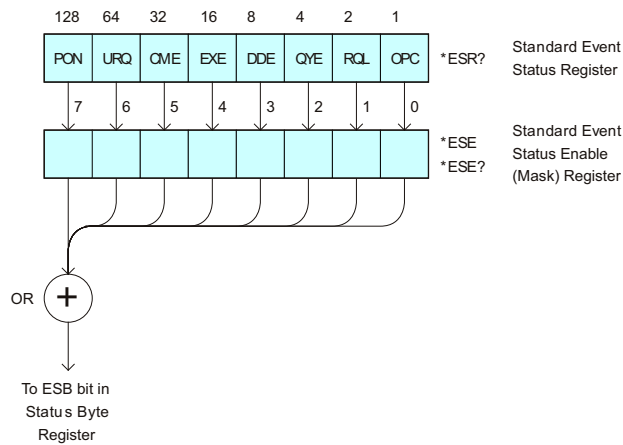
### \*ESR (Standard Event Status Register)

**C** (see page 608)

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

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

**Return Format**

<status><NL>

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

**NOTE**

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

---

- See Also**
- "Introduction to Common (\*) Commands" on page 97
  - "\*ESE (Standard Event Status Enable)" on page 100
  - "\*OPC (Operation Complete)" on page 106
  - "\*CLS (Clear Status)" on page 99
  - ":SYSTem:ERRor" on page 401

## \*IDN (Identification Number)

**C** (see [page 608](#))

**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 97
  - ["\\*OPT \(Option Identification\)"](#) on page 107



## \*LRN (Learn Device Setup)

**C** (see [page 608](#))

### Query Syntax

\*LRN?

The \*LRN? query result contains the current state of the instrument. This query is similar to the :SYSTEM:SETup? (see [page 407](#)) 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 97
  - "[\\*RCL \(Recall\)](#)" on page 108
  - "[\\*SAV \(Save\)](#)" on page 112
  - "[:SYSTEM:SETup](#)" on page 407

### \*OPC (Operation Complete)

**C** (see [page 608](#))

**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 97
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 100
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 102
  - ["\\*CLS \(Clear Status\)"](#) on page 99

## \*OPT (Option Identification)

**C** (see page 608)

### Query Syntax \*OPT?

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

### Return Format

```
0,0,<license info>

<license info> ::= <All field>, <reserved>, <MSO>, <reserved>,
                 <reserved>, <reserved>, <reserved>,
                 <reserved>, <reserved>,
                 <Power Measurements>, <reserved>,
                 <Segmented Memory>, <Mask Test>, <reserved>,
                 <Bandwidth>, <reserved>, <reserved>, <reserved>,
                 <reserved>, <reserved>, <reserved>,
                 <Waveform Generator>, <reserved>, <reserved>

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

<reserved> ::= 0

<MSO> ::= {0 | MSO}

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

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

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

<Bandwidth> ::= {0 | BW10 | BW20}

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

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

The \*OPT? query returns the following:

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

- See Also**
- ["Introduction to Common \(\\*\) Commands" on page 97](#)
  - ["\\*IDN \(Identification Number\)" on page 104](#)

### \*RCL (Recall)

**C** (see [page 608](#))

**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 97
  - ["\\*SAV \(Save\)"](#) on page 112

**\*RST (Reset)**

**C** (see page 608)

**Command Syntax** \*RST

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

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

Reset conditions are:

<b>Acquire Menu</b>	
Mode	Normal
Averaging	Off
# Averages	8

<b>Analog Channel Menu</b>	
Channel 1	On
Channel 2	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	10:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm (cannot be changed)
Units	Volts
Skew	0

<b>Cursor Menu</b>	
Source	Channel 1

## 5 Common (\*) Commands

<b>Digital Channel Menu (MSO models only)</b>	
Channel 0 - 7	Off
Labels	Off
Threshold	TTL (1.4V)

<b>Display Menu</b>	
Persistence	Off
Grid	33%

<b>Quick Meas Menu</b>	
Source	Channel 1

<b>Run Control</b>	
	Scope is running

<b>Time Base Menu</b>	
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

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	60 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 97
  - [":SYSTem:PRESet"](#) on page 403

**Example Code**

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

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

### \*SAV (Save)

**C** (see [page 608](#))

**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 97
  - ["\\*RCL \(Recall\)"](#) on page 108



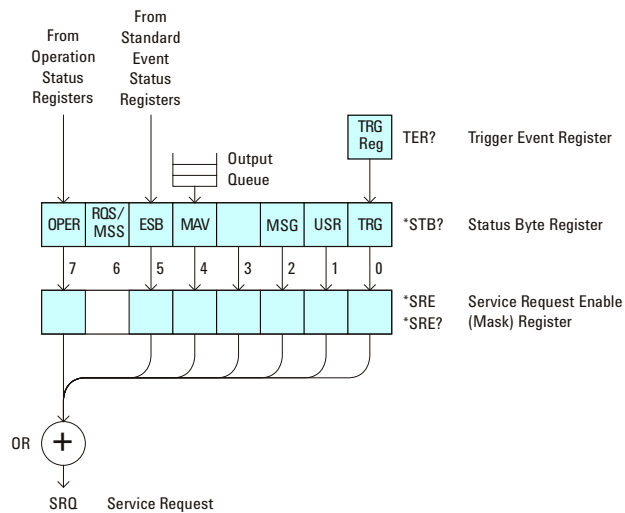
**\*SRE (Service Request Enable)**

**C** (see [page 608](#))

**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 33** Service Request Enable Register (SRE)

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

**Query Syntax** \*SRE?

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

**Return Format** <mask><NL>

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

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 97
  - ["\\*STB \(Read Status Byte\)"](#) on page 115
  - ["\\*CLS \(Clear Status\)"](#) on page 99

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

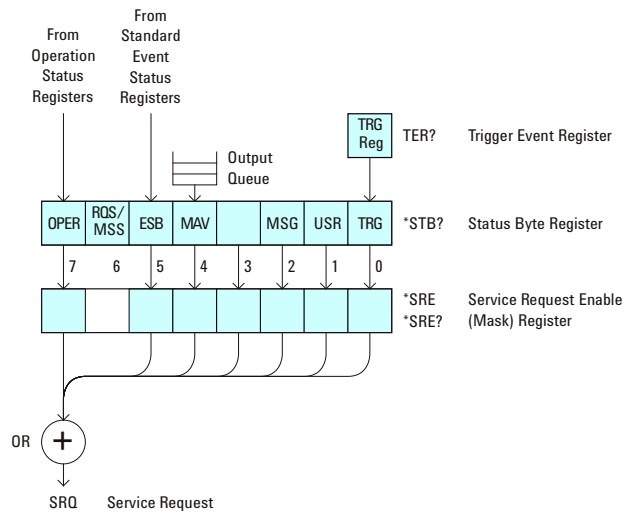
**C** (see [page 608](#))

**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 34** Status Byte Register (STB)

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

**NOTE**

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

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 97
  - ["\\*SRE \(Service Request Enable\)"](#) on page 113

## \*TRG (Trigger)

**C** (see [page 608](#))

### 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 97
  - [":DIGitize"](#) on page 133
  - [":RUN"](#) on page 150
  - [":STOP"](#) on page 154

### \*TST (Self Test)

**C** (see [page 608](#))

**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 97

## \*WAI (Wait To Continue)

**C** (see [page 608](#))

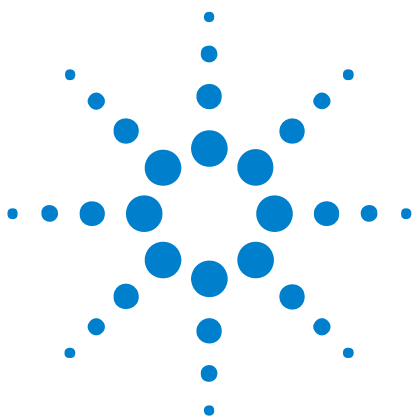
**Command Syntax** \*WAI

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 97

## 5 Common (\*) Commands





## 6 Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree. See "[Introduction to Root \(:\) Commands](#)" on page 124.

**Table 35** Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see <a href="#">page 125</a> )	:ACTivity? (see <a href="#">page 125</a> )	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER? (see <a href="#">page 126</a> )	{0   1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see <a href="#">page 127</a> )	n/a	<source> ::= CHANnel<n> for DSO models <source> ::= {CHANnel<n>   DIGital<d>   POD1   POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:AUToscale:AMODE <value> (see <a href="#">page 129</a> )	:AUToscale:AMODE? (see <a href="#">page 129</a> )	<value> ::= {NORMal   CURRent}}
:AUToscale:CHANnels <value> (see <a href="#">page 130</a> )	:AUToscale:CHANnels? (see <a href="#">page 130</a> )	<value> ::= {ALL   DISPlayed}}
:AUToscale:FDEBug {{0   OFF}   {1   ON}} (see <a href="#">page 131</a> )	:AUToscale:FDEBug? (see <a href="#">page 131</a> )	{0   1}



## 6 Root (:) Commands

**Table 35** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:BLANK [<source>] (see <a href="#">page 132</a> )	n/a	<source> ::= {CHANnel<n>}   FUNction   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNction   MATH} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:DIGitize [<source>[, ..., <source >]] (see <a href="#">page 133</a> )	n/a	<source> ::= {CHANnel<n>   FUNction   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNction   MATH} for MSO models <source> can be repeated up to 5 times <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:MTEenable <n> (see <a href="#">page 135</a> )	:MTEenable? (see <a href="#">page 135</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:MTEregister[:EVENT]? (see <a href="#">page 137</a> )	<n> ::= 16-bit integer in NR1 format
:OPEE <n> (see <a href="#">page 139</a> )	:OPEE? (see <a href="#">page 139</a> )	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister:CONDiti on? (see <a href="#">page 141</a> )	<n> ::= 15-bit integer in NR1 format
n/a	:OPERregister[:EVENT]? (see <a href="#">page 143</a> )	<n> ::= 15-bit integer in NR1 format

**Table 35** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:OVLenable <mask> (see <a href="#">page 145</a> )	:OVLenable? (see <a href="#">page 146</a> )	<mask> ::= 16-bit integer in NR1 format as shown:  Bit Weight Input ----- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see <a href="#">page 147</a> )	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINT [<options>] (see <a href="#">page 149</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 150</a> )	n/a	n/a
n/a	:SERial (see <a href="#">page 151</a> )	<return value> ::= unquoted string containing serial number
:SINGle (see <a href="#">page 152</a> )	n/a	n/a
n/a	:STATus? <display> (see <a href="#">page 153</a> )	{0   1} <display> ::= {CHANnel<n>   DIGital<d>   POD{1   2}   BUS{1   2}   FUNctioN   MATH} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:STOP (see <a href="#">page 154</a> )	n/a	n/a

## 6 Root (:) Commands

**Table 35** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see <a href="#">page 155</a> )	{0   1}
:VIEW <source> (see <a href="#">page 156</a> )	n/a	<p>&lt;source&gt; ::= {CHANnel&lt;n&gt;   FUNCTION   MATH} for DSO models            &lt;source&gt; ::= {CHANnel&lt;n&gt;   DIGital&lt;d&gt;   POD{1   2}   BUS{1   2}   FUNCTION   MATH} for MSO models            &lt;n&gt; ::= 1 to (# analog channels) in NR1 format            &lt;d&gt; ::= 0 to (# digital channels - 1) in NR1 format</p>

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

**:ACTivity**

**N** (see [page 608](#))

**Command Syntax** :ACTivity

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

**Query Syntax** :ACTivity?

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

**NOTE**

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

**Return Format**

<edges>,<levels><NL>

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

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

bit 0 ::= DIGital 0

bit 15 ::= DIGital 15

**NOTE**

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

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

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

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 124
  - "[:POD<n>:THReshold](#)" on page 368
  - "[:DIGital<d>:THReshold](#)" on page 223

## :AER (Arm Event Register)

**C** (see [page 608](#))

**Query Syntax** :AER?

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

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

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

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

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 139
  - [":OPERRegister:CONDition \(Operation Status Condition Register\)"](#) on page 141
  - [":OPERRegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 143
  - ["\\*STB \(Read Status Byte\)"](#) on page 115
  - ["\\*SRE \(Service Request Enable\)"](#) on page 113

## :AUToscale

**C** (see [page 608](#))

**Command Syntax** :AUToscale

```
:AUToscale [<source>[,...,<source>]]
<source> ::= CHANnel<n> for the DSO models
<source> ::= {DIGital<d> | POD1 | POD2 | CHANnel<n>} for the
              MSO models
<n> ::= 1 to (# analog channels) in NR1 format
<d> ::= 0 to (# digital channels - 1) in NR1 format
The <source> parameter may be repeated up to 5 times.
```

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

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see [":AUToscale:CHANnels"](#) on page 130) is set to DISplayed channels. Then, the autoscale is performed.

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

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

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

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

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

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":AUToscale:CHANnels"](#) on page 130
  - [":AUToscale:AMODE"](#) on page 129

**Example Code**

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

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617



**:AUToscale:AMODE**

**N** (see [page 608](#))

**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 127
  - "[:AUToscale:CHANnels](#)" on page 130
  - "[:ACQUIRE:TYPE](#)" on page 169
  - "[:ACQUIRE:MODE](#)" on page 161

## :AUToscale:CHANnels

**N** (see [page 608](#))

**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 127
  - "[:AUToscale:AMODE](#)" on page 129
  - "[:VIEW](#)" on page 156
  - "[:BLANK](#)" on page 132

**:AUToscale:FDEBug**

**N** (see [page 608](#))

**Command Syntax** :AUToscale:FDEBug <on\_off>  
 <on\_off> ::= {{1 | ON} | {0 | OFF}}

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

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

Channel coupling is maintained for easy viewing of oscillating signals.

**Query Syntax** :AUToscale:FDEBug?

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

**Return Format** <on\_off><NL>  
 <on\_off> ::= {1 | 0}

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":AUToscale"](#) on page 127

**:BLANK**

**N** (see [page 608](#))

**Command Syntax**

```
:BLANK [<source>]
```

```
<source> ::= {CHANnel<n> | FUNCTION | MATH}
           for the DSO models
```

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

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

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

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

**NOTE**

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

**NOTE**

MATH is an alias for FUNCTION.

**See Also**

- ["Introduction to Root \(:\) Commands"](#) on page 124
- [":DISPlay:CLEar"](#) on page 227
- [":CHANnel<n>:DISPlay"](#) on page 196
- [":DIGital<d>:DISPlay"](#) on page 219
- [":FUNCTION:DISPlay"](#) on page 243
- [":POD<n>:DISPlay"](#) on page 366
- [":STATus"](#) on page 153
- [":VIEW"](#) on page 156

**Example Code**

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

**:DIGitize**

**C** (see [page 608](#))

**Command Syntax** :DIGitize [<source>[,...,<source>]]

<source> ::= {CHANnel<n> | FUNction | MATH}  
for the DSO models

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

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

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

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

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

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

**NOTE**

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

**NOTE**

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

**NOTE**

MATH is an alias for FUNction.

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":RUN"](#) on page 150
  - [":SINGLE"](#) on page 152
  - [":STOP"](#) on page 154
  - [":TIMEbase:MODE"](#) on page 413
  - [Chapter 7, "ACQUIRE Commands,"](#) starting on page 157
  - [Chapter 26, "WAVEFORM Commands,"](#) starting on page 459

## 6 Root (:) Commands

### Example Code

```
' Capture an acquisition using :DIGitize.
```

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

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

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

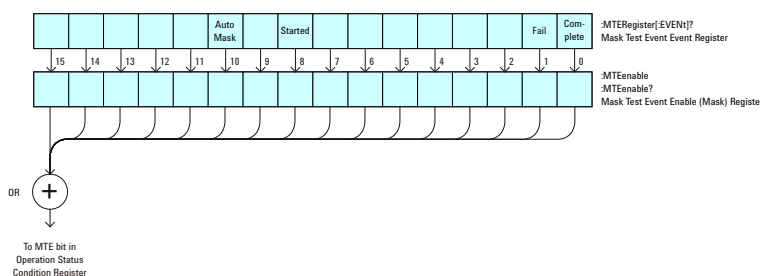
## :MTEenable (Mask Test Event Enable Register)

**N** (see page 608)

**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 36** 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 126

## 6 Root (:) Commands

- ":CHANnel<n>:PROTection" on page 206
- ":OPERRegister[:EVENT] (Operation Status Event Register)" on page 143
- ":OVLenable (Overload Event Enable Register)" on page 145
- ":OVLRegister (Overload Event Register)" on page 147
- "\*\*STB (Read Status Byte)" on page 115
- "\*\*SRE (Service Request Enable)" on page 113

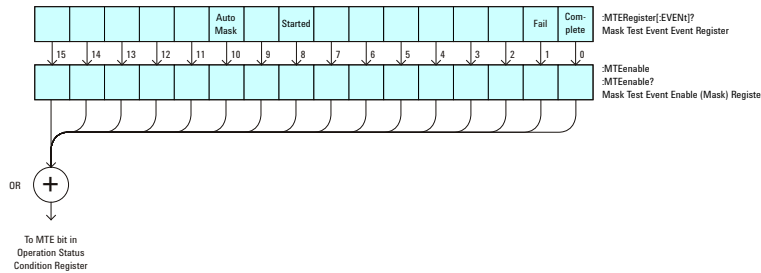


## :MTERegister[:EVENT] (Mask Test Event Event Register)

**N** (see page 608)

**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 37** 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 206
  - ":OPEE (Operation Status Enable Register)" on page 139
  - ":OPERegister:CONDition (Operation Status Condition Register)" on page 141
  - ":OVLenable (Overload Event Enable Register)" on page 145
  - ":OVLRegister (Overload Event Register)" on page 147

## 6 Root (:) Commands

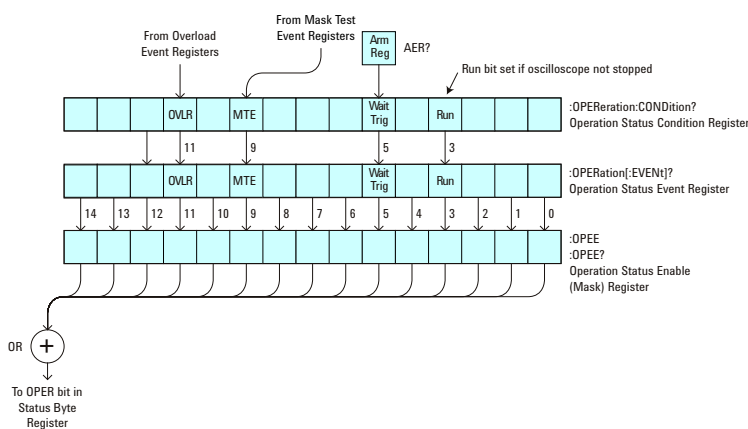
- `*STB (Read Status Byte)` on page 115
- `*SRE (Service Request Enable)` on page 113

## :OPEE (Operation Status Enable Register)

**C** (see page 608)

**Command Syntax** :OPEE <mask>  
 <mask> ::= 15-bit integer

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



**Table 38** Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
14-12	---	---	(Not used.)
11	OVL R	Overload	Event when 50Ω input overload occurs.
10	---	---	(Not used.)
9	MTE	Mask Test Event	Event when mask test event occurs.
8-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)
3	Run	Running	Event when the oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

**Query Syntax** :OPEE?

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

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

<value> ::= integer in NR1 format.

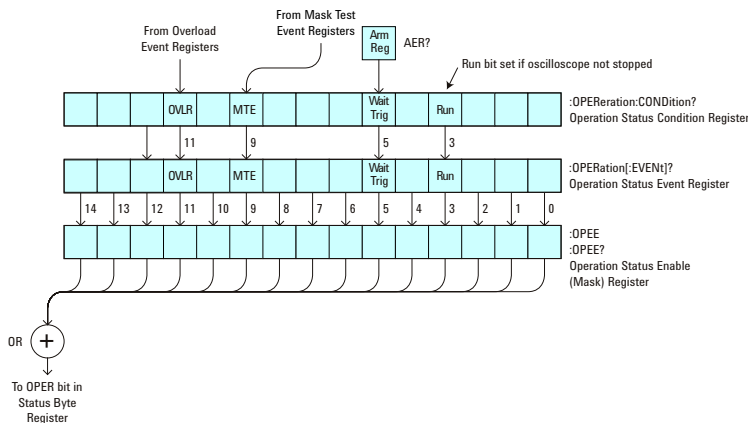
- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":AER \(Arm Event Register\)"](#) on page 126
  - [":CHANnel<n>:PROTection"](#) on page 206
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 143
  - [":OVLenable \(Overload Event Enable Register\)"](#) on page 145
  - [":OVLRegister \(Overload Event Register\)"](#) on page 147
  - ["\\*STB \(Read Status Byte\)"](#) on page 115
  - ["\\*SRE \(Service Request Enable\)"](#) on page 113

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

**C** (see page 608)

**Query Syntax** :OPERRegister:CONDition?

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



**Table 39** Operation Status Condition Register

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

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

- See Also**
- "Introduction to Root (:) Commands" on page 124
  - ":CHANnel<n>:PROTection" on page 206
  - ":OPEE (Operation Status Enable Register)" on page 139

## 6 Root (:) Commands

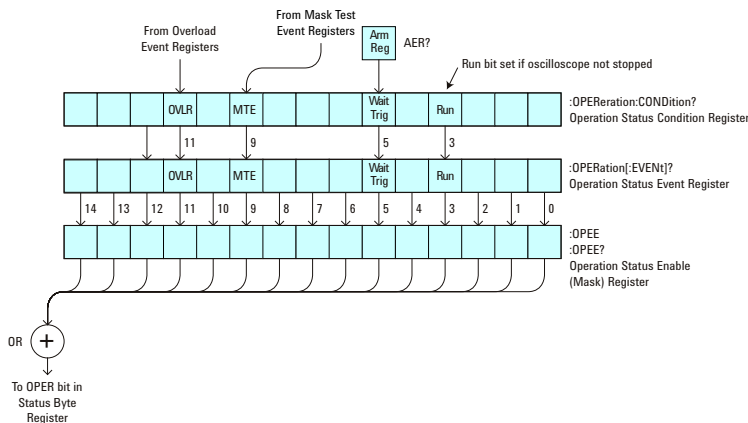
- ":OPERRegister[:EVENT] (Operation Status Event Register)" on page 143
- ":OVLenable (Overload Event Enable Register)" on page 145
- ":OVLRegister (Overload Event Register)" on page 147
- "\*STB (Read Status Byte)" on page 115
- "\*SRE (Service Request Enable)" on page 113
- ":MTERegister[:EVENT] (Mask Test Event Event Register)" on page 137
- ":MTEenable (Mask Test Event Enable Register)" on page 135

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

**C** (see page 608)

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

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



**Table 40** Operation Status Event Register

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

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

- See Also**
- "Introduction to Root (: ) Commands" on page 124
  - ":CHANnel<n>:PROTEction" on page 206

## 6 Root (:) Commands

- ":OPEE (Operation Status Enable Register)" on page 139
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 141
- ":OVLenable (Overload Event Enable Register)" on page 145
- ":OVLRegister (Overload Event Register)" on page 147
- "\*STB (Read Status Byte)" on page 115
- "\*SRE (Service Request Enable)" on page 113
- ":MTERegister[:EVENT] (Mask Test Event Event Register)" on page 137
- ":MTEenable (Mask Test Event Enable Register)" on page 135



## :OVLenable (Overload Event Enable Register)

**C** (see page 608)

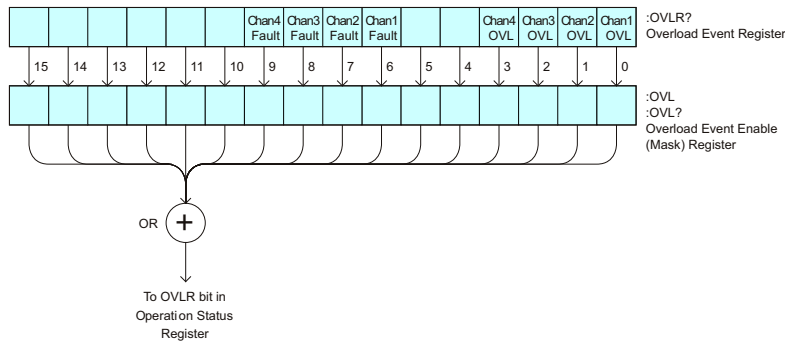
**Command Syntax** :OVLenable <enable\_mask>  
 <enable\_mask> ::= 16-bit integer

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

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

**NOTE**

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



**Table 41** Overload Event Enable Register (OVL)

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

**Table 41** Overload Event Enable Register (OVL) (continued)

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

**Query Syntax** :OVLenable?

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

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

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

- See Also**
- ["Introduction to Root \(:\) Commands" on page 124](#)
  - [":CHANnel<n>:PROTection" on page 206](#)
  - [":OPEE \(Operation Status Enable Register\)" on page 139](#)
  - [":OPERegister:CONDition \(Operation Status Condition Register\)" on page 141](#)
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)" on page 143](#)
  - [":OVLRegister \(Overload Event Register\)" on page 147](#)
  - ["\\*STB \(Read Status Byte\)" on page 115](#)
  - ["\\*SRE \(Service Request Enable\)" on page 113](#)

## :OVLRegister (Overload Event Register)

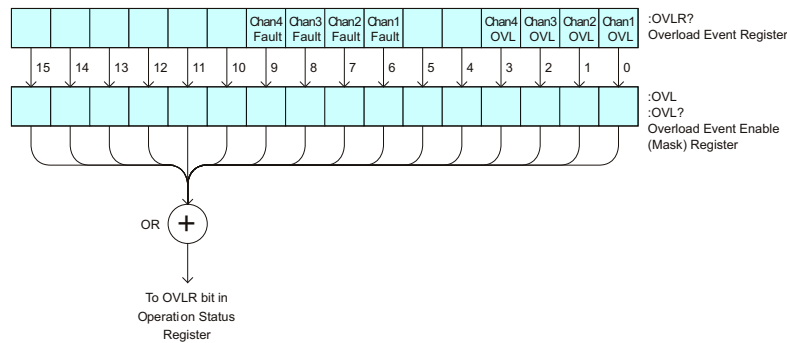
**C** (see page 608)

**Query Syntax** :OVLRegister?

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

### NOTE

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



**Table 42** Overload Event Register (OVL)

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

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

## 6 Root (:) Commands

<value> ::= integer in NR1 format.

- See Also**
- "Introduction to Root (:) Commands" on page 124
  - ":CHANnel<n>:PROTection" on page 206
  - ":OPEE (Operation Status Enable Register)" on page 139
  - ":OVLenable (Overload Event Enable Register)" on page 145
  - "\*STB (Read Status Byte)" on page 115
  - "\*SRE (Service Request Enable)" on page 113

**:PRINt**

**C** (see [page 608](#))

**Command Syntax**

```
:PRINt [<options>]
```

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

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

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

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

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - ["Introduction to :HARDcopy Commands"](#) on page 258
  - [":HARDcopy:FACTors"](#) on page 261
  - [":HARDcopy:GRAYscale"](#) on page 540
  - [":DISPlay:DATA"](#) on page 228

## :RUN

**C** (see [page 608](#))

**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 152
  - [":STOP"](#) on page 154

**Example Code**

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

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

**:SERial**

**N** (see [page 608](#))

**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

**C** (see [page 608](#))

**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 150
  - "[:STOP](#)" on page 154



**:STATus**

**N** (see [page 608](#))

**Query Syntax** :STATus? <source>

<source> ::= {CHANnel<n> | FUNCTION | MATH}  
for the DSO models

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

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

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

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

**NOTE**

MATH is an alias for FUNCTION.

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

<value> ::= {1 | 0}

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 124
  - [":BLANK"](#) on page 132
  - [":CHANnel<n>:DISPlay"](#) on page 196
  - [":DIGital<d>:DISPlay"](#) on page 219
  - [":FUNCTION:DISPlay"](#) on page 243
  - [":POD<n>:DISPlay"](#) on page 366
  - [":VIEW"](#) on page 156

### **:STOP**

**C** (see [page 608](#))

**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 150
  - [":SINGLE"](#) on page 152

- Example Code**
- ["Example Code"](#) on page 150

**:TER (Trigger Event Register)**

**C** (see [page 608](#))

**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 113
  - ["\\*STB \(Read Status Byte\)"](#) on page 115

**:VIEW**

**N** (see [page 608](#))

**Command Syntax** :VIEW <source>

```
<source> ::= {CHANnel<n> | FUNction | MATH}
           for DSO models
```

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

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

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

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

**NOTE**

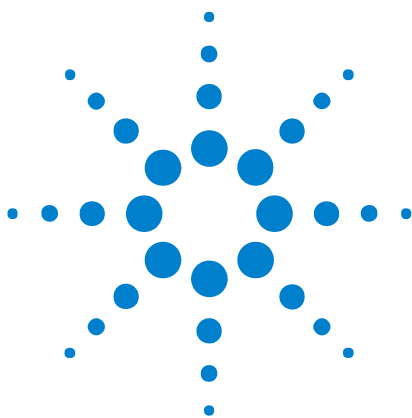
MATH is an alias for FUNction.

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on page 124
  - "[:BLANK](#)" on page 132
  - "[:CHANnel<n>:DISPlay](#)" on page 196
  - "[:DIGital<d>:DISPlay](#)" on page 219
  - "[:FUNction:DISPlay](#)" on page 243
  - "[:POD<n>:DISPlay](#)" on page 366
  - "[:STATus](#)" on page 153

**Example Code**

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

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617



## 7 :ACQUIRE Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQUIRE Commands" on page 157.

**Table 43** :ACQUIRE Commands Summary

Command	Query	Options and Query Returns
:ACQUIRE:COMPLETE <complete> (see page 159)	:ACQUIRE:COMPLETE? (see page 159)	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNT <count> (see page 160)	:ACQUIRE:COUNT? (see page 160)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQUIRE:MODE <mode> (see page 161)	:ACQUIRE:MODE? (see page 161)	<mode> ::= {RTIME   SEGMENTED}
n/a	:ACQUIRE:POINTS? (see page 162)	<# points> ::= an integer in NR1 format
:ACQUIRE:SEGMENTED:ANALYZE (see page 163)	n/a	n/a (with Option SGM)
:ACQUIRE:SEGMENTED:COUNT <count> (see page 164)	:ACQUIRE:SEGMENTED:COUNT? (see page 164)	<count> ::= an integer from 2 to 25 in NR1 format (with Option SGM)
:ACQUIRE:SEGMENTED:INDEX <index> (see page 165)	:ACQUIRE:SEGMENTED:INDEX? (see page 165)	<index> ::= an integer from 1 to 25 in NR1 format (with Option SGM)
n/a	:ACQUIRE:SRATE? (see page 168)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQUIRE:TYPE <type> (see page 169)	:ACQUIRE:TYPE? (see page 169)	<type> ::= {NORMAL   AVERAGE   HRESOLUTION   PEAK}

### Introduction to :ACQUIRE Commands

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

Normal



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

### Averaging

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

### High-Resolution

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

### Peak Detect

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

### Reporting the Setup

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

### Return Format

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

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

**:ACquire:COMplete**

**C** (see [page 608](#))

**Command Syntax** :ACquire:COMplete <complete>

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

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

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

**Query Syntax** :ACquire:COMplete?

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

**Return Format** <completion\_criteria><NL>

<completion\_criteria> ::= 100; an integer in NR1 format

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 157
  - [":ACquire:TYPE"](#) on page 169
  - [":DIGitize"](#) on page 133
  - [":WAVEform:POINTs"](#) on page 472

**Example Code**

```
' 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"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:ACquire:COUNT**

**C** (see [page 608](#))

**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 157
  - [":ACquire:TYPE"](#) on page 169
  - [":DIGitize"](#) on page 133
  - [":WAVEform:COUNT"](#) on page 468



**:ACquire:MODE**

**C** (see [page 608](#))

**Command Syntax** :ACquire:MODE <mode>  
 <mode> ::= {RTIME | SEGmented}

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

- The :ACquire:MODE RTIME command sets the oscilloscope in real time mode.

**NOTE**

The obsolete command ACquire:TYPE:REALtime is functionally equivalent to sending ACquire:MODE RTIME; TYPE NORMAl.

- The :ACquire:MODE SEGmented command sets the oscilloscope in segmented memory mode.

**Query Syntax** :ACquire:MODE?

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

**Return Format** <mode\_argument><NL>  
 <mode\_argument> ::= {RTIM | SEGM}

- See Also**
- "[Introduction to :ACquire Commands](#)" on page 157
  - "[:ACquire:TYPE](#)" on page 169

## :ACquire:POINts

**C** (see [page 608](#))

**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 157
  - [":DIGitize"](#) on page 133
  - [":WAVEform:POINts"](#) on page 472

**:ACquire:SEGmented:ANALyze****N** (see [page 608](#))**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 161
  - [":ACquire:SEGmented:COUNT"](#) on page 164
  - ["Introduction to :ACquire Commands"](#) on page 157

**:ACquire:SEGmented:COUNT**

**N** (see [page 608](#))

**Command Syntax** :ACquire:SEGmented:COUNT <count>  
 <count> ::= an integer from 2 to 25 (w/100K memory) in NR1 format

**NOTE**

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

The :ACquire:SEGmented:COUNT command sets the number of memory segments to acquire.

The segmented memory acquisition mode is enabled with the :ACquire:MODE command, and data is acquired using the :DIGitize, :SINGLE, or :RUN commands. The number of memory segments in the current acquisition is returned by the :WAVEform:SEGmented:COUNT? query.

The maximum number of segments may be limited by the memory depth of your oscilloscope. For example, an oscilloscope with 100K memory allows a maximum of 25 segments.

**Query Syntax** :ACquire:SEGmented:COUNT?

The :ACquire:SEGmented:COUNT? query returns the current count setting.

**Return Format** <count><NL>  
 <count> ::= an integer from 2 to 25 (w/100K memory) in NR1 format

- See Also**
- [":ACquire:MODE"](#) on page 161
  - [":DIGitize"](#) on page 133
  - [":SINGLE"](#) on page 152
  - [":RUN"](#) on page 150
  - [":WAVEform:SEGmented:COUNT"](#) on page 479
  - [":ACquire:SEGmented:ANALyze"](#) on page 163
  - ["Introduction to :ACquire Commands"](#) on page 157

**Example Code** • ["Example Code"](#) on page 165

**:ACQUIRE:SEGMENTED:INDEX**

**N** (see [page 608](#))

**Command Syntax** :ACQUIRE:SEGMENTED:INDEX <index>  
 <index> ::= an integer from 1 to 25 (w/100K memory) in NR1 format

**NOTE**

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

The :ACQUIRE:SEGMENTED:INDEX command sets the index into the memory segments that have been acquired.

The segmented memory acquisition mode is enabled with the :ACQUIRE:MODE command. The number of segments to acquire is set using the :ACQUIRE:SEGMENTED:COUNT command, and data is acquired using the :DIGITIZE, :SINGLE, or :RUN commands. The number of memory segments that have been acquired is returned by the :WAVEFORM:SEGMENTED:COUNT? query. The time tag of the currently indexed memory segment is returned by the :WAVEFORM:SEGMENTED:TTAG? query.

The maximum number of segments may be limited by the memory depth of your oscilloscope. For example, an oscilloscope with 100K memory allows a maximum of 25 segments.

**Query Syntax** :ACQUIRE:SEGMENTED:INDEX?

The :ACQUIRE:SEGMENTED:INDEX? query returns the current segmented memory index setting.

**Return Format** <index><NL>  
 <index> ::= an integer from 1 to 25 (w/100K memory) in NR1 format

- See Also**
- [":ACQUIRE:MODE"](#) on page 161
  - [":ACQUIRE:SEGMENTED:COUNT"](#) on page 164
  - [":DIGITIZE"](#) on page 133
  - [":SINGLE"](#) on page 152
  - [":RUN"](#) on page 150
  - [":WAVEFORM:SEGMENTED:COUNT"](#) on page 479
  - [":WAVEFORM:SEGMENTED:TTAG"](#) on page 480
  - [":ACQUIRE:SEGMENTED:ANALYZE"](#) on page 163
  - ["Introduction to :ACQUIRE COMMANDS"](#) on page 157

**Example Code** ' Segmented memory commands example.  
 ' -----

```

Option Explicit

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

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = _
        myMgr.Open("USB0::0x0957::0x17A6::US50210029::0::INSTR")
    myScope.IO.Clear ' Clear the interface.

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

    ' Set the number of segments to 25.
    myScope.WriteString ":ACquire:SEGmented:COUNT 25"
    myScope.WriteString ":ACquire:SEGmented:COUNT?"
    strQueryResult = myScope.ReadString
    Debug.Print "Acquisition memory segments: " + strQueryResult

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

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

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

    Dim lngSegments As Long
    lngSegments = varQueryResult

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

```

```

For lngI = lngSegments To 1 Step -1

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

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

Next lngI

Exit Sub

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

End Sub

```

## :ACquire:SRATe

**N** (see [page 608](#))

**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 157
  - [":ACquire:POINts"](#) on page 162



**:ACquire:TYPE**

**C** (see page 608)

**Command Syntax** :ACquire:TYPE <type>

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

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

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

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

- **HRESolution** – sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

For example, if the digitizer samples at 200 MSa/s, but the effective sample rate is 1 MSa/s (because of a slower sweep speed), only 1 out of every 200 samples needs to be stored. Instead of storing one sample (and throwing others away), the 200 samples are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.

- **PEAK** – sets the oscilloscope in the peak detect mode. In this mode, :ACquire:COUNT has no meaning.

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

**NOTE**

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

**Query Syntax** :ACquire:TYPE?

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

**Return Format** <acq\_type><NL>

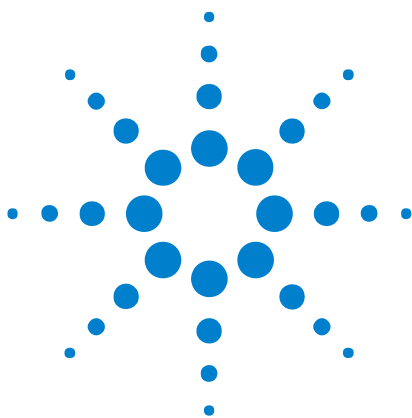
<acq\_type> ::= {NORM | AVER | HRES | PEAK}

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 157
  - [":ACquire:COUNT"](#) on page 160
  - [":ACquire:MODE"](#) on page 161
  - [":DIGitize"](#) on page 133
  - [":WAVEform:FORMat"](#) on page 471
  - [":WAVEform:TYPE"](#) on page 486
  - [":WAVEform:PREamble"](#) on page 476

**Example Code**

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

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617



## 8 :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See "[Introduction to :BUS<n> Commands](#)" on page 172.

**Table 44** :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0   OFF}   {1   ON}} (see <a href="#">page 173</a> )	:BUS<n>:BIT<m>? (see <a href="#">page 173</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0   OFF}   {1   ON}} (see <a href="#">page 174</a> )	:BUS<n>:BITS? (see <a href="#">page 174</a> )	<channel_list>, {0   1} <channel_list> ::= (@<m>, <m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-7; an integer in NR1 format
:BUS<n>:CLear (see <a href="#">page 176</a> )	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 177</a> )	:BUS<n>:DISPlay? (see <a href="#">page 177</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format



**Table 44** :BUS<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see page 178)	:BUS<n>:LABel? (see page 178)	<string> ::= quoted ASCII string up to 10 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 179)	:BUS<n>:MASK? (see page 179)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

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

<n> ::= {1 | 2}

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

**NOTE**

These commands are only valid for the MSO models.

**Reporting the Setup**

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

**Return Format**

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

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

**:BUS<n>:BIT<m>**

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:BIT<m> <display>

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

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

<m> ::= An integer, 0,...,7, is attached as a suffix to BIT and defines the digital channel that is affected by the command.

The :BUS<n>:BIT<m> command includes or excludes the selected bit as part of the definition for the selected bus. If the parameter is a 1 (ON), the bit is included in the definition. If the parameter is a 0 (OFF), the bit is excluded from the definition. *Note:* BIT0-7 correspond to DIGital0-7.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :BUS<n>:BIT<m>?

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

**Return Format** <display><NL>

<display> ::= {0 | 1}

- See Also**
- "[Introduction to :BUS<n> Commands](#)" on page 172
  - "[:BUS<n>:BITS](#)" on page 174
  - "[:BUS<n>:CLEar](#)" on page 176
  - "[:BUS<n>:DISPlay](#)" on page 177
  - "[:BUS<n>:LABel](#)" on page 178
  - "[:BUS<n>:MASK](#)" on page 179

**Example Code**

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

**:BUS<n>:BITS**

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:BITS <channel\_list>, <display>

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

<m> ::= An integer, 0,...,7, defines a digital channel affected by the command.

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

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :BUS<n>:BITS?

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

**Return Format** <channel\_list>, <display><NL>

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

<display> ::= {0 | 1}

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 172
  - [":BUS<n>:BIT<m>"](#) on page 173
  - [":BUS<n>:CLEar"](#) on page 176
  - [":BUS<n>:DISPlay"](#) on page 177
  - [":BUS<n>:LABel"](#) on page 178
  - [":BUS<n>:MASK"](#) on page 179

**Example Code**

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

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

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

```
' Include digital channels 1 through 3, 5, and 7 in bus 1:  
myScope.WriteString ":BUS1:BITS (@1:3,5,7), ON"
```

## :BUS<n>:CLEAr

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:CLEAr

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

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

**NOTE**

This command is only valid for the MSO models.

- 
- See Also**
- "[Introduction to :BUS<n> Commands](#)" on page 172
  - "[:BUS<n>:BIT<m>](#)" on page 173
  - "[:BUS<n>:BITS](#)" on page 174
  - "[:BUS<n>:DISPlay](#)" on page 177
  - "[:BUS<n>:LABel](#)" on page 178
  - "[:BUS<n>:MASK](#)" on page 179



**:BUS<n>:DISPlay**

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:DISplay <value>  
 <value> ::= {{1 | ON} | {0 | OFF}}

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :BUS<n>:DISPlay?

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

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

- See Also**
- "[Introduction to :BUS<n> Commands](#)" on page 172
  - "[:BUS<n>:BIT<m>](#)" on page 173
  - "[:BUS<n>:BITS](#)" on page 174
  - "[:BUS<n>:CLEar](#)" on page 176
  - "[:BUS<n>:LABel](#)" on page 178
  - "[:BUS<n>:MASK](#)" on page 179

**:BUS<n>:LABel**

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:LABel <quoted\_string>

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

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

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

**NOTE**

This command is only valid for the MSO models.

**NOTE**

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

**Query Syntax** :BUS<n>:LABel?

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

**Return Format** <quoted\_string><NL>

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

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 172
  - [":BUS<n>:BIT<m>"](#) on page 173
  - [":BUS<n>:BITS"](#) on page 174
  - [":BUS<n>:CLEar"](#) on page 176
  - [":BUS<n>:DISPlay"](#) on page 177
  - [":BUS<n>:MASK"](#) on page 179
  - [":CHANnel<n>:LABel"](#) on page 199
  - [":DISPlay:LABList"](#) on page 230
  - [":DIGital<d>:LABel"](#) on page 220

**Example Code**

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

**:BUS<n>:MASK**

**N** (see [page 608](#))

**Command Syntax** :BUS<n>:MASK <mask>

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

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

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

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

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

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

**NOTE**

This command is only valid for the MSO models.

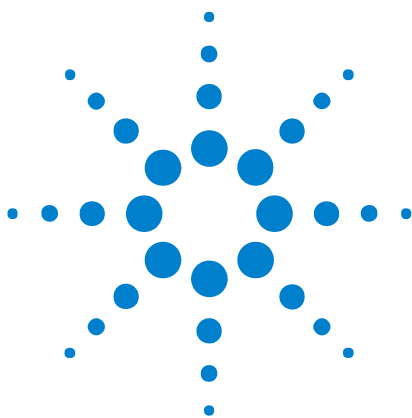
**Query Syntax** :BUS<n>:MASK?

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

**Return Format** <mask><NL> in decimal format

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 172
  - [":BUS<n>:BIT<m>"](#) on page 173
  - [":BUS<n>:BITS"](#) on page 174
  - [":BUS<n>:CLEar"](#) on page 176
  - [":BUS<n>:DISPlay"](#) on page 177
  - [":BUS<n>:LABel"](#) on page 178

## **8 :BUS<n> Commands**



## 9 :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "[Introduction to :CALibrate Commands](#)" on page 181.

**Table 45** :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see <a href="#">page 183</a> )	<return value> ::= <year>,<month>,<day>; all in NR1 format
:CALibrate:LABel <string> (see <a href="#">page 184</a> )	:CALibrate:LABel? (see <a href="#">page 184</a> )	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:OUTPut <signal> (see <a href="#">page 185</a> )	:CALibrate:OUTPut? (see <a href="#">page 185</a> )	<signal> ::= {TRIGgers   MASK   WAVEgen}
n/a	:CALibrate:PROTected? (see <a href="#">page 186</a> )	{PROTected   UNPRotected}
:CALibrate:START (see <a href="#">page 187</a> )	n/a	n/a
n/a	:CALibrate:STATus? (see <a href="#">page 188</a> )	<return value> ::= <status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:TEMPerature? (see <a href="#">page 189</a> )	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see <a href="#">page 190</a> )	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

### Introduction to :CALibrate Commands

The CALibrate subsystem provides utility commands for:



## 9 :CALibrate Commands

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

## :CALibrate:DATE

**N** (see [page 608](#))

**Query Syntax** :CALibrate:DATE?

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

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

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

**See Also** • ["Introduction to :CALibrate Commands"](#) on page 181

## :CALibrate:LABel

**N** (see [page 608](#))

**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 181



**:CALibrate:OUTPut**

**N** (see [page 608](#))

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

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

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

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

Waveform Type	Sync Signal Characteristics
SINusoid, RAMP, PULSe	The Sync signal is a square waveform with a 50% duty cycle.
SQUare	The Sync signal is a square waveform with the same duty cycle as the main output.
DC	N/A
NOISe	N/A

The Sync signal is a TTL "high" when the waveform's output is positive, relative to zero volts (or the DC offset value). The Sync signal is a TTL "low" when the output is negative, relative to zero volts (or the DC offset value).

**Query Syntax** :CALibrate:OUTPut?

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

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

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

- See Also**
- ["Introduction to :CALibrate Commands"](#) on page 181
  - [":WGEN:FUNCTION"](#) on page 498

## :CALibrate:PROTECTED

**N** (see [page 608](#))

**Query Syntax** :CALibrate:PROTECTED?

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

**Return Format** <switch><NL>

<switch> ::= {PROT | UNPR}

**See Also** • ["Introduction to :CALibrate Commands"](#) on page 181

## :CALibrate:START

**N** (see [page 608](#))

**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 181
  - "[:CALibrate:PROTECTED](#)" on page 186

## :CALibrate:STATus

**N** (see [page 608](#))

**Query Syntax** :CALibrate:STATus?

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

**Return Format** <return value><NL>  
<return value> ::= <status\_code>,<status\_string>  
<status\_code> ::= an integer status code  
<status\_string> ::= an ASCII status string

**See Also** • ["Introduction to :CALibrate Commands"](#) on page 181

**:CALibrate:TEMPerature**

**N** (see [page 608](#))

**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 181

## :CALibrate:TIME

**N** (see [page 608](#))

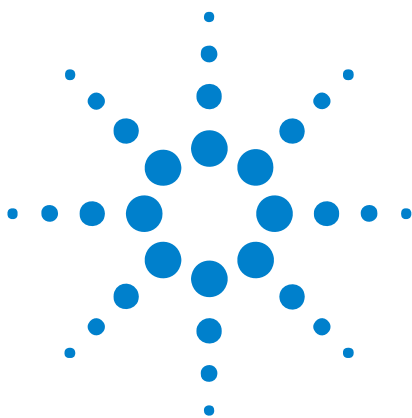
**Query Syntax** :CALibrate:TIME?

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 181



## 10 :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "[Introduction to :CHANnel<n> Commands](#)" on page 192.

**Table 46** :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit { {0   OFF}   {1   ON} } (see <a href="#">page 194</a> )	:CHANnel<n>:BWLimit? (see <a href="#">page 194</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:COUpling <coupling> (see <a href="#">page 195</a> )	:CHANnel<n>:COUpling? (see <a href="#">page 195</a> )	<coupling> ::= {AC   DC} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:DISPlay { {0   OFF}   {1   ON} } (see <a href="#">page 196</a> )	:CHANnel<n>:DISPlay? (see <a href="#">page 196</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:IMPedance <impedance> (see <a href="#">page 197</a> )	:CHANnel<n>:IMPedance ? (see <a href="#">page 197</a> )	<impedance> ::= ONEMeg <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:INVert { {0   OFF}   {1   ON} } (see <a href="#">page 198</a> )	:CHANnel<n>:INVert? (see <a href="#">page 198</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:LABel <string> (see <a href="#">page 199</a> )	:CHANnel<n>:LABel? (see <a href="#">page 199</a> )	<string> ::= any series of 10 or less ASCII characters enclosed in quotation marks <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see <a href="#">page 200</a> )	:CHANnel<n>:OFFSet? (see <a href="#">page 200</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 201</a> )	:CHANnel<n>:PROBe? (see <a href="#">page 201</a> )	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format



## 10 :CHANnel<n> Commands

**Table 46** :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param> (see <a href="#">page 202</a> )	:CHANnel<n>:PROBe:HEAD[:TYPE]? (see <a href="#">page 202</a> )	<head_param> ::= {SEND0   SEND6   SEND12   SEND20   DIFF0   DIFF6   DIFF12   DIFF20   NONE} <n> ::= 1 to (# analog channels) in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see <a href="#">page 203</a> )	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see <a href="#">page 204</a> )	:CHANnel<n>:PROBe:SKEW? (see <a href="#">page 204</a> )	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see <a href="#">page 205</a> )	:CHANnel<n>:PROBe:STYPe? (see <a href="#">page 205</a> )	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:PROTection (see <a href="#">page 206</a> )	:CHANnel<n>:PROTection? (see <a href="#">page 206</a> )	NORM <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see <a href="#">page 207</a> )	:CHANnel<n>:RANGe? (see <a href="#">page 207</a> )	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:SCALE <scale>[suffix] (see <a href="#">page 208</a> )	:CHANnel<n>:SCALE? (see <a href="#">page 208</a> )	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:UNITs <units> (see <a href="#">page 209</a> )	:CHANnel<n>:UNITs? (see <a href="#">page 209</a> )	<units> ::= {VOLT   AMPere} <n> ::= 1 to (# analog channels) in NR1 format
:CHANnel<n>:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 210</a> )	:CHANnel<n>:VERNier? (see <a href="#">page 210</a> )	{0   1} <n> ::= 1 to (# analog channels) in NR1 format

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



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

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

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

#### NOTE

The obsolete CHANnel subsystem is supported.

#### Reporting the Setup

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

#### Return Format

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

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

## :CHANnel<n>:BWLimit

**C** (see [page 608](#))

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

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

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

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

**Query Syntax** :CHANnel<n>:BWLimit?

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

**Return Format** <bwlimit><NL>

<bwlimit> ::= {1 | 0}

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

**:CHANnel<n>:COUpling**

**C** (see [page 608](#))

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

<coupling> ::= {AC | DC}

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

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

**Query Syntax** :CHANnel<n>:COUpling?

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

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

## :CHANnel<n>:DISPlay

**C** (see [page 608](#))

**Command Syntax** :CHANnel<n>:DISPlay <display value>  
<display value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= 1 to (# analog channels) in NR1 format

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

**Query Syntax** :CHANnel<n>:DISPlay?

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

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 192
  - [":VIEW"](#) on page 156
  - [":BLANK"](#) on page 132
  - [":STATus"](#) on page 153
  - [":POD<n>:DISPlay"](#) on page 366
  - [":DIGital<d>:DISPlay"](#) on page 219

**:CHANnel<n>:IMPedance**

**C** (see [page 608](#))

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

<impedance> ::= ONEMeg

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

The :CHANnel<n>:IMPedance command selects the input impedance setting for the specified analog channel. The only legal value for this command is ONEMeg (1 MΩ).

**Query Syntax** :CHANnel<n>:IMPedance?

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

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

<impedance value> ::= ONEM

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

## :CHANnel<n>:INVert

**N** (see [page 608](#))

**Command Syntax** :CHANnel<n>:INVert <invert value>  
<invert value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= 1 to (# analog channels) in NR1 format

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

**Query Syntax** :CHANnel<n>:INVert?

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

**:CHANnel<n>:LABel**

**N** (see [page 608](#))

**Command Syntax** :CHANnel<n>:LABel <string>  
 <string> ::= quoted ASCII string  
 <n> ::= 1 to (# analog channels) in NR1 format

**NOTE**

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

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

**Query Syntax** :CHANnel<n>:LABel?

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

**Return Format** <string><NL>  
 <string> ::= quoted ASCII string

- See Also**
- "Introduction to :CHANnel<n> Commands" on page 192
  - ":DISPlay:LABel" on page 229
  - ":DIGital<d>:LABel" on page 220
  - ":DISPlay:LABList" on page 230
  - ":BUS<n>:LABel" on page 178

**Example Code**

```
' LABEL - This command allows you to write a name (10 characters
' maximum) next to the channel number. It is not necessary, but
' can be useful for organizing the display.
myScope.WriteString ":CHANnel1:LABel " "CAL 1" " " ' Label ch1 "CAL 1".
myScope.WriteString ":CHANnel2:LABel " "CAL2" " " ' Label ch1 "CAL2".
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

## :CHANnel<n>:OFFSet

**C** (see [page 608](#))

**Command Syntax** :CHANnel<n>:OFFSet <offset> [<suffix>]  
<offset> ::= Vertical offset value in NR3 format  
<suffix> ::= {V | mV}  
<n> ::= 1 to (# analog channels) in NR1 format

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

**Query Syntax** :CHANnel<n>:OFFSet?

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

**Return Format** <offset><NL>  
<offset> ::= Vertical offset value in NR3 format

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 192
  - [":CHANnel<n>:RANGe"](#) on page 207
  - [":CHANnel<n>:SCALE"](#) on page 208
  - [":CHANnel<n>:PROBE"](#) on page 201



**:CHANnel<n>:PROBe**

**C** (see [page 608](#))

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

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

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

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

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

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

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

**Return Format** <attenuation><NL>

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 192
  - [":CHANnel<n>:RANGe"](#) on page 207
  - [":CHANnel<n>:SCALE"](#) on page 208
  - [":CHANnel<n>:OFFSet"](#) on page 200

**Example Code**

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

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

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

**C** (see page 608)

**Command Syntax****NOTE**

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

```
:CHANnel<n>:PROBe:HEAD[:TYPE] <head_param>
<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6
                  | DIFF12 | DIFF20 | NONE}
<n> ::= {1 | 2 | 3 | 4}
```

The :CHANnel<n>:PROBe:HEAD[:TYPE] command sets an analog channel probe head type and dB value. You can choose from:

- SEND0 – Single-ended, 0dB.
- SEND6 – Single-ended, 6dB.
- SEND12 – Single-ended, 12dB.
- SEND20 – Single-ended, 20dB.
- DIFF0 – Differential, 0dB.
- DIFF6 – Differential, 6dB.
- DIFF12 – Differential, 12dB.
- DIFF20 – Differential, 20dB.

**Query Syntax** :CHANnel<n>:PROBe:HEAD[:TYPE]?

The :CHANnel<n>:PROBe:HEAD[:TYPE]? query returns the current probe head type setting for the selected channel.

**Return Format** <head\_param><NL>

```
<head_param> ::= {SEND0 | SEND6 | SEND12 | SEND20 | DIFF0 | DIFF6
                  | DIFF12 | DIFF20 | NONE}
```

- See Also**
- "Introduction to :CHANnel<n> Commands" on page 192
  - ":CHANnel<n>:PROBe" on page 201
  - ":CHANnel<n>:PROBe:ID" on page 203
  - ":CHANnel<n>:PROBe:SKEW" on page 204
  - ":CHANnel<n>:PROBe:STYPe" on page 205

**:CHANnel<n>:PROBe:ID**

**C** (see [page 608](#))

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

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

The :CHANnel<n>:PROBe:ID? query returns the type of probe attached to the specified oscilloscope channel.

**Return Format** <probe id><NL>

<probe id> ::= unquoted ASCII string up to 11 characters

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

## :CHANnel<n>:PROBe:SKEW

**C** (see [page 608](#))

**Command Syntax** :CHANnel<n>:PROBe:SKEW <skew value>  
<skew value> ::= skew time in NR3 format  
<skew value> ::= -100 ns to +100 ns  
<n> ::= 1 to (# analog channels) in NR1 format

The :CHANnel<n>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. Each analog channel can be adjusted + or -100 ns for a total of 200 ns difference between channels. You can use the oscilloscope's probe skew control to remove cable-delay errors between channels.

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

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

**Return Format** <skew value><NL>  
<skew value> ::= skew value in NR3 format

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192

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

**C** (see [page 608](#))

**Command Syntax****NOTE**

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

```
:CHANnel<n>:PROBe:STYPe <signal type>
<signal type> ::= {DIFFerential | SINGle}
<n> ::= 1 to (# analog channels) in NR1 format
```

The :CHANnel<n>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

When single-ended is selected, the :CHANnel<n>:OFFset command changes the offset value of the probe amplifier. When differential is selected, the :CHANnel<n>:OFFset command changes the offset value of the channel amplifier.

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

The :CHANnel<n>:PROBe:STYPe? query returns the current probe signal type setting for the selected channel.

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

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 192
  - [":CHANnel<n>:OFFSet"](#) on page 200

## :CHANnel<n>:PROTection

**N** (see [page 608](#))

**Command Syntax** :CHANnel<n>:PROTection[:CLEar]

<n> ::= 1 to (# analog channels) in NR1 format | 4}

With the 2000 X-Series oscilloscopes, the analog channel input impedance is always 1 M $\Omega$ , so automatic overvoltage protection is not necessary (as it is for channels with 50 $\Omega$  input impedance). There are no protection settings to clear, so the :CHANnel<n>:PROTection[:CLEar] command does nothing.

**Query Syntax** :CHANnel<n>:PROTection?

The :CHANnel<n>:PROTection query always returns NORM (normal).

**Return Format** NORM<NL>

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 192
  - "[:CHANnel<n>:COUPling](#)" on page 195
  - "[:CHANnel<n>:PROBe](#)" on page 201

**:CHANnel<n>:RANGe**

**C** (see [page 608](#))

**Command Syntax** :CHANnel<n>:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

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

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, legal values for the range are from 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 192
  - "[:CHANnel<n>:SCALE](#)" on page 208
  - "[:CHANnel<n>:PROBE](#)" on page 201

**Example Code**

```
' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
' range value is 8 times the volts per division.
myScope.WriteString ":CHANnel1:RANGe 8" ' Set the vertical range to
8 volts.
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

## :CHANnel<n>:SCALe

**N** (see [page 608](#))

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

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

<suffix> ::= {V | mV}

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

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel. When using 1:1 probe attenuation, legal values for the scale are 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 192
  - "[:CHANnel<n>:RANGe](#)" on page 207
  - "[:CHANnel<n>:PROBe](#)" on page 201



**:CHANnel<n>:UNITs**

**N** (see [page 608](#))

**Command Syntax** :CHANnel<n>:UNITs <units>  
 <units> ::= {VOLT | AMPere}  
 <n> ::= 1 to (# analog channels) in NR1 format

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

**Query Syntax** :CHANnel<n>:UNITs?

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

**Return Format** <units><NL>  
 <units> ::= {VOLT | AMP}

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 192
  - [":CHANnel<n>:RANGe"](#) on page 207
  - [":CHANnel<n>:PROBe"](#) on page 201
  - [":EXTeRnal:UNITs"](#) on page 237

### :CHANnel<n>:VERNier

**N** (see [page 608](#))

**Command Syntax** :CHANnel<n>:VERNier <vernier value>  
<vernier value> ::= {{1 | ON} | {0 | OFF}}  
<n> ::= 1 to (# analog channels) in NR1 format

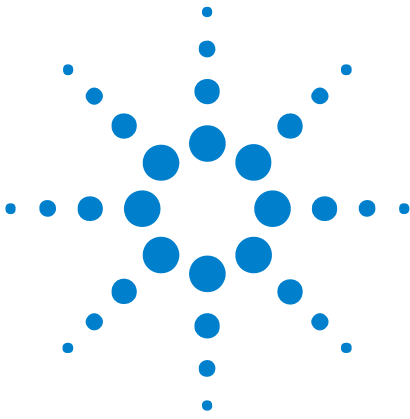
The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

**Query Syntax** :CHANnel<n>:VERNier?

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 192



# 11 :DEMO Commands

When the education kit is licensed (Option EDU), you can output demonstration signals on the oscilloscope's Demo 1 and Demo 2 terminals. See "[Introduction to :DEMO Commands](#)" on page 211.

**Table 47** :DEMO Commands Summary

Command	Query	Options and Query Returns
:DEMO:FUNCTION <signal> (see <a href="#">page 212</a> )	:DEMO:FUNCTION? (see <a href="#">page 213</a> )	<signal> ::= {SINusoid   NOISy   PHASe   RINGing   SINGle   AM   CLK   GLITCh   BURSt   MSO   RFBurst   LFSine}
:DEMO:FUNCTION:PHASe: PHASe <angle> (see <a href="#">page 214</a> )	:DEMO:FUNCTION:PHASe: PHASe? (see <a href="#">page 214</a> )	<angle> ::= angle in degrees from 0 to 360 in NR3 format
:DEMO:OUTPut {{0   OFF}   {1   ON}} (see <a href="#">page 215</a> )	:DEMO:OUTPut? (see <a href="#">page 215</a> )	{0   1}

### Introduction to :DEMO Commands

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

#### Reporting the Setup

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

#### Return Format

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

```
:DEMO:FUNC SIN;OUTP 0
```



**:DEMO:FUNCTION**

**N** (see page 608)

**Command Syntax** :DEMO:FUNCTION <signal>

<signal> ::= {SINusoid | NOISy | PHASe | RINGing | SINGle | AM | CLK  
| GLITch | BURSt | MSO | RFBurst | LFSine}

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

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
SINusoid	5 MHz sine wave @ ~ 6 Vpp, 0 V offset	Off
NOISy	1 kHz sine wave @ ~ 2.4 Vpp, 0.0 V offset, with ~ 0.5 Vpp of random noise added	Off
PHASe	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset	1 kHz sine wave @ 2.4 Vpp, 0.0 V offset , phase shifted by the amount entered using the <a href="#">":DEMO:FUNCTION:PHASe:PHASe"</a> on page 214 command
RINGing	500 kHz digital pulse @ ~ 3 Vpp, 1.5 V offset, and ~500 ns pulse width with ringing	Off
SINGle	~500 ns wide digital pulse with ringing @ ~ 3 Vpp, 1.5 V offset Press the front panel <b>Set Off Single-Shot</b> softkey to cause the selected single-shot signal to be output.	Off
AM	26 kHz sine wave, ~ 7 Vpp, 0 V offset	Amplitude modulated signal, ~ 3 Vpp, 0 V offset, with ~13 MHz carrier and sine envelope
CLK	500 kHz clock @ ~2 Vpp, 1 V offset, with infrequent glitch (1 glitch per 50,000 clocks)	Off
GLITch	Burst of 6 digital pulses (plus infrequent glitch) that occurs once every 80 $\mu$ s @ ~3.6 Vpp, ~1.8 V offset	Off
BURSt	Burst of digital pulses that occur every 50 $\mu$ s @ ~ 3.6 Vpp, ~1.5 V offset	Off

Demo Signal Function	Demo 1 Terminal	Demo 2 Terminal
MSO	3.1 kHz stair-step sine wave output of DAC @ ~1.5 Vpp, 0.75 V offset DAC input signals are internally routed to digital channels D0 through D7	~3.1 kHz sine wave filtered from DAC output @ ~ 600 mVpp, 300 mV offset
RFBurst	5-cycle burst of a 10 MHz amplitude modulated sine wave @ ~ 2.6 Vpp, 0 V offset occurring once every 4 ms	Off
LFSine	30 Hz sine wave @ ~2.7 Vpp, 0 V offset, with very narrow glitch near each positive peak	Off

**Query Syntax** :DEMO:FUNCTION?

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

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

```
<signal> ::= {SIN | NOIS | PHAS | RING | SING | AM | CLK | GLIT | BURS
              | MSO | RFB | LFS}
```

**See Also** • ["Introduction to :DEMO Commands"](#) on page 211

## :DEMO:FUNCTION:PHASe:PHASe

**N** (see [page 608](#))

**Command Syntax** :DEMO:FUNCTION:PHASe:PHASe <angle>

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

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

**Query Syntax** :DEMO:FUNCTION:PHASe:PHASe?

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

**Return Format** <angle><NL>

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

- See Also**
- "[Introduction to :DEMO Commands](#)" on page 211
  - "[:DEMO:FUNCTION](#)" on page 212

## :DEMO:OUTPut

**N** (see [page 608](#))

**Command Syntax** :DEMO:OUTPut <on\_off>

<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :DEMO:OUTPut command specifies whether the demo signal output is ON (1) or OFF (0).

**Query Syntax** :DEMO:OUTPut?

The :DEMO:OUTPut? query returns the current state of the demo signal output setting.

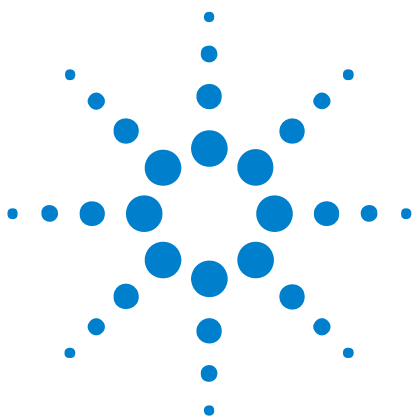
**Return Format** <on\_off><NL>

<on\_off> ::= {1 | 0}

- See Also**
- "[Introduction to :DEMO Commands](#)" on page 211
  - "[:DEMO:FUNCTION](#)" on page 212

## 11 :DEMO Commands





## 12 :DIGital<d> Commands

Control all oscilloscope functions associated with individual digital channels. See "[Introduction to :DIGital<d> Commands](#)" on page 217.

**Table 48** :DIGital<d> Commands Summary

Command	Query	Options and Query Returns
:DIGital<d>:DISPlay {0   OFF}   {1   ON} (see <a href="#">page 219</a> )	:DIGital<d>:DISPlay? (see <a href="#">page 219</a> )	<d> ::= 0 to (# digital channels - 1) in NR1 format {0   1}
:DIGital<d>:LABel <string> (see <a href="#">page 220</a> )	:DIGital<d>:LABel? (see <a href="#">page 220</a> )	<d> ::= 0 to (# digital channels - 1) in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:DIGital<d>:POSition <position> (see <a href="#">page 221</a> )	:DIGital<d>:POSition? (see <a href="#">page 221</a> )	<d> ::= 0 to (# digital channels - 1) in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small Returns -1 when there is no space to display the digital waveform.
:DIGital<d>:SIZE <value> (see <a href="#">page 222</a> )	:DIGital<d>:SIZE? (see <a href="#">page 222</a> )	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {SMALl   MEDium   LARGe}
:DIGital<d>:THReshold <value>[suffix] (see <a href="#">page 223</a> )	:DIGital<d>:THReshold? (see <a href="#">page 223</a> )	<d> ::= 0 to (# digital channels - 1) in NR1 format <value> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V   mV   uV}

**Introduction to  
:DIGital<d>  
Commands**

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



## 12 :DIGital<d> Commands

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

### NOTE

These commands are only valid for the MSO models.

---

#### Reporting the Setup

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

#### Return Format

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

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

**:DIGital<d>:DISPlay**

**N** (see [page 608](#))

**Command Syntax** :DIGital<d>:DISPlay <display>

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

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<d>:DISPlay?

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

**Return Format** <display><NL>

<display> ::= {0 | 1}

- See Also**
- "[Introduction to :DIGital<d> Commands](#)" on page 217
  - "[:POD<n>:DISPlay](#)" on page 366
  - "[:CHANnel<n>:DISPlay](#)" on page 196
  - "[:VIEW](#)" on page 156
  - "[:BLANK](#)" on page 132
  - "[:STATus](#)" on page 153

## :DIGital<d>:LABel

**N** (see [page 608](#))

**Command Syntax** :DIGital<d>:LABel <string>

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

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

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

**NOTE**

This command is only valid for the MSO models.

**NOTE**

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

**Query Syntax** :DIGital<d>:LABel?

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

**Return Format** <label string><NL>

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

- See Also**
- "[Introduction to :DIGital<d> Commands](#)" on page 217
  - "[:CHANnel<n>:LABel](#)" on page 199
  - "[:DISPlay:LABList](#)" on page 230
  - "[:BUS<n>:LABel](#)" on page 178

**:DIGital<d>:POSition**

**N** (see [page 608](#))

**Command Syntax** :DIGital<d>:POSition <position>  
 <d> ::= 0 to (# digital channels - 1) in NR1 format  
 <position> ::= integer in NR1 format.

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

The :DIGital<d>:POSition command sets the position of the specified channel.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<d>:POSition?

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

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

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

**See Also** • ["Introduction to :DIGital<d> Commands"](#) on page 217

**:DIGital<d>:SIZE**

**N** (see [page 608](#))

**Command Syntax** :DIGital<d>:SIZE <value>

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

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<d>:SIZE?

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

**Return Format** <size\_value><NL>

<size\_value> ::= {SMAL | MED | LARG}

- See Also**
- "[Introduction to :DIGital<d> Commands](#)" on page 217
  - "[:POD<n>:SIZE](#)" on page 367
  - "[:DIGital<d>:POSition](#)" on page 221

**:DIGital<d>:THReshold**

**N** (see [page 608](#))

**Command Syntax** :DIGital<d>:THReshold <value>

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

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

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

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

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

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

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<d>:THReshold?

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

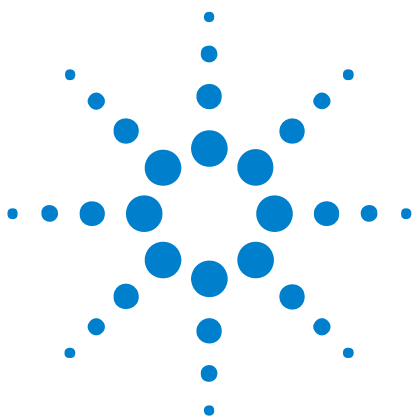
**Return Format** <value><NL>

<value> ::= threshold value in NR3 format

- See Also**
- "[Introduction to :DIGital<d> Commands](#)" on page 217
  - "[:POD<n>:THReshold](#)" on page 368
  - "[:TRIGger\[:EDGE\]:LEVel](#)" on page 435

## 12 :DIGital<d> Commands





## 13 :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "[Introduction to :DISPlay Commands](#)" on page 225.

**Table 49** :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLear (see <a href="#">page 227</a> )	n/a	n/a
n/a	:DISPlay:DATA? [ <a href="#">format</a> ][,][ <a href="#">palette</a> ] (see <a href="#">page 228</a> )	<format> ::= {BMP   BMP8bit   PNG} <palette> ::= {COLor   GRAYscale} <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0   OFF}   {1   ON}} (see <a href="#">page 229</a> )	:DISPlay:LABel? (see <a href="#">page 229</a> )	{0   1}
:DISPlay:LABList <binary block> (see <a href="#">page 230</a> )	:DISPlay:LABList? (see <a href="#">page 230</a> )	<binary block> ::= an ordered list of up to 75 labels, each 10 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see <a href="#">page 231</a> )	:DISPlay:PERsistence? (see <a href="#">page 231</a> )	<value> ::= {MINimum   INFinite   <time>} <time> ::= seconds in in NR3 format from 100E-3 to 60E0
:DISPlay:VECTors {1   ON} (see <a href="#">page 232</a> )	:DISPlay:VECTors? (see <a href="#">page 232</a> )	1

### Introduction to :DISPlay Commands

The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.
- Set waveform persistence.
- Specify labels.



## 13 :DISPlay Commands

- Save and Recall display data.

### Reporting the Setup

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

### Return Format

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

```
:DISP:LAB 0;CONN 1;PERS MIN
```

## :DISPlay:CLEar

**N** (see [page 608](#))

**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 225

**:DISPlay:DATA**

**N** (see [page 608](#))

**Query Syntax** :DISPlay:DATA? [<format>][,][<palette>]

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

<palette> ::= {COLor | GRAYscale}

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

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

Screen image data is returned in the IEEE-488.2 # binary block data format.

**Return Format** <display data><NL>

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 225
  - [":HARDcopy:INKSaver"](#) on page 263
  - [":PRINT"](#) on page 149
  - ["\\*RCL \(Recall\)"](#) on page 108
  - ["\\*SAV \(Save\)"](#) on page 112
  - [":VIEW"](#) on page 156

**Example Code**

```
' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPlay:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPlay:DATA? BMP, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1 ' Open file f
or output.
Put #1, , byteData ' Write data.
Close #1 ' Close file.
myScope.IO.Timeout = 5000
```

See complete example programs at: [Chapter 34, "Programming Examples,"](#) starting on page 617

**:DISPlay:LABel**

**N** (see [page 608](#))

**Command Syntax** :DISPlay:LABel <value>  
 <value> ::= {{1 | ON} | {0 | OFF}}

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

**Query Syntax** :DISPlay:LABel?

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

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

- See Also**
- "[Introduction to :DISPlay Commands](#)" on page 225
  - "[:CHANnel<n>:LABel](#)" on page 199

**Example Code**

```
' DISP_LABEL
' - Turns label names ON or OFF on the analyzer display.
myScope.WriteString ":DISPlay:LABel ON" ' Turn on labels.
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

## :DISPlay:LABList

**N** (see [page 608](#))

**Command Syntax** :DISPlay:LABList <binary block data>

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

The :DISPlay:LABList command adds labels to the label list. Labels are added in alphabetical order.

### NOTE

Labels that begin with the same alphabetic base string followed by decimal digits are considered duplicate labels. Duplicate labels are not added to the label list. For example, if label "A0" is in the list and you try to add a new label called "A123456789", the new label is not added.

**Query Syntax** :DISPlay:LABList?

The :DISPlay:LABList? query returns the label list.

**Return Format** <binary block><NL>

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 225
  - [":DISPlay:LABel"](#) on page 229
  - [":CHANnel<n>:LABel"](#) on page 199
  - [":DIGital<d>:LABel"](#) on page 220
  - [":BUS<n>:LABel"](#) on page 178

**:DISPlay:PERsistence**

**N** (see [page 608](#))

**Command Syntax** :DISPlay:PERsistence <value>  
 <value> ::= {MINimum | INFinite | <time>}  
 <time> ::= seconds in in NR3 format from 100E-3 to 60E0

The :DISPlay:PERsistence command specifies the persistence setting:

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

Use the :DISPlay:CLEar command to erase points stored by persistence.

**Query Syntax** :DISPlay:PERsistence?

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

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 225
  - [":DISPlay:CLEar"](#) on page 227

## :DISPlay:VECTors

**N** (see [page 608](#))

**Command Syntax** :DISPlay:VECTors <vectors>  
<vectors> ::= {1 | ON}

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

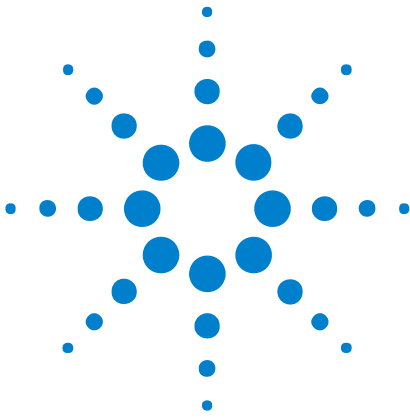
**Query Syntax** :DISPlay:VECTors?

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

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

**See Also** • ["Introduction to :DISPlay Commands"](#) on page 225





## 14 :EXternal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXternal Trigger Commands" on page 233.

**Table 50** :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see page 234)	:EXternal:BWLimit? (see page 234)	<bwlimit> ::= {0   OFF}
:EXternal:PROBe <attenuation> (see page 235)	:EXternal:PROBe? (see page 235)	<attenuation> ::= probe attenuation ratio in NR3 format
:EXternal:RANGe <range>[<suffix>] (see page 236)	:EXternal:RANGe? (see page 236)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXternal:UNITs <units> (see page 237)	:EXternal:UNITs? (see page 237)	<units> ::= {VOLT   AMPere}

### Introduction to :EXternal Trigger Commands

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

#### Reporting the Setup

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

#### Return Format

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

```
:EXT:BWL 0;RANG +8E+00;UNIT VOLT;PROB +1.000E+00
```



### :EXternal:BWLimit

**C** (see [page 608](#))

**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 233
  - ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:HFReject"](#) on page 426

**:EXternal:PROBe**

**C** (see [page 608](#))

**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 233
  - [":EXternal:RANGe"](#) on page 236
  - ["Introduction to :TRIGger Commands"](#) on page 423
  - [":CHANnel<n>:PROBe"](#) on page 201

### :EXtErnal:RANGe

**C** (see [page 608](#))

**Command Syntax** :EXtErnal:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

The :EXtErnal:RANGe command is provided for product compatibility. When using 1:1 probe attenuation, the range can only be set to 8.0 V.

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

**Query Syntax** :EXtErnal:RANGe?

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

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

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

- See Also**
- ["Introduction to :EXtErnal Trigger Commands"](#) on page 233
  - [":EXtErnal:PROBe"](#) on page 235
  - ["Introduction to :TRIGger Commands"](#) on page 423

**:EXternal:UNITs**

**N** (see [page 608](#))

**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?

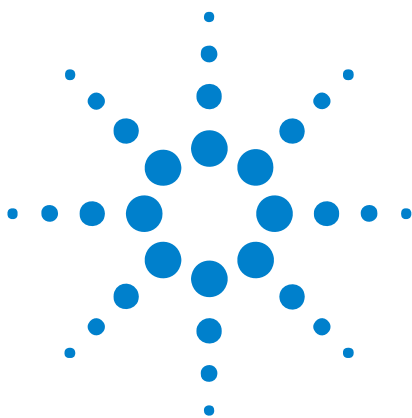
The :CHANnel<n>:UNITs? query returns the current units setting for the external trigger.

**Return Format** <units><NL>

<units> ::= {VOLT | AMP}

- See Also**
- ["Introduction to :EXternal Trigger Commands"](#) on page 233
  - ["Introduction to :TRIGger Commands"](#) on page 423
  - [":EXternal:RANGe"](#) on page 236
  - [":EXternal:PROBe"](#) on page 235
  - [":CHANnel<n>:UNITs"](#) on page 209

## 14 :EXtErnal Trigger Commands



## 15 :FUNCTION Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCTION Commands" on page 240.

**Table 51** :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTer <frequency> (see page 242)	:FUNCTION:CENTer? (see page 242)	<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 page 243)	:FUNCTION:DISPlay? (see page 243)	{0   1}
:FUNCTION:GOFT:OPERat ion <operation> (see page 244)	:FUNCTION:GOFT:OPERat ion? (see page 244)	<operation> ::= {ADD   SUBTract   MULTiply}
:FUNCTION:GOFT:SOURce 1 <source> (see page 245)	:FUNCTION:GOFT:SOURce 1? (see page 245)	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 246)	:FUNCTION:GOFT:SOURce 2? (see page 246)	<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 247)	:FUNCTION:OFFSet? (see page 247)	<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 248)	:FUNCTION:OPERation? (see page 248)	<operation> ::= {ADD   SUBTract   MULTiply   FFT}



**Table 51** :FUNCTION Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION:RANGE <range> (see page 249)	:FUNCTION:RANGE? (see page 249)	<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 FFT function is 8 to 800 dBV.
:FUNCTION:REFERENCE <level> (see page 250)	:FUNCTION:REFERENCE? (see page 250)	<level> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:SCALE <scale value>[<suffix>] (see page 251)	:FUNCTION:SCALE? (see page 251)	<scale value> ::= integer in NR1 format <suffix> ::= {V   dB}
:FUNCTION:SOURCE1 <source> (see page 252)	:FUNCTION:SOURCE1? (see page 252)	<source> ::= {CHANNEL<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models GOFT is only for FFT operation.
:FUNCTION:SOURCE2 <source> (see page 253)	:FUNCTION:SOURCE2? (see page 253)	<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 page 254)	:FUNCTION:SPAN? (see page 254)	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCTION:WINDOW <window> (see page 255)	:FUNCTION:WINDOW? (see page 255)	<window> ::= {RECTANGULAR   HANNING   FLATTOP   BHARRIS}

**Introduction to :FUNCTION Commands**

The FUNCTION subsystem controls the math functions in the oscilloscope. Add, subtract, multiply, 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 608](#))

**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 240
  - "[:FUNCTION:SPAN](#)" on page 254
  - "[:TIMEbase:RANGe](#)" on page 415
  - "[:TIMEbase:SCALE](#)" on page 417

**:FUNCTION:DISPlay**

**N** (see [page 608](#))

**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 240
  - [":VIEW"](#) on page 156
  - [":BLANK"](#) on page 132
  - [":STATus"](#) on page 153

**:FUNCTION:GOFT:OPERation**

**N** (see [page 608](#))

**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 function:

- 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 240
  - [":FUNCTION:GOFT:SOURce1"](#) on page 245
  - [":FUNCTION:GOFT:SOURce2"](#) on page 246
  - [":FUNCTION:SOURce1"](#) on page 252

**:FUNCTION:GOFT:SOURce1**

**N** (see [page 608](#))

**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 function.

**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 240
  - [":FUNCTION:GOFT:SOURce2"](#) on page 246
  - [":FUNCTION:GOFT:OPERation"](#) on page 244

**:FUNCTION:GOFT:SOURce2**

**N** (see [page 608](#))

**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 function.

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 240
  - "[:FUNCTION:GOFT:SOURce1](#)" on page 245
  - "[:FUNCTION:GOFT:OPERation](#)" on page 244

**:FUNCTION:OFFSet**

**N** (see [page 608](#))

**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 240
  - [":FUNCTION:RANGE"](#) on page 249
  - [":FUNCTION:REFerence"](#) on page 250
  - [":FUNCTION:SCALE"](#) on page 251

**:FUNCTION:OPERation**

**N** (see [page 608](#))

**Command Syntax** :FUNCTION:OPERation <operation>  
 <operation> ::= {ADD | SUBTRACT | MULTiply | FFT}

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

- ADD – Source1 + source2.
- SUBTRACT – Source1 - source2.
- MULTiply – Source1 \* source2.
- FFT – Fast Fourier Transform 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 FFT, 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 | FFT}

- See Also**
- "[Introduction to :FUNCTION Commands](#)" on page 240
  - "[:FUNCTION:SOURce1](#)" on page 252
  - "[:FUNCTION:SOURce2](#)" on page 253



## :FUNCTION:RANGe

**N** (see [page 608](#))

**Command Syntax** :FUNCTION:RANGe <range>

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

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

**Query Syntax** :FUNCTION:RANGe?

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

**Return Format** <range><NL>

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

- See Also**
- "[Introduction to :FUNCTION Commands](#)" on [page 240](#)
  - "[:FUNCTION:SCALE](#)" on [page 251](#)

**:FUNCTION:REFERENCE**

**N** (see [page 608](#))

**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 240
  - [":FUNCTION:OFFSET"](#) on page 247
  - [":FUNCTION:RANGE"](#) on page 249
  - [":FUNCTION:SCALE"](#) on page 251

**:FUNCTION:SCALE**

**N** (see [page 608](#))

**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 240
  - [":FUNCTION:RANGE"](#) on page 249

**:FUNCTION:SOURce1**

**N** (see [page 608](#))

**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 transform). 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 function. 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 240
  - "[:FUNCTION:OPERation](#)" on page 248
  - "[:FUNCTION:GOFT:OPERation](#)" on page 244
  - "[:FUNCTION:GOFT:SOURce1](#)" on page 245
  - "[:FUNCTION:GOFT:SOURce2](#)" on page 246

**:FUNCTION:SOURce2**

**N** (see [page 608](#))

**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 specifies the second source for math operations that have two sources (see the :FUNCTION:OPERation command), in other words, ADD, SUBTRact, or MULTiply. (The :FUNCTION:SOURce1 command specifies the first source.)

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.

The :FUNCTION:SOURce2 setting is not used when the :FUNCTION:OPERation is FFT (Fast Fourier Transform).

**Query Syntax** :FUNCTION:SOURce2?

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

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

- See Also**
- "[Introduction to :FUNCTION Commands](#)" on page 240
  - "[:FUNCTION:OPERation](#)" on page 248
  - "[:FUNCTION:SOURce1](#)" on page 252

**:FUNCTION:SPAN**

**N** (see [page 608](#))

**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 240
  - [":FUNCTION:CENTer"](#) on page 242
  - [":TIMEbase:RANGe"](#) on page 415
  - [":TIMEbase:SCALE"](#) on page 417

**:FUNCTION:WINDow**

**N** (see [page 608](#))

**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.

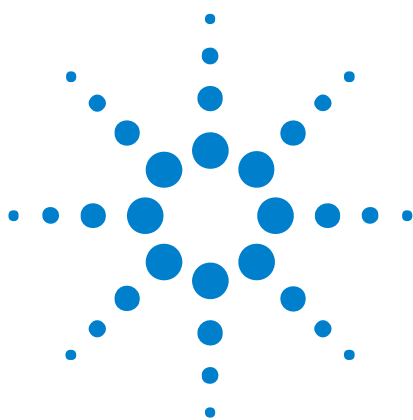
**Return Format** <window><NL>

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

**See Also** • ["Introduction to :FUNCTION Commands"](#) on page 240







## 16 :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "[Introduction to :HARDcopy Commands](#)" on page 258.

**Table 52** :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see <a href="#">page 259</a> )	:HARDcopy:AREA? (see <a href="#">page 259</a> )	<area> ::= SCreen
:HARDcopy:APRinter <active_printer> (see <a href="#">page 260</a> )	:HARDcopy:APRinter? (see <a href="#">page 260</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 261</a> )	:HARDcopy:FACTors? (see <a href="#">page 261</a> )	{0   1}
:HARDcopy:FFEed {{0   OFF}   {1   ON}} (see <a href="#">page 262</a> )	:HARDcopy:FFEed? (see <a href="#">page 262</a> )	{0   1}
:HARDcopy:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 263</a> )	:HARDcopy:INKSaver? (see <a href="#">page 263</a> )	{0   1}
:HARDcopy:LAYout <layout> (see <a href="#">page 264</a> )	:HARDcopy:LAYout? (see <a href="#">page 264</a> )	<layout> ::= {LANDscape   PORTRait}
:HARDcopy:NETWork:ADD Ress <address> (see <a href="#">page 265</a> )	:HARDcopy:NETWork:ADD Ress? (see <a href="#">page 265</a> )	<address> ::= quoted ASCII string
:HARDcopy:NETWork:APP Ly (see <a href="#">page 266</a> )	n/a	n/a
:HARDcopy:NETWork:DOM ain <domain> (see <a href="#">page 267</a> )	:HARDcopy:NETWork:DOM ain? (see <a href="#">page 267</a> )	<domain> ::= quoted ASCII string



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

Command	Query	Options and Query Returns
:HARDcopy:NETWork:PASSWORD <password> (see page 268)	n/a	<password> ::= quoted ASCII string
:HARDcopy:NETWork:SLOT <slot> (see page 269)	:HARDcopy:NETWork:SLOT? (see page 269)	<slot> ::= {NET0   NET1}
:HARDcopy:NETWork:USERNAME <username> (see page 270)	:HARDcopy:NETWork:USERNAME? (see page 270)	<username> ::= quoted ASCII string
:HARDcopy:PALETTE <palette> (see page 271)	:HARDcopy:PALETTE? (see page 271)	<palette> ::= {COLOR   GRAYscale   NONE}
n/a	:HARDcopy:PRINTER:LIST? (see page 272)	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y   N} <name> ::= name of printer
:HARDcopy:START (see page 273)	n/a	n/a

**Introduction to :HARDcopy Commands** The HARDcopy subsystem provides commands to set and query the selection of hardcopy device and formatting options such as inclusion of instrument settings (FACTors) and generation of formfeed (FFEed).

:HARDC is an acceptable short form for :HARDcopy.

**Reporting the Setup**

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

**Return Format**

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

```
:HARD:APR " ";AREA SCR;FACT 0;FFE 0;INKS 1;PAL NONE;LAY PORT
```

**:HARDcopy:AREA**

**N** (see [page 608](#))

**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 258
  - [":HARDcopy:START"](#) on page 273
  - [":HARDcopy:APRinter"](#) on page 260
  - [":HARDcopy:PRINter:LIST"](#) on page 272
  - [":HARDcopy:FACTors"](#) on page 261
  - [":HARDcopy:FFEed"](#) on page 262
  - [":HARDcopy:INKSaver"](#) on page 263
  - [":HARDcopy:LAYout"](#) on page 264
  - [":HARDcopy:PALette"](#) on page 271

## :HARDcopy:APRinter

**N** (see [page 608](#))

**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 258
  - "[:HARDcopy:PRINter:LIST](#)" on page 272
  - "[:HARDcopy:START](#)" on page 273

**:HARDcopy:FACTors**

**N** (see [page 608](#))

**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 258
  - "[:HARDcopy:START](#)" on page 273
  - "[:HARDcopy:FFeEd](#)" on page 262
  - "[:HARDcopy:INKSaver](#)" on page 263
  - "[:HARDcopy:LAYout](#)" on page 264
  - "[:HARDcopy:PALETTE](#)" on page 271

## :HARDcopy:FFeed

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:FFeed <ffeed>  
<ffeed> ::= {{OFF | 0} | {ON | 1}}

The HARDcopy:FFeed command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

**Query Syntax** :HARDcopy:FFeed?

The :HARDcopy:FFeed? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

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

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 258
  - "[:HARDcopy:START](#)" on page 273
  - "[:HARDcopy:FACTors](#)" on page 261
  - "[:HARDcopy:INKSaver](#)" on page 263
  - "[:HARDcopy:LAYout](#)" on page 264
  - "[:HARDcopy:PALETTE](#)" on page 271

**:HARDcopy:INKSaver**

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:INKSaver <value>  
 <value> ::= {{OFF | 0} | {ON | 1}}

The :HARDcopy:INKSaver command controls whether the graticule colors are inverted or not.

**Query Syntax** :HARDcopy:INKSaver?

The :HARDcopy:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

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

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 258
  - "[:HARDcopy:START](#)" on page 273
  - "[:HARDcopy:FACTors](#)" on page 261
  - "[:HARDcopy:FFEed](#)" on page 262
  - "[:HARDcopy:LAYout](#)" on page 264
  - "[:HARDcopy:PALETTE](#)" on page 271

## :HARDcopy:LAYout

**N** (see [page 608](#))

**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 258
  - [":HARDcopy:START"](#) on page 273
  - [":HARDcopy:FACTors"](#) on page 261
  - [":HARDcopy:PALETTE"](#) on page 271
  - [":HARDcopy:FFEed"](#) on page 262
  - [":HARDcopy:INKSaver"](#) on page 263



**:HARDcopy:NETWork:ADDRess**

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:ADDRess <address>

<address> ::= quoted ASCII string

The :HARDcopy:NETWork:ADDRess command sets the address for a network printer slot. The address is the server/computer name and the printer's share name in the \\server\share format.

The network printer slot is selected by the :HARDcopy:NETWork:SLOT command.

To apply the entered address, use the :HARDcopy:NETWork:APPLY command.

**Query Syntax** :HARDcopy:NETWork:ADDRess?

The :HARDcopy:NETWork:ADDRess? query returns the specified address for the currently selected network printer slot.

**Return Format** <address><NL>

<address> ::= quoted ASCII string

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 258
  - [":HARDcopy:NETWork:SLOT"](#) on page 269
  - [":HARDcopy:NETWork:APPLY"](#) on page 266
  - [":HARDcopy:NETWork:DOMain"](#) on page 267
  - [":HARDcopy:NETWork:USERname"](#) on page 270
  - [":HARDcopy:NETWork:PASSword"](#) on page 268

## :HARDcopy:NETWork:APPLy

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:APPLy

The :HARDcopy:NETWork:APPLy command applies the network printer settings and makes the printer connection.

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 258
  - "[:HARDcopy:NETWork:SLOT](#)" on page 269
  - "[:HARDcopy:NETWork:ADDRESS](#)" on page 265
  - "[:HARDcopy:NETWork:DOMAIN](#)" on page 267
  - "[:HARDcopy:NETWork:USERNAME](#)" on page 270
  - "[:HARDcopy:NETWork:PASSWORD](#)" on page 268

**:HARDcopy:NETWork:DOMain**

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:DOMain <domain>  
 <domain> ::= quoted ASCII string

The :HARDcopy:NETWork:DOMain command sets the Windows network domain name.

The domain name setting is a common setting for both network printer slots.

**Query Syntax** :HARDcopy:NETWork:DOMain?

The :HARDcopy:NETWork:DOMain? query returns the current Windows network domain name.

**Return Format** <domain><NL>  
 <domain> ::= quoted ASCII string

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 258
  - [":HARDcopy:NETWork:SLOT"](#) on page 269
  - [":HARDcopy:NETWork:APPLY"](#) on page 266
  - [":HARDcopy:NETWork:ADDRESS"](#) on page 265
  - [":HARDcopy:NETWork:USERNAME"](#) on page 270
  - [":HARDcopy:NETWork:PASSWORD"](#) on page 268

## :HARDcopy:NETWork:PASSword

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:PASSword <password>  
<password> ::= quoted ASCII string

The :HARDcopy:NETWork:PASSword command sets the password for the specified Windows network domain and user name.

The password setting is a common setting for both network printer slots.

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 258
  - [":HARDcopy:NETWork:USERname"](#) on page 270
  - [":HARDcopy:NETWork:DOMain"](#) on page 267
  - [":HARDcopy:NETWork:SLOT"](#) on page 269
  - [":HARDcopy:NETWork:APPLY"](#) on page 266
  - [":HARDcopy:NETWork:ADDRESS"](#) on page 265

**:HARDcopy:NETWork:SLOT**

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:SLOT <slot>

<slot> ::= {NET0 | NET1}

The :HARDcopy:NETWork:SLOT command selects the network printer slot used for the address and apply commands. There are two network printer slots to choose from.

**Query Syntax** :HARDcopy:NETWork:SLOT?

The :HARDcopy:NETWork:SLOT? query returns the currently selected network printer slot.

**Return Format** <slot><NL>

<slot> ::= {NET0 | NET1}

- See Also**
- "Introduction to :HARDcopy Commands" on page 258
  - ":HARDcopy:NETWork:APPLY" on page 266
  - ":HARDcopy:NETWork:ADDRESS" on page 265
  - ":HARDcopy:NETWork:DOMAIN" on page 267
  - ":HARDcopy:NETWork:USERNAME" on page 270
  - ":HARDcopy:NETWork:PASSWORD" on page 268

## :HARDcopy:NETWork:USERname

**N** (see [page 608](#))

**Command Syntax** :HARDcopy:NETWork:USERname <username>  
<username> ::= quoted ASCII string

The :HARDcopy:NETWork:USERname command sets the user name to use when connecting to the Windows network domain.

The user name setting is a common setting for both network printer slots.

**Query Syntax** :HARDcopy:NETWork:USERname?

The :HARDcopy:NETWork:USERname? query returns the currently set user name.

**Return Format** <username><NL>

<username> ::= quoted ASCII string

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 258
  - "[:HARDcopy:NETWork:DOMain](#)" on page 267
  - "[:HARDcopy:NETWork:PASSword](#)" on page 268
  - "[:HARDcopy:NETWork:SLOT](#)" on page 269
  - "[:HARDcopy:NETWork:APPLY](#)" on page 266
  - "[:HARDcopy:NETWork:ADDRESS](#)" on page 265

**:HARDcopy:PALETTE**

**N** (see [page 608](#))

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

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

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

The oscilloscope's print driver cannot print color images to color laser printers, so the COLOR option is not available when connected to laser printers.

**Query Syntax** :HARDcopy:PALETTE?

The :HARDcopy:PALETTE? query returns the selected hardcopy palette color.

**Return Format** <palette><NL>

<palette> ::= {COL | GRAY | NONE}

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 258
  - [":HARDcopy:START"](#) on page 273
  - [":HARDcopy:FACTors"](#) on page 261
  - [":HARDcopy:LAYout"](#) on page 264
  - [":HARDcopy:FFeEd"](#) on page 262
  - [":HARDcopy:INKSaver"](#) on page 263

## :HARDcopy:PRINter:LIST

**N** (see [page 608](#))

**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 258
  - [":HARDcopy:APRinter"](#) on page 260
  - [":HARDcopy:START"](#) on page 273



## :HARDcopy:STARt

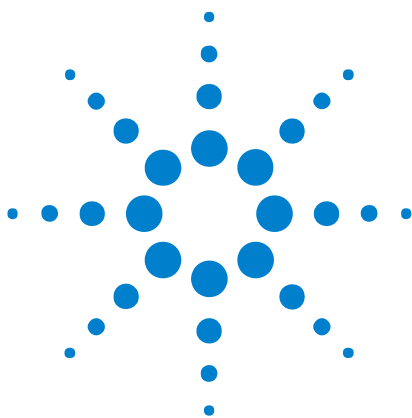
**N** (see [page 608](#))

**Command Syntax** :HARDcopy:STARt

The :HARDcopy:STARt command starts a print job.

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 258
  - "[:HARDcopy:APRinter](#)" on page 260
  - "[:HARDcopy:PRINter:LIST](#)" on page 272
  - "[:HARDcopy:FACTors](#)" on page 261
  - "[:HARDcopy:FFEed](#)" on page 262
  - "[:HARDcopy:INKSaver](#)" on page 263
  - "[:HARDcopy:LAYout](#)" on page 264
  - "[:HARDcopy:PALETTE](#)" on page 271





## 17 :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 276.

**Table 53** :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see <a href="#">page 277</a> )	:MARKer:MODE? (see <a href="#">page 277</a> )	<mode> ::= {OFF   MEASurement   MANual   WAVeform}
:MARKer:X1Position <position>[suffix] (see <a href="#">page 278</a> )	:MARKer:X1Position? (see <a href="#">page 278</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 279</a> )	:MARKer:X1Y1source? (see <a href="#">page 279</a> )	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see <a href="#">page 280</a> )	:MARKer:X2Position? (see <a href="#">page 280</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 281</a> )	:MARKer:X2Y2source? (see <a href="#">page 281</a> )	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see <a href="#">page 282</a> )	<return_value> ::= X cursors delta value in NR3 format



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

Command	Query	Options and Query Returns
:MARKer:Y1Position <position>[suffix] (see <a href="#">page 283</a> )	:MARKer:Y1Position? (see <a href="#">page 283</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 284</a> )	:MARKer:Y2Position? (see <a href="#">page 284</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 285</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 CHAN1;X2Y2 CHAN1;MODE MAN
```

**:MARKer:MODE**

**N** (see [page 608](#))

**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 276
  - "[:MARKer:X1Y1source](#)" on page 279
  - "[:MARKer:X2Y2source](#)" on page 281
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MARKer:X1Position](#)" on page 278
  - "[:MARKer:X2Position](#)" on page 280
  - "[:MARKer:Y1Position](#)" on page 283
  - "[:MARKer:Y2Position](#)" on page 284

**:MARKer:X1Position**

**N** (see [page 608](#))

**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 277](#)).
- 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 276](#)
  - [":MARKer:MODE"](#) on [page 277](#)
  - [":MARKer:X2Position"](#) on [page 280](#)
  - [":MARKer:X1Y1source"](#) on [page 279](#)
  - [":MARKer:X2Y2source"](#) on [page 281](#)
  - [":MEASure:TSTArt"](#) on [page 547](#)

**:MARKer:X1Y1source**

**N** (see [page 608](#))

**Command Syntax** :MARKer:X1Y1source <source>  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= {1 | 2}

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see [":MARKer:MODE"](#) on [page 277](#)):

- Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X1Y1) sets the source for the other (for example, X2Y2).

If the marker mode is currently WAVEform, the X1Y1 source can be set separate from the X2Y2 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNction, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

**NOTE**

MATH is an alias for FUNction. The query will return FUNC if the source is FUNction or MATH.

**Query Syntax** :MARKer:X1Y1source?

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format** <source><NL>  
 <source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}

- See Also**
- ["Introduction to :MARKer Commands"](#) on [page 276](#)
  - [":MARKer:MODE"](#) on [page 277](#)
  - [":MARKer:X2Y2source"](#) on [page 281](#)
  - [":MEASure:SOURce"](#) on [page 315](#)

**:MARKer:X2Position**

**N** (see [page 608](#))

**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 277](#)).
- 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 276](#)
  - [":MARKer:MODE"](#) on [page 277](#)
  - [":MARKer:X1Position"](#) on [page 278](#)
  - [":MARKer:X2Y2source"](#) on [page 281](#)
  - [":MEASure:TSTOp"](#) on [page 548](#)



**:MARKer:X2Y2source**

**N** (see [page 608](#))

**Command Syntax** :MARKer:X2Y2source <source>  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= {1 | 2}

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued.

If the marker mode is not currently WAVEform (see [":MARKer:MODE"](#) on [page 277](#)):

- Sending a :MARKer:X2Y2source command will put the cursors in the MANual mode.
- Setting the source for one pair of markers (for example, X2Y2) sets the source for the other (for example, X1Y1).

If the marker mode is currently WAVEform, the X2Y2 source can be set separate from the X1Y1 source.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNction, MATH, or WMEMory<r> will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

**NOTE**

MATH is an alias for FUNction. The query will return FUNC if the source is FUNction or MATH.

**Query Syntax** :MARKer:X2Y2source?

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format** <source><NL>  
 <source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}

- See Also**
- ["Introduction to :MARKer Commands"](#) on [page 276](#)
  - [":MARKer:MODE"](#) on [page 277](#)
  - [":MARKer:X1Y1source"](#) on [page 279](#)
  - [":MEASure:SOURce"](#) on [page 315](#)

**:MARKer:XDELta**

**N** (see [page 608](#))

**Query Syntax** :MARKer:XDELta?

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

Xdelta = (Value at X2 cursor) - (Value at X1 cursor)

**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 276
  - "[:MARKer:MODE](#)" on page 277
  - "[:MARKer:X1Position](#)" on page 278
  - "[:MARKer:X2Position](#)" on page 280
  - "[:MARKer:X1Y1source](#)" on page 279
  - "[:MARKer:X2Y2source](#)" on page 281

**:MARKer:Y1Position**

**N** (see [page 608](#))

**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 277](#)), 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 276](#)
  - "[:MARKer:MODE](#)" on [page 277](#)
  - "[:MARKer:X1Y1source](#)" on [page 279](#)
  - "[:MARKer:X2Y2source](#)" on [page 281](#)
  - "[:MARKer:Y2Position](#)" on [page 284](#)
  - "[:MEASure:VSTart](#)" on [page 553](#)

**:MARKer:Y2Position**

**N** (see [page 608](#))

**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 277](#)), 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 276](#)
  - "[:MARKer:MODE](#)" on [page 277](#)
  - "[:MARKer:X1Y1source](#)" on [page 279](#)
  - "[:MARKer:X2Y2source](#)" on [page 281](#)
  - "[:MARKer:Y1Position](#)" on [page 283](#)
  - "[:MEASure:VSTOP](#)" on [page 554](#)

**:MARKer:YDELta**

**N** (see [page 608](#))

**Query Syntax** :MARKer:YDELta?

The :MARKer:YDELta? query returns the value difference between the current Y1 and Y2 cursor positions.

$Ydelta = (\text{Value at Y2 cursor}) - (\text{Value at Y1 cursor})$

**NOTE**

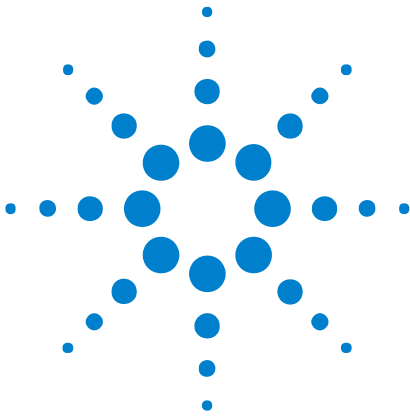
If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual or WAVeform to put the cursors in the front-panel Normal mode.

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

<value> ::= difference value in NR3 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 276
  - "[:MARKer:MODE](#)" on page 277
  - "[:MARKer:X1Y1source](#)" on page 279
  - "[:MARKer:X2Y2source](#)" on page 281
  - "[:MARKer:Y1Position](#)" on page 283
  - "[:MARKer:Y2Position](#)" on page 284





## 18 :MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 294.

**Table 54** :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:ALL (see <a href="#">page 296</a> )	n/a	n/a
:MEASure:CLear (see <a href="#">page 297</a> )	n/a	n/a
:MEASure:DEFine DELay, <delay spec> (see <a href="#">page 298</a> )	:MEASure:DEFine? DELay (see <a href="#">page 299</a> )	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+   -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see <a href="#">page 298</a> )	:MEASure:DEFine? THResholds (see <a href="#">page 299</a> )	<threshold spec> ::= {STANdard}   {<threshold mode>,<upper>,<middle>,<lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DELay [<source1> [,<source2>] (see <a href="#">page 301</a> )	:MEASure:DELay? [<source1> [,<source2>] (see <a href="#">page 301</a> )	<source1,2> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format



**Table 54** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DUTYcycle [<source>] (see page 303)	:MEASure:DUTYcycle? [<source>] (see page 303)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNction   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 304)	:MEASure:FALLtime? [<source>] (see page 304)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNction   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 305)	:MEASure:FREQuency? [<source>] (see page 305)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNction   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= frequency in Hertz in NR3 format



Table 54 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NWIDth [<source>] (see page 306)	:MEASure:NWIDth? [<source>] (see page 306)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 307)	:MEASure:OVERshoot? [<source>] (see page 307)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 309)	:MEASure:PERiod? [<source>] (see page 309)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 310)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 310)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format

**Table 54** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PREShoot [<source>] (see page 311)	:MEASure:PREShoot? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 312)	:MEASure:PWIDth? [<source>] (see page 312)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISetime [<source>] (see page 313)	:MEASure:RISetime? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SHOW {1   ON} (see page 314)	:MEASure:SHOW? (see page 314)	{1}

Table 54 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:SOURce <source1> [,<source2>] (see page 315)	:MEASure:SOURce? (see page 315)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>   EXTErnal} for DSO models <source1,2> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>   EXTErnal} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= {<source>   NONE}
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 317)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of the specified transition

**Table 54** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 319)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNctIon   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= time in seconds of specified voltage crossing in NR3 format
:MEASure:VAMplitude [<source>] (see page 321)	:MEASure:VAMplitude? [<source>] (see page 321)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<interval>][,][<source>] (see page 322)	:MEASure:VAverage? [<interval>][,][<source>] (see page 322)	<interval> ::= {CYCLE   DISPlay} <source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 323)	:MEASure:VBASe? [<source>] (see page 323)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

**Table 54** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 324)	:MEASure:VMAX? [<source>] (see page 324)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 325)	:MEASure:VMIN? [<source>] (see page 325)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 326)	:MEASure:VPP? [<source>] (see page 326)	<source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<interval>][,] [<type>][,] [<source>] (see page 327)	:MEASure:VRMS? [<interval>][,] [<type>][,] [<source>] (see page 327)	<interval> ::= {CYCLE   DISPLAY} <type> ::= {AC   DC} <source> ::= {CHANnel<n>   FUNctIon   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format

**Table 54** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:VTIME? <vtime>[,<source>] (see <a href="#">page 328</a> )	<vtime> ::= displayed time from trigger in seconds in NR3 format <source> ::= {CHANnel<n>   FUNCTION   MATH   WMEMory<r>} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   FUNCTION   MATH   WMEMory<r>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <return_value> ::= voltage at the specified time in NR3 format
:MEASure:VTOP [<source>] (see <a href="#">page 329</a> )	:MEASure:VTOP? [<source>] (see <a href="#">page 329</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:WINDow <type> (see <a href="#">page 330</a> )	:MEASure:WINDow? (see <a href="#">page 330</a> )	<type> ::= {MAIN   ZOOM   AUTO}

**Introduction to :MEASure Commands**

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

**Measurement Setup**

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

**Measurement Error**

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

### Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the zoomed (delayed) time base mode (:TIMEbase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the zoomed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNcTION source.

Not all measurements are available on the digital channels or FFT (Fast Fourier Transform).

### Reporting the Setup

Use the :MEASure? query to obtain setup information for the MEASure subsystem. (Currently, this is only :MEASure:SOURce.)

### Return Format

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

```
:MEAS:SOUR CHAN1,CHAN2;STAT ON
```

## :MEASure:ALL

**N** (see [page 608](#))

**Command Syntax** :MEASure:ALL

This command clears installs a Snapshot All measurement on the screen.

**See Also** • ["Introduction to :MEASure Commands"](#) on page 294



## :MEASure:CLEar

**N** (see [page 608](#))

**Command Syntax** :MEASure:CLEar

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

**See Also** • ["Introduction to :MEASure Commands"](#) on page 294

**:MEASure:DEFine**

**N** (see page 608)

**Command Syntax** :MEASure:DEFine <meas\_spec>

<meas\_spec> ::= {DElay | THResholds}

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DElay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DElay	THResholds
DUTYcycle		x
DElay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PREShoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

**:MEASure:DEFine** :MEASure:DEFine DElay,<delay\_spec>

**DElay Command Syntax**

<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 208:CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

**Query Syntax**

```
:MEASure:DEfine? <meas_spec>
```

```
<meas_spec> ::= {DElay | THResholds}
```

The :MEASure:DEfine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

**Return Format** for <meas\_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas\_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>
```

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and lower threshold percentage values between Vbase and Vtop in NR3 format.

for <meas\_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>
```

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and lower threshold voltages in NR3 format.

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:DELay"](#) on page 301
  - [":MEASure:SOURce"](#) on page 315
  - [":CHANnel<n>:RANGe"](#) on page 207
  - [":CHANnel<n>:SCALE"](#) on page 208
  - [":CHANnel<n>:PROBE"](#) on page 201
  - [":CHANnel<n>:UNITs"](#) on page 209

**:MEASure:DElay**

**N** (see page 608)

**Command Syntax** :MEASure:DElay [<source1>][,<source2>]  
 <source1>, <source2> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:DElay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

$$\text{delay} = t(\text{<edge spec 2>}) - t(\text{<edge spec 1>})$$

where the <edge spec> definitions are set by the :MEASure:DEFine command

**NOTE**

The :MEASure:DElay command and the front-panel delay measurement differ from the :MEASure:DElay? query.

The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DElay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

**Query Syntax** :MEASure:DElay? [<source1>][,<source2>]

The :MEASure:DElay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also in the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and

Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THResholds command.

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

<value> ::= floating-point number delay time in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:DEFine"](#) on page 298
  - [":MEASure:PHASe"](#) on page 310

**:MEASure:DUTYcycle**

**C** (see [page 608](#))

**Command Syntax** :MEASure:DUTYcycle [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

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

**NOTE**

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

**Query Syntax** :MEASure:DUTYcycle? [<source>]

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

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

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

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:PERiod"](#) on page 309
  - [":MEASure:PWIDth"](#) on page 312
  - [":MEASure:SOURce"](#) on page 315

- Example Code**
- ["Example Code"](#) on page 316

**:MEASure:FALLtime**

**C** (see [page 608](#))

**Command Syntax** :MEASure:FALLtime [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:FALLtime? [<source>]

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

$$\text{fall time} = \text{time at lower threshold} - \text{time at upper threshold}$$

**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 294
  - "[:MEASure:RISetime](#)" on page 313
  - "[:MEASure:SOURce](#)" on page 315



**:MEASure:FREQuency**

**C** (see [page 608](#))

**Command Syntax** :MEASure:FREQuency [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :MEASure:FREQuency command installs a screen measurement and starts a frequency measurement. If the optional source parameter is specified, the current source is modified.

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:FREQuency? [<source>]

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

**Return Format** <source><NL>  
 <source> ::= frequency in Hertz in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:PERiod"](#) on page 309

**Example Code** • ["Example Code"](#) on page 316

**:MEASure:NWIDth**

**C** (see [page 608](#))

**Command Syntax** :MEASure:NWIDth [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is not specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:NWIDth? [<source>]

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

**Return Format** <value><NL>  
 <value> ::= negative pulse width in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:PWIDth"](#) on page 312
  - [":MEASure:PERiod"](#) on page 309

**:MEASure:OVERshoot**

**C** (see [page 608](#))

**Command Syntax** :MEASure:OVERshoot [<source>]  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:OVERshoot? [<source>]

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{overshoot} = ((V_{\text{max}} - V_{\text{top}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

For a falling edge:

$$\text{overshoot} = ((V_{\text{base}} - V_{\text{min}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

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 294
  - [":MEASure:PREShoot"](#) on page 311
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324

## 18 :MEASure Commands

- ":MEASure:VTOP" on page 329
- ":MEASure:VBASe" on page 323
- ":MEASure:VMIN" on page 325

**:MEASure:PERiod**

**C** (see [page 608](#))

**Command Syntax** :MEASure:PERiod [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :MEASure:PERiod command installs a screen measurement and starts the period measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:PERiod? [<source>]

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

**Return Format** <value><NL>  
 <value> ::= waveform period in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:NWIDTH"](#) on page 306
  - [":MEASure:PWIDTh"](#) on page 312
  - [":MEASure:FREQuency"](#) on page 305

- Example Code**
- ["Example Code"](#) on page 316

**:MEASure:PHASe**

**N** (see [page 608](#))

**Command Syntax** :MEASure:PHASe [<source1>][,<source2>]  
 <source1>, <source2> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

**Query Syntax** :MEASure:PHASe? [<source1>][,<source2>]

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELAY for more detail on selecting the 2nd edge.

The phase is calculated as follows:

$$\text{phase} = (\text{delay} / \text{period of input 1}) \times 360$$

**Return Format** <value><NL>  
 <value> ::= the phase angle value in degrees in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:DELAY"](#) on page 301
  - [":MEASure:PERiod"](#) on page 309
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:PREShoot**

**C** (see [page 608](#))

**Command Syntax** :MEASure:PREShoot [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:PREShoot? [<source>]

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{preshoot} = ((V_{\text{min}} - V_{\text{base}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

For a falling edge:

$$\text{preshoot} = ((V_{\text{max}} - V_{\text{top}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

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 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VTOP"](#) on page 329
  - [":MEASure:VBASe"](#) on page 323

**:MEASure:PWIDth**

**C** (see [page 608](#))

**Command Syntax** :MEASure:PWIDth [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:PWIDth? [<source>]

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

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

<value> ::= width of positive pulse in seconds in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:NWIDth](#)" on page 306
  - "[:MEASure:PERiod](#)" on page 309



**:MEASure:RISetime**

**C** (see [page 608](#))

**Command Syntax** :MEASure: RISetime [<source>]  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure: RISetime? [<source>]

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

$$\text{rise time} = \text{time at upper threshold} - \text{time at lower threshold}$$

**Return Format** <value><NL>  
 <value> ::= rise time in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:FALLtime"](#) on page 304

## :MEASure:SHOW

**N** (see [page 608](#))

**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 294

**:MEASure:SOURce**

**C** (see [page 608](#))

**Command Syntax** :MEASure:SOURce <source1>[,<source2>]

```
<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNcTion
                        | MATH | WMEMory<r> | EXTErnal}
```

```
<digital channels> ::= DIGital<d> for the MSO models
```

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

```
<r> ::= 1-2 in NR1 format
```

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

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNcTion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

EXTErnal is only a valid source for the counter measurement (and <source1>).

**Query Syntax** :MEASure:SOURce?

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELAy and :MEASure:PHASe measurements.

**NOTE**

MATH is an alias for FUNcTion. The query will return FUNC if the source is FUNcTion or MATH.

**Return Format** <source1>,<source2><NL>

```
<source1>,<source2> ::= {<digital channels> | CHAN<n> | FUNC | WMWM<r>
                        | EXT | NONE}
```

- See Also:**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MARKer:MODE"](#) on page 277
  - [":MARKer:X1Y1source"](#) on page 279
  - [":MARKer:X2Y2source"](#) on page 281
  - [":MEASure:DELAy"](#) on page 301

- ":MEASure:PHASe" on page 310

**Example Code**

```
' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1" ' Source to measure.
myScope.WriteString ":MEASURE:FREQUENCY?" ' Query for frequency.
varQueryResult = myScope.ReadNumber ' Read frequency.
MsgBox "Frequency:" + vbCrLf _
      + FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASURE:DUTYCYCLE?" ' Query for duty cycle.
varQueryResult = myScope.ReadNumber ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf _
      + FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASURE:RISETIME?" ' Query for risetime.
varQueryResult = myScope.ReadNumber ' Read risetime.
MsgBox "Risetime:" + vbCrLf _
      + FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASURE:VPP?" ' Query for Pk to Pk voltage.
varQueryResult = myScope.ReadNumber ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASURE:VMAX?" ' Query for Vmax.
varQueryResult = myScope.ReadNumber ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

**:MEASure:TEDGe**

**N** (see page 608)

**Query Syntax** :MEASure:TEDGe? <slope><occurrence>[,<source>]

<slope> ::= direction of the waveform. A rising slope is indicated by a space or plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing from the left screen edge is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
| WMEMory<r>}

<digital channels> ::= DIGital<d> for the MSO models

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

<r> ::= 1-2 in NR1 format

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

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGe command:

Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel

Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see "[:MEASure:TEDGe Code](#)" on page 318.

If the optional source parameter is specified, the current source is modified.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

**Return Format**

<value><NL>

<value> ::= time in seconds of the specified transition in NR3 format

**:MEASure:TEDGe  
Code**

```
' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.
' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:TVALue"](#) on page 319
  - [":MEASure:VTIME"](#) on page 328

**:MEASure:TVALue**

**C** (see [page 608](#))

**Query Syntax** :MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross. The value can be volts or a math function value such as dB, Vs, or V/s.

<slope> ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMORY<r>}

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

<r> ::= 1-2 in NR1 format

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

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

## 18 :MEASure Commands

<value> ::= time in seconds of the specified value crossing in  
NR3 format

- See Also**
- "Introduction to :MEASure Commands" on page 294
  - ":MEASure:TEDGE" on page 317
  - ":MEASure:VTIME" on page 328



**:MEASure:VAMPlitude**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VAMPlitude [<source>]  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VAMPlitude? [<source>]

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

$$\text{vertical amplitude} = V_{\text{top}} - V_{\text{base}}$$

**Return Format** <value><NL>  
 <value> ::= the amplitude of the selected waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VBASe"](#) on page 323
  - [":MEASure:VTOP"](#) on page 329
  - [":MEASure:VPP"](#) on page 326

**:MEASure:VAverage**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VAverage [<interval>][,][<source>]  
 <interval> ::= {CYCLe | DISPlay}  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}  
 <n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VAverage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

**Query Syntax** :MEASure:VAverage? [<source>]

The :MEASure:VAverage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

**Return Format** <value><NL>  
 <value> ::= calculated average value in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MEASure:SOURce](#)" on page 315

**:MEASure:VBASe**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VBASe [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VBASe? [<source>]

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

**Return Format** <base\_voltage><NL>  
 <base\_voltage> ::= value at the base of the selected waveform in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:VTOP](#)" on page 329
  - "[:MEASure:VAMPLitude](#)" on page 321
  - "[:MEASure:VMIN](#)" on page 325

**:MEASure:VMAX**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VMAX [<source>]  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VMAX? [<source>]

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

**Return Format** <value><NL>  
 <value> ::= maximum vertical value of the selected waveform in  
 NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VPP"](#) on page 326
  - [":MEASure:VTOP"](#) on page 329

**:MEASure:VMIN**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VMIN [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMemory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VMIN? [<source>]

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

**Return Format** <value><NL>  
 <value> ::= minimum vertical value of the selected waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VBASe"](#) on page 323
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VPP"](#) on page 326

**:MEASure:VPP**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VPP [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VPP? [<source>]

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

$$V_{pp} = V_{max} - V_{min}$$

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 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VAMPLitude"](#) on page 321

**:MEASure:VRMS**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VRMS [<interval>][,][<type>][,][<source>]  
 <interval> ::= {CYCLe | DISPlay}  
 <type> ::= {AC | DC}  
 <source> ::= {CHANnel<n> | FUNction | MATH | WMEMory<r>}  
 <n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VRMS command installs a screen measurement and starts an RMS value measurement. If the optional source parameter is specified, the current source is modified.

The <interval> option lets you specify the measurement interval: either an integral number of cycles, or the full screen. If <interval> is not specified, DISPlay is implied.

The <type> option lets you choose between a DC RMS measurement and an AC RMS measurement. If <type> is not specified, DC is implied.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VRMS? [<source>]

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

**Return Format** <value><NL>  
 <value> ::= calculated dc RMS value in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:VTIME**

**N** (see [page 608](#))

**Query Syntax** :MEASure:VTIME? <vtime\_argument>[,<source>]  
 <vtime\_argument> ::= time from trigger in seconds  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH  
 | WMEMory<r>}  
 <digital channels> ::= DIGital<d> for the MSO models  
 <n> ::= 1 to (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :MEASure:VTIME? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

**Return Format** <value><NL>  
 <value> ::= value at the specified time in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:TEDGE](#)" on page 317
  - "[:MEASure:TVALue](#)" on page 319



**:MEASure:VTOP**

**C** (see [page 608](#))

**Command Syntax** :MEASure:VTOP [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH}  
 <n> ::= 1-2 or 1-4 (# of analog channels) in NR1 format  
 <r> ::= 1-2 in NR1 format

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VTOP? [<source>]

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

**Return Format** <value><NL>  
 <value> ::= vertical value at the top of the waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VAMPLitude"](#) on page 321
  - [":MEASure:VBASe"](#) on page 323

**:MEASure:WINDow**

**N** (see [page 608](#))

**Command Syntax** :MEASure:WINDow <type>

<type> ::= {MAIN | ZOOM | AUTO}

When the zoomed time base is displayed, the :MEASure:WINDow command lets you specify the measurement window:

- MAIN – the measurement window is the upper, Main window.
- ZOOM – the measurement window is the lower, Zoom window.
- AUTO – the measurement is attempted in the lower, Zoom window; if it cannot be made there, the upper, Main window is used.

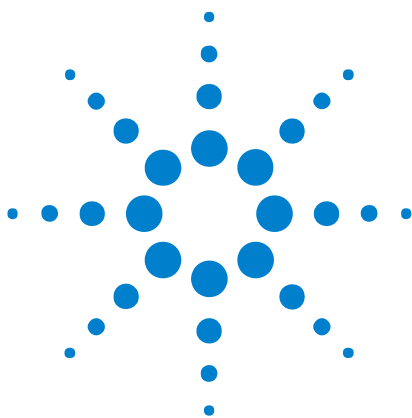
**Query Syntax** :MEASure:WINDow?

The :MEASure:WINDow? query returns the current measurement window setting.

**Return Format** <type><NL>

<type> ::= {MAIN | ZOOM | AUTO}

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 294
  - [":MEASure:SOURce"](#) on page 315



## 19 :MTESt Commands

The MTESt subsystem commands and queries control the mask test features. See "[Introduction to :MTESt Commands](#)" on page 333.

**Table 55** :MTESt Commands Summary

Command	Query	Options and Query Returns
:MTESt:ALL {{0   OFF}   {1   ON}} (see <a href="#">page 336</a> )	:MTESt:ALL? (see <a href="#">page 336</a> )	{0   1}
:MTESt:AMASk:CREate (see <a href="#">page 337</a> )	n/a	n/a
:MTESt:AMASk:SOURce <source> (see <a href="#">page 338</a> )	:MTESt:AMASk:SOURce? (see <a href="#">page 338</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 339</a> )	:MTESt:AMASk:UNITs? (see <a href="#">page 339</a> )	<units> ::= {CURRent   DIVisions}
:MTESt:AMASk:XDELta <value> (see <a href="#">page 340</a> )	:MTESt:AMASk:XDELta? (see <a href="#">page 340</a> )	<value> ::= X delta value in NR3 format
:MTESt:AMASk:YDELta <value> (see <a href="#">page 341</a> )	:MTESt:AMASk:YDELta? (see <a href="#">page 341</a> )	<value> ::= Y delta value in NR3 format
n/a	:MTESt:COUNt:FWAVEfor ms? [CHANnel<n>] (see <a href="#">page 342</a> )	<failed> ::= number of failed waveforms in NR1 format
:MTESt:COUNt:RESet (see <a href="#">page 343</a> )	n/a	n/a
n/a	:MTESt:COUNt:TIME? (see <a href="#">page 344</a> )	<time> ::= elapsed seconds in NR3 format
n/a	:MTESt:COUNt:WAVEform s? (see <a href="#">page 345</a> )	<count> ::= number of waveforms in NR1 format
:MTESt:DATA <mask> (see <a href="#">page 346</a> )	:MTESt:DATA? (see <a href="#">page 346</a> )	<mask> ::= data in IEEE 488.2 # format.



**Table 55** :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:DELeTe (see <a href="#">page 347</a> )	n/a	n/a
:MTESt:ENABLe {{0   OFF}   {1   ON}} (see <a href="#">page 348</a> )	:MTESt:ENABLe? (see <a href="#">page 348</a> )	{0   1}
:MTESt:LOCK {{0   OFF}   {1   ON}} (see <a href="#">page 349</a> )	:MTESt:LOCK? (see <a href="#">page 349</a> )	{0   1}
:MTESt:RMODe <rmode> (see <a href="#">page 350</a> )	:MTESt:RMODe? (see <a href="#">page 350</a> )	<rmode> ::= {FORever   TIME   SIGMa   WAVEforms}
:MTESt:RMODe:FACTion:MEASure {{0   OFF}   {1   ON}} (see <a href="#">page 351</a> )	:MTESt:RMODe:FACTion:MEASure? (see <a href="#">page 351</a> )	{0   1}
:MTESt:RMODe:FACTion:PRINt {{0   OFF}   {1   ON}} (see <a href="#">page 352</a> )	:MTESt:RMODe:FACTion:PRINt? (see <a href="#">page 352</a> )	{0   1}
:MTESt:RMODe:FACTion:SAVE {{0   OFF}   {1   ON}} (see <a href="#">page 353</a> )	:MTESt:RMODe:FACTion:SAVE? (see <a href="#">page 353</a> )	{0   1}
:MTESt:RMODe:FACTion:STOP {{0   OFF}   {1   ON}} (see <a href="#">page 354</a> )	:MTESt:RMODe:FACTion:STOP? (see <a href="#">page 354</a> )	{0   1}
:MTESt:RMODe:SIGMa <level> (see <a href="#">page 355</a> )	:MTESt:RMODe:SIGMa? (see <a href="#">page 355</a> )	<level> ::= from 0.1 to 9.3 in NR3 format
:MTESt:RMODe:TIME <seconds> (see <a href="#">page 356</a> )	:MTESt:RMODe:TIME? (see <a href="#">page 356</a> )	<seconds> ::= from 1 to 86400 in NR3 format
:MTESt:RMODe:WAVEform s <count> (see <a href="#">page 357</a> )	:MTESt:RMODe:WAVEform s? (see <a href="#">page 357</a> )	<count> ::= number of waveforms in NR1 format
:MTESt:SCALE:BINd {{0   OFF}   {1   ON}} (see <a href="#">page 358</a> )	:MTESt:SCALE:BINd? (see <a href="#">page 358</a> )	{0   1}
:MTESt:SCALE:X1 <x1_value> (see <a href="#">page 359</a> )	:MTESt:SCALE:X1? (see <a href="#">page 359</a> )	<x1_value> ::= X1 value in NR3 format

**Table 55** :MTESt Commands Summary (continued)

Command	Query	Options and Query Returns
:MTESt:SCALe:XDELta <xdelta_value> (see page 360)	:MTESt:SCALe:XDELta? (see page 360)	<xdelta_value> ::= X delta value in NR3 format
:MTESt:SCALe:Y1 <y1_value> (see page 361)	:MTESt:SCALe:Y1? (see page 361)	<y1_value> ::= Y1 value in NR3 format
:MTESt:SCALe:Y2 <y2_value> (see page 362)	:MTESt:SCALe:Y2? (see page 362)	<y2_value> ::= Y2 value in NR3 format
:MTESt:SOURce <source> (see page 363)	:MTESt:SOURce? (see page 363)	<source> ::= {CHANnel<n>   NONE} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
n/a	:MTESt:TITLe? (see page 364)	<title> ::= a string of up to 128 ASCII characters

**Introduction to :MTESt Commands** Mask testing automatically compares the current displayed waveform with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

#### Reporting the Setup

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

#### Return Format

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

```
:MTES:SOUR CHAN1;ENAB 0;LOCK 1;:MTES:AMAS:SOUR CHAN1;UNIT DIV;XDEL
+2.5000000E-001;YDEL +2.5000000E-001;:MTES:SCAL:X1 +200.000E-06;XDEL
+400.000E-06;Y1 -3.00000E+00;Y2 +3.00000E+00;BIND 0;:MTES:RMOD
FOR;RMOD:TIME +1E+00;WAV 1000;SIGM +6.0E+00;:MTES:RMOD:FACT:STOP
0;PRIN 0;SAVE 0
```

#### Example Code

```
' Mask testing commands example.
```

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

```
Option Explicit
```

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

```

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

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488
    Set myScope.IO = _
        myMgr.Open("USB0::0x0957::0x17A6::US50210029::0::INSTR")
    myScope.IO.Clear ' Clear the interface.

    ' Make sure oscilloscope is running.
    myScope.WriteString ":RUN"

    ' Set mask test termination conditions.
    myScope.WriteString ":MTESt:RMODE SIGMA"
    myScope.WriteString ":MTESt:RMODE?"
    strQueryResult = myScope.ReadString
    Debug.Print "Mask test termination mode: " + strQueryResult

    myScope.WriteString ":MTESt:RMODE:SIGMA 4.2"
    myScope.WriteString ":MTESt:RMODE:SIGMA?"
    varQueryResult = myScope.ReadNumber
    Debug.Print "Mask test termination 'test sigma': " + _
        FormatNumber(varQueryResult)

    ' Use auto-mask to create mask.
    myScope.WriteString ":MTESt:AMASK:SOURce CHANnel1"
    myScope.WriteString ":MTESt:AMASK:SOURce?"
    strQueryResult = myScope.ReadString
    Debug.Print "Mask test auto-mask source: " + strQueryResult

    myScope.WriteString ":MTESt:AMASK:UNITs DIVisions"
    myScope.WriteString ":MTESt:AMASK:UNITs?"
    strQueryResult = myScope.ReadString
    Debug.Print "Mask test auto-mask units: " + strQueryResult

    myScope.WriteString ":MTESt:AMASK:XDELta 0.1"
    myScope.WriteString ":MTESt:AMASK:XDELta?"
    varQueryResult = myScope.ReadNumber
    Debug.Print "Mask test auto-mask X delta: " + _
        FormatNumber(varQueryResult)

    myScope.WriteString ":MTESt:AMASK:YDELta 0.1"
    myScope.WriteString ":MTESt:AMASK:YDELta?"
    varQueryResult = myScope.ReadNumber
    Debug.Print "Mask test auto-mask Y delta: " + _
        FormatNumber(varQueryResult)

    ' Enable "Auto Mask Created" event (bit 10, &H400)
    myScope.WriteString "*CLS"
    myScope.WriteString ":MTEenable " + CStr(CInt("&H400"))

    ' Create mask.

```

```

myScope.WriteString ":MTESt:AMASk:CREate"
Debug.Print "Auto-mask created, mask test automatically enabled."

' Set up timeout variables.
Dim lngTimeout As Long ' Max millisecs to wait.
Dim lngElapsed As Long
lngTimeout = 60000 ' 60 seconds.

' Wait until mask is created.
lngElapsed = 0
Do While lngElapsed <= lngTimeout
  myScope.WriteString ":OPERRegister:CONDition?"
  varQueryResult = myScope.ReadNumber
  ' Operation Status Condition Register MTE bit (bit 9, &H200).
  If (varQueryResult And &H200) <> 0 Then
    Exit Do
  Else
    Sleep 100 ' Small wait to prevent excessive queries.
    lngElapsed = lngElapsed + 100
  End If
Loop

' Look for RUN bit = stopped (mask test termination).
lngElapsed = 0
Do While lngElapsed <= lngTimeout
  myScope.WriteString ":OPERRegister:CONDition?"
  varQueryResult = myScope.ReadNumber
  ' Operation Status Condition Register RUN bit (bit 3, &H8).
  If (varQueryResult And &H8) = 0 Then
    Exit Do
  Else
    Sleep 100 ' Small wait to prevent excessive queries.
    lngElapsed = lngElapsed + 100
  End If
Loop

' Get total waveforms, failed waveforms, and test time.
myScope.WriteString ":MTESt:COUNT:WAVEforms?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test total waveforms: " + strQueryResult

myScope.WriteString ":MTESt:COUNT:FWAVEforms?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test failed waveforms: " + strQueryResult

myScope.WriteString ":MTESt:COUNT:TIME?"
strQueryResult = myScope.ReadString
Debug.Print "Mask test elapsed seconds: " + strQueryResult

Exit Sub

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

End Sub

```

## :MTESt:ALL

**N** (see [page 608](#))

**Command Syntax** :MTESt:ALL <on\_off>  
<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :MTESt:ALL command specifies the channel(s) that are included in the mask test:

- ON – All displayed analog channels are included in the mask test.
- OFF – Just the selected source channel is included in the test.

**Query Syntax** :MTESt:ENABle?

The :MTESt:ENABle? query returns the current setting.

**Return Format** <on\_off><NL>  
<on\_off> ::= {1 | 0}

**See Also** • ["Introduction to :MTESt Commands"](#) on page 333



**:MTESt:AMASk:CREate**

**N** (see [page 608](#))

**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 333
  - [":MTESt:AMASk:XDELta"](#) on page 340
  - [":MTESt:AMASk:YDELta"](#) on page 341
  - [":MTESt:AMASk:UNITs"](#) on page 339
  - [":MTESt:AMASk:SOURce"](#) on page 338
  - [":MTESt:SOURce"](#) on page 363

- Example Code**
- ["Example Code"](#) on page 333

**:MTESt:AMASk:SOURce**

**N** (see [page 608](#))

**Command Syntax** :MTESt:AMASk:SOURce <source>

<source> ::= CHANnel<n>

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

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the :MTESt:AMASk:XDELta and :MTESt:AMASk:YDELta parameters when :MTESt:AMASk:UNITs is set to CURRent.

When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel<n>:UNITs command, of the selected source.

Suppose that UNITs are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

This command is the same as the :MTESt:SOURce command.

**Query Syntax** :MTESt:AMASk:SOURce?

The :MTESt:AMASk:SOURce? query returns the currently set source.

**Return Format** <source> ::= CHAN<n>

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

- See Also**
- ["Introduction to :MTESt Commands"](#) on page 333
  - [":MTESt:AMASk:XDELta"](#) on page 340
  - [":MTESt:AMASk:YDELta"](#) on page 341
  - [":MTESt:AMASk:UNITs"](#) on page 339
  - [":MTESt:SOURce"](#) on page 363

**Example Code** • ["Example Code"](#) on page 333

**:MTESt:AMASk:UNITs**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:AMASk:XDELta](#)" on page 340
  - "[:MTESt:AMASk:YDELta](#)" on page 341
  - "[:CHANnel<n>:UNITs](#)" on page 209
  - "[:MTESt:AMASk:SOURce](#)" on page 338
  - "[:MTESt:SOURce](#)" on page 363

**Example Code** • "[Example Code](#)" on page 333

**:MTESt:AMASk:XDELta**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:AMASk:UNITs](#)" on page 339
  - "[:MTESt:AMASk:YDELta](#)" on page 341
  - "[:MTESt:AMASk:SOURce](#)" on page 338
  - "[:MTESt:SOURce](#)" on page 363

- Example Code**
- "[Example Code](#)" on page 333

**:MTESt:AMASk:YDELta**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:AMASk:UNITs](#)" on page 339
  - "[:MTESt:AMASk:XDELta](#)" on page 340
  - "[:MTESt:AMASk:SOURce](#)" on page 338
  - "[:MTESt:SOURce](#)" on page 363

- Example Code**
- "[Example Code](#)" on page 333

**:MTESt:COUNT:FWAVEforms**

**N** (see [page 608](#))

**Query Syntax** :MTESt:COUNT:FWAVEforms? [CHANnel<n>]

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

The :MTESt:COUNT:FWAVEforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms.

**Return Format** <failed><NL>

<failed> ::= number of failed waveforms in NR1 format.

- See Also**
- ["Introduction to :MTESt Commands"](#) on page 333
  - [":MTESt:COUNT:WAVEforms"](#) on page 345
  - [":MTESt:COUNT:TIME"](#) on page 344
  - [":MTESt:COUNT:RESet"](#) on page 343
  - [":MTESt:SOURce"](#) on page 363

**Example Code** • ["Example Code"](#) on page 333

## :MTESt:COUNT:RESet

**N** (see [page 608](#))

**Command Syntax** :MTESt:COUNT:RESet

The :MTESt:COUNT:RESet command resets the mask statistics.

- See Also**
- "[Introduction to :MTESt Commands](#)" on page 333
  - "[:MTESt:COUNT:WAVEforms](#)" on page 345
  - "[:MTESt:COUNT:FWAVEforms](#)" on page 342
  - "[:MTESt:COUNT:TIME](#)" on page 344

## :MTESt:COUNT:TIME

**N** (see [page 608](#))

**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 333
  - [":MTESt:COUNT:WAVEforms"](#) on page 345
  - [":MTESt:COUNT:FWAVEforms"](#) on page 342
  - [":MTESt:COUNT:RESet"](#) on page 343

**Example Code** • ["Example Code"](#) on page 333



**:MTESt:COUNT:WAVEforms****N** (see [page 608](#))**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 333
  - [":MTESt:COUNT:FWAVEforms"](#) on page 342
  - [":MTESt:COUNT:TIME"](#) on page 344
  - [":MTESt:COUNT:RESet"](#) on page 343

**Example Code** • ["Example Code"](#) on page 333

## :MTEST:DATA

**N** (see [page 608](#))

**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 388](#)
  - [":RECall:MASK\[:START\]"](#) on [page 374](#)

## :MTESt:DELeTe

**N** (see [page 608](#))

**Command Syntax** :MTESt:DELeTe

The :MTESt:DELeTe command clears the currently loaded mask.

- See Also**
- "[Introduction to :MTESt Commands](#)" on page 333
  - "[:MTESt:AMASk:CREate](#)" on page 337

## :MTESt:ENABle

**N** (see [page 608](#))

**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 333

**:MTESt:LOCK**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:SOURce](#)" on page 363

**:MTESt:RMODe**

**N** (see [page 608](#))

**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 355](#) 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 356](#) 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 357](#) 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 333](#)
  - "[:MTESt:RMODe:SIGMa](#)" on [page 355](#)
  - "[:MTESt:RMODe:TIME](#)" on [page 356](#)
  - "[:MTESt:RMODe:WAVeforms](#)" on [page 357](#)

**Example Code** • "[Example Code](#)" on [page 333](#)

**:MTESt:RMODe:FACTion:MEASure**

**N** (see [page 608](#))

**Command Syntax** :MTESt:RMODe:FACTion:MEASure <on\_off>

<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :MTESt:RMODe:FACTion:MEASure command sets measuring only mask failures on or off.

When ON, measurements and measurement statistics run only on waveforms that contain a mask violation; passing waveforms do not affect measurements and measurement statistics.

This mode is not available when the acquisition mode is set to Averaging.

**Query Syntax** :MTESt:RMODe:FACTion:MEASure?

The :MTESt:RMODe:FACTion:MEASure? query returns the current mask failure measure setting.

**Return Format** <on\_off><NL>

<on\_off> ::= {1 | 0}

- See Also**
- ["Introduction to :MTESt Commands"](#) on page 333
  - [":MTESt:RMODe:FACTion:PRINT"](#) on page 352
  - [":MTESt:RMODe:FACTion:SAVE"](#) on page 353
  - [":MTESt:RMODe:FACTion:STOP"](#) on page 354

**:MTESt:RMODe:FACTion:PRINt**

**N** (see [page 608](#))

**Command Syntax** :MTESt:RMODe:FACTion:PRINt <on\_off>  
 <on\_off> ::= {{1 | ON} | {0 | OFF}}

The :MTESt:RMODe:FACTion:PRINt command sets printing on mask failures on or off.

**NOTE**

Setting :MTESt:RMODe:FACTion:PRINt ON automatically sets :MTESt:RMODe:FACTion:SAVE OFF.

See [Chapter 16](#), “:HARDcopy Commands,” starting on page 257 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 333
  - "[:MTESt:RMODe:FACTion:MEASure](#)" on page 351
  - "[:MTESt:RMODe:FACTion:SAVE](#)" on page 353
  - "[:MTESt:RMODe:FACTion:STOP](#)" on page 354



**:MTESt:RMODe:FACTion:SAVE**

**N** (see [page 608](#))

**Command Syntax** :MTESt:RMODe:FACTion:SAVE <on\_off>  
 <on\_off> ::= {{1 | ON} | {0 | OFF}}

The :MTESt:RMODe:FACTion:SAVE command sets saving on mask failures on or off.

**NOTE**

Setting :MTESt:RMODe:FACTion:SAVE ON automatically sets :MTESt:RMODe:FACTion:PRINT OFF.

See [Chapter 22](#), “:SAVE Commands,” starting on [page 379](#) 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 333](#)
  - "[:MTESt:RMODe:FACTion:MEASure](#)" on [page 351](#)
  - "[:MTESt:RMODe:FACTion:PRINT](#)" on [page 352](#)
  - "[:MTESt:RMODe:FACTion:STOP](#)" on [page 354](#)

**:MTESt:RMODe:FACTion:STOP**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:RMODe:FACTion:MEASure](#)" on page 351
  - "[:MTESt:RMODe:FACTion:PRINt](#)" on page 352
  - "[:MTESt:RMODe:FACTion:SAVE](#)" on page 353

**:MTESt:RMODE:SIGMa**

**N** (see [page 608](#))

**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 333
  - [":MTESt:RMODE"](#) on page 350

- Example Code**
- ["Example Code"](#) on page 333

## :MTESt:RMODe:TIME

**N** (see [page 608](#))

**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 333
  - "[:MTESt:RMODe](#)" on page 350

**:MTESt:RMODe:WAVeforms**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:RMODe](#)" on page 350

**:MTESt:SCALe:BIND**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:SCALe:X1](#)" on page 359
  - "[:MTESt:SCALe:XDELta](#)" on page 360
  - "[:MTESt:SCALe:Y1](#)" on page 361
  - "[:MTESt:SCALe:Y2](#)" on page 362

**:MTESt:SCALe:X1**

**N** (see [page 608](#))

**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 333
  - [":MTESt:SCALe:BIND"](#) on page 358
  - [":MTESt:SCALe:XDELta"](#) on page 360
  - [":MTESt:SCALe:Y1"](#) on page 361
  - [":MTESt:SCALe:Y2"](#) on page 362

**:MTESt:SCALe:XDELta**

**N** (see [page 608](#))

**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 333
  - [":MTESt:SCALe:BIND"](#) on page 358
  - [":MTESt:SCALe:X1"](#) on page 359
  - [":MTESt:SCALe:Y1"](#) on page 361
  - [":MTESt:SCALe:Y2"](#) on page 362



**:MTESt:SCALe:Y1**

**N** (see [page 608](#))

**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 333
  - [":MTESt:SCALe:BIND"](#) on page 358
  - [":MTESt:SCALe:X1"](#) on page 359
  - [":MTESt:SCALe:XDELta"](#) on page 360
  - [":MTESt:SCALe:Y2"](#) on page 362

**:MTESt:SCALe:Y2**

**N** (see [page 608](#))

**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 333
  - "[:MTESt:SCALe:BIND](#)" on page 358
  - "[:MTESt:SCALe:X1](#)" on page 359
  - "[:MTESt:SCALe:XDELta](#)" on page 360
  - "[:MTESt:SCALe:Y1](#)" on page 361

**:MTESt:SOURce**

**N** (see [page 608](#))

**Command Syntax** :MTESt:SOURce <source>

<source> ::= CHANnel<n>

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

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

**Query Syntax** :MTESt:SOURce?

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

**Return Format** <source><NL>

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

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

- See Also**
- ["Introduction to :MTESt Commands"](#) on page 333
  - [":MTESt:AMASK:SOURce"](#) on page 338

## :MTESt:TITLe

**N** (see [page 608](#))

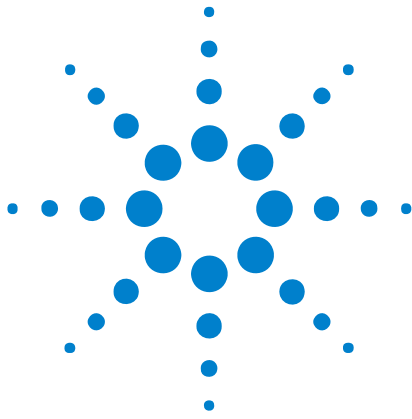
**Query Syntax** :MTESt:TITLe?

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 333



## 20 :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "[Introduction to :POD<n> Commands](#)" on page 365.

**Table 56** :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 366</a> )	:POD<n>:DISPlay? (see <a href="#">page 366</a> )	{0   1} <n> ::= 1 in NR1 format
:POD<n>:SIZE <value> (see <a href="#">page 367</a> )	:POD<n>:SIZE? (see <a href="#">page 367</a> )	<value> ::= {SMALl   MEDium   LARGe}
:POD<n>:THReshold <type>[suffix] (see <a href="#">page 368</a> )	:POD<n>:THReshold? (see <a href="#">page 368</a> )	<n> ::= 1 in NR1 format <type> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V   mV   uV }

### Introduction to :POD<n> Commands

<n> ::= 1

The POD subsystem commands control the viewing and threshold of groups of digital channels.

POD1 ::= D0-D7

### NOTE

These commands are only valid for the MSO models.

### Reporting the Setup

Use :POD1? to query setup information for the POD subsystem.

### Return Format

The following is a sample response from the :POD1? query. In this case, the query was issued following a \*RST command.

```
:POD1:DISP 0;THR 1.40E+00
```



**:POD<n>:DISPlay**

**N** (see [page 608](#))

**Command Syntax** :POD<n>:DISPlay <display>

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

<n> ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.

POD1 ::= D0-D7

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :POD<n>:DISPlay?

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

**Return Format** <display><NL>

<display> ::= {0 | 1}

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 365
  - [":DIGital<d>:DISPlay"](#) on page 219
  - [":CHANnel<n>:DISPlay"](#) on page 196
  - [":VIEW"](#) on page 156
  - [":BLANK"](#) on page 132
  - [":STATus"](#) on page 153

**:POD<n>:SIZE**

**N** (see [page 608](#))

**Command Syntax** :POD<n>:SIZE <value>

<n> ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.

POD1 ::= D0-D7

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

The :POD<n>:SIZE command specifies the size of digital channels on the display.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :POD<n>:SIZE?

The :POD<n>:SIZE? query returns the digital channels size setting.

**Return Format** <size\_value><NL>

<size\_value> ::= {SMAL | MED | LARG}

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 365
  - [":DIGital<d>:SIZE"](#) on page 222
  - [":DIGital<d>:POSition"](#) on page 221

**:POD<n>:THReshold**

**N** (see [page 608](#))

**Command Syntax** :POD<n>:THReshold <type>[<suffix>]

<n> ::= An integer, 1, is attached as a suffix to the command and defines the group of channels that are affected by the command.

<type> ::= {CMOS | ECL | TTL | <user defined value>}

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

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

POD1 ::= D0-D7

TTL ::= 1.4V

CMOS ::= 2.5V

ECL ::= -1.3V

The :POD<n>:THReshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :POD<n>:THReshold?

The :POD<n>:THReshold? query returns the threshold value for the specified group of channels.

**Return Format** <threshold><NL>

<threshold> ::= Floating point number in NR3 format

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 365
  - [":DIGital<d>:THReshold"](#) on page 223
  - [":TRIGger\[:EDGE\]:LEVel"](#) on page 435

**Example Code**

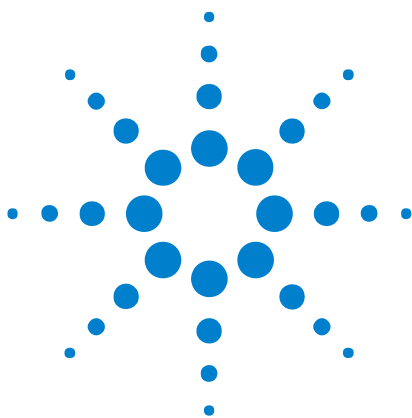
```
' THRESHOLD - This command is used to set the voltage threshold for
' the waveforms. There are three preset values (TTL, CMOS, and ECL)
' and you can also set a user-defined threshold value between
' -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, and then set the
' external trigger to TTL. Of course, you only need to set the
' thresholds for the channels you will be using in your program.
'
' Set channels 0-7 to CMOS threshold.
myScope.WriteString ":POD1:THRESHOLD CMOS"
```



```
' Set external channel to TTL threshold (short form).  
myScope.WriteString ":TRIG:LEV TTL,EXT"
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617





## 21 :RECall Commands

Recall previously saved oscilloscope setups, reference waveforms, and masks.

**Table 57** :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILEname <base_name> (see page 373)	:RECall:FILEname? (see page 373)	<base_name> ::= quoted ASCII string
:RECall:MASK[:START] [<file_spec>] (see page 374)	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 375)	:RECall:PWD? (see page 375)	<path_name> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 376)	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:RECall:WMEMemory<r>[:S TART] [<file_name>] (see page 377)	n/a	<r> ::= 1-2 in NR1 format <file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

### Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups, reference waveforms, and masks.

#### Reporting the Setup

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



## 21 :RECall Commands

### 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 608](#))

**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 371
  - [":RECall:SETup\[:START\]"](#) on page 376
  - [":SAVE:FILEname"](#) on page 382

## :RECall:MASK[:START]

**N** (see [page 608](#))

**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 371
  - [":RECall:FILENAME"](#) on page 373
  - [":SAVE:MASK\[:START\]"](#) on page 388
  - [":MTEST:DATA"](#) on page 346

## :RECall:PWD

**N** (see [page 608](#))

**Command Syntax** :RECall:PWD <path\_name>  
<path\_name> ::= quoted ASCII string

The :RECall:PWD command sets the present working directory for recall operations.

**Query Syntax** :RECall:PWD?

The :RECall:PWD? query returns the currently set working directory for recall operations.

**Return Format** <path\_name><NL>  
<path\_name> ::= quoted ASCII string

- See Also**
- ["Introduction to :RECall Commands"](#) on page 371
  - [":SAVE:PWD"](#) on page 389

## :RECall:SETup[:START]

**N** (see [page 608](#))

**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 371
  - "[:RECall:FILENAME](#)" on page 373
  - "[:SAVE:SETup\[:START\]](#)" on page 390



**:RECall:WMEMemory<r>[:START]**

**N** (see [page 608](#))

**Command Syntax** :RECall:WMEMemory<r>[:START] [<file\_name>]

<r> ::= 1-2 in NR1 format

<file\_name> ::= quoted ASCII string

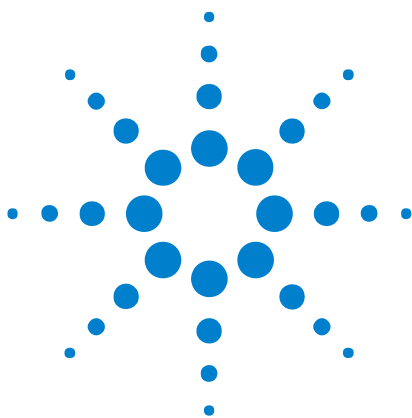
The :RECall:WMEMemory<r>[:START] command recalls a reference waveform.

**NOTE**

If a file extension is provided as part of a specified <file\_name>, it must be ".h5".

- See Also**
- "[Introduction to :RECall Commands](#)" on page 371
  - "[:RECall:FILENAME](#)" on page 373
  - "[:SAVE:WMEMemory\[:START\]](#)" on page 396





## 22 :SAVE Commands

Save oscilloscope setups, screen images, and data. See "[Introduction to :SAVE Commands](#)" on page 380.

**Table 58** :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILENAME <base_name> (see <a href="#">page 382</a> )	:SAVE:FILENAME? (see <a href="#">page 382</a> )	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_name>] (see <a href="#">page 383</a> )	n/a	<file_name> ::= quoted ASCII string
:SAVE:IMAGE:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 384</a> )	:SAVE:IMAGE:FACTors? (see <a href="#">page 384</a> )	{0   1}
:SAVE:IMAGE:FORMat <format> (see <a href="#">page 385</a> )	:SAVE:IMAGE:FORMat? (see <a href="#">page 385</a> )	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGE:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 386</a> )	:SAVE:IMAGE:INKSaver? (see <a href="#">page 386</a> )	{0   1}
:SAVE:IMAGE:PALette <palette> (see <a href="#">page 387</a> )	:SAVE:IMAGE:PALette? (see <a href="#">page 387</a> )	<palette> ::= {COLor   GRAYscale   MONochrome}
:SAVE:MASK[:START] [<file_spec>] (see <a href="#">page 388</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 389</a> )	:SAVE:PWD? (see <a href="#">page 389</a> )	<path_name> ::= quoted ASCII string



**Table 58** :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:SETup[:START] [ <file_spec>] (see page 390)</file_spec>	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 391)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVEform:FORMat <format> (see page 392)	:SAVE:WAVEform:FORMat? (see page 392)	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVEform:LENGth <length> (see page 393)	:SAVE:WAVEform:LENGth? (see page 393)	<length> ::= 100 to max. length; an integer in NR1 format
:SAVE:WAVEform:SEGMent <option> (see page 394)	:SAVE:WAVEform:SEGMent? (see page 394)	<option> ::= {ALL   CURRent}
:SAVE:WMEMory:SOURce <source> (see page 395)	:SAVE:WMEMory:SOURce? (see page 395)	<source> ::= {CHANnel<n>   FUNction   MATH   WMEMory<r>} <n> ::= 1 to (# analog channels) in NR1 format <r> ::= 1-2 in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms. <return_value> ::= <source>
:SAVE:WMEMory[:START] [<file_name>] (see page 396)	n/a	<file_name> ::= quoted ASCII string If extension included in file name, it must be ".h5".

**Introduction to :SAVE Commands** The :SAVE subsystem provides commands to save oscilloscope setups, screen images, and data.

:SAV is an acceptable short form for :SAVE.

**Reporting the Setup**

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

**Return Format**

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

```
:SAVE:FIL " ";:SAVE:IMAG:AREA GRAT;FACT 0;FORM TIFF;INKS 0;PAL  
MON;:SAVE:PWD "C:/setups/";:SAVE:WAV:FORM NONE;LENG 1000;SEGM CURR
```

**:SAVE:FILEname**

**N** (see [page 608](#))

**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 380
  - [":SAVE:IMAGe\[:START\]"](#) on page 383
  - [":SAVE:SETup\[:START\]"](#) on page 390
  - [":SAVE:WAVEform\[:START\]"](#) on page 391
  - [":SAVE:PWD"](#) on page 389
  - [":RECall:FILEname"](#) on page 373

**:SAVE:IMAGe[:START]**

**N** (see [page 608](#))

**Command Syntax** :SAVE:IMAGe[:START] [<file\_name>  
 <file\_name> ::= quoted ASCII string

The :SAVE:IMAGe[:START] command saves an image.

**NOTE**

Be sure to set the :SAVE:IMAGe:FORMat before saving an image. If the format is NONE, the save image command will not succeed.

**NOTE**

If a file extension is provided as part of a specified <file\_name>, and it does not match the extension expected by the format specified in :SAVE:IMAGe:FORMat, the format will be changed if the extension is a valid image file extension.

**NOTE**

If the extension ".bmp" is used and the current :SAVE:IMAGe:FORMat is not BMP or BMP8, the format will be changed to BMP.

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 380
  - "[:SAVE:IMAGe:FACTors](#)" on page 384
  - "[:SAVE:IMAGe:FORMat](#)" on page 385
  - "[:SAVE:IMAGe:INKSaver](#)" on page 386
  - "[:SAVE:IMAGe:PALette](#)" on page 387
  - "[:SAVE:FILEname](#)" on page 382

**:SAVE:IMAGe:FACTors**

**N** (see [page 608](#))

**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 380
  - [":SAVE:IMAGe\[:START\]"](#) on page 383
  - [":SAVE:IMAGe:FORMat"](#) on page 385
  - [":SAVE:IMAGe:INKSaver"](#) on page 386
  - [":SAVE:IMAGe:PALette"](#) on page 387



**:SAVE:IMAGe:FORMat**

**N** (see [page 608](#))

**Command Syntax** :SAVE:IMAGe:FORMat <format>

<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}

The :SAVE:IMAGe:FORMat command sets the image format type.

**Query Syntax** :SAVE:IMAGe:FORMat?

The :SAVE:IMAGe:FORMat? query returns the selected image format type.

**Return Format** <format><NL>

<format> ::= {BMP | BMP8 | PNG | NONE}

When NONE is returned, it indicates that a waveform data file format is currently selected.

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 380
  - [":SAVE:IMAGe\[:START\]"](#) on page 383
  - [":SAVE:IMAGe:FACTors"](#) on page 384
  - [":SAVE:IMAGe:INKSaver"](#) on page 386
  - [":SAVE:IMAGe:PALette"](#) on page 387
  - [":SAVE:WAVEform:FORMat"](#) on page 392

## :SAVE:IMAGe:INKSaver

**N** (see [page 608](#))

**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 380
  - "[:SAVE:IMAGe\[:START\]](#)" on page 383
  - "[:SAVE:IMAGe:FACTors](#)" on page 384
  - "[:SAVE:IMAGe:FORMat](#)" on page 385
  - "[:SAVE:IMAGe:PALette](#)" on page 387

**:SAVE:IMAGe:PALette**

**N** (see [page 608](#))

**Command Syntax** :SAVE:IMAGe:PALette <palette>

<palette> ::= {COLor | GRAYscale}

The :SAVE:IMAGe:PALette command sets the image palette color.

**Query Syntax** :SAVE:IMAGe:PALette?

The :SAVE:IMAGe:PALette? query returns the selected image palette color.

**Return Format** <palette><NL>

<palette> ::= {COL | GRAY}

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 380
  - [":SAVE:IMAGe\[:START\]"](#) on page 383
  - [":SAVE:IMAGe:FACTors"](#) on page 384
  - [":SAVE:IMAGe:FORMat"](#) on page 385
  - [":SAVE:IMAGe:INKSaver"](#) on page 386

## :SAVE:MASK[:START]

**N** (see [page 608](#))

**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 380
  - [":SAVE:FILENAME"](#) on page 382
  - [":RECALL:MASK\[:START\]"](#) on page 374
  - [":MTEST:DATA"](#) on page 346

## :SAVE:PWD

**N** (see [page 608](#))

**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 380
  - [":SAVE:FILENAME"](#) on page 382
  - [":RECALL:PWD"](#) on page 375

## :SAVE:SETup[:START]

**N** (see [page 608](#))

**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 380
  - "[:SAVE:FILENAME](#)" on page 382
  - "[:RECall:SETup\[:START\]](#)" on page 376

**:SAVE:WAVEform[:START]**

**N** (see [page 608](#))

**Command Syntax** :SAVE:WAVEform[:START] [<file\_name>]  
 <file\_name> ::= quoted ASCII string

The :SAVE:WAVEform[:START] command saves oscilloscope waveform data to a file.

**NOTE**

Be sure to set the :SAVE:WAVEform:FORMat before saving waveform data. If the format is NONE, the save waveform command will not succeed.

**NOTE**

If a file extension is provided as part of a specified <file\_name>, and it does not match the extension expected by the format specified in :SAVE:WAVEform:FORMat, the format will be changed if the extension is a valid waveform file extension.

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 380
  - "[:SAVE:WAVEform:FORMat](#)" on page 392
  - "[:SAVE:WAVEform:LENGth](#)" on page 393
  - "[:SAVE:FILEname](#)" on page 382
  - "[:RECall:SETup\[:START\]](#)" on page 376

**:SAVE:WAVEform:FORMat**

**N** (see [page 608](#))

**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 380
  - "[:SAVE:WAVEform\[:START\]](#)" on page 391
  - "[:SAVE:WAVEform:LENGth](#)" on page 393
  - "[:SAVE:IMAGe:FORMat](#)" on page 385



**:SAVE:WAVEform:LENGth**

**N** (see [page 608](#))

**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 380
  - [":SAVE:WAVEform\[:START\]"](#) on page 391
  - [":WAVEform:POINTs"](#) on page 472
  - [":SAVE:WAVEform:FORMat"](#) on page 392

**:SAVE:WAVEform:SEGmented**

**N** (see [page 608](#))

**Command Syntax** :SAVE:WAVEform:SEGmented <option>  
 <option> ::= {ALL | CURRent}

When segmented memory is used for acquisitions, the :SAVE:WAVEform:SEGmented command specifies which segments are included when the waveform is saved:

- ALL – all acquired segments are saved.
- CURRent – only the currently selected segment is saved.

**Query Syntax** :SAVE:WAVEform:SEGmented?

The :SAVE:WAVEform:SEGmented? query returns the current segmented waveform save option setting.

**Return Format** <option><NL>  
 <option> ::= {ALL | CURR}

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 380
  - [":SAVE:WAVEform\[:START\]"](#) on page 391
  - [":SAVE:WAVEform:FORMat"](#) on page 392
  - [":SAVE:WAVEform:LENGth"](#) on page 393

**:SAVE:WMEMemory:SOURce**

**N** (see [page 608](#))

**Command Syntax** :SAVE:WMEMemory:SOURce <source>  
 <source> ::= {CHANnel<n> | FUNCtion | MATH | WMEMemory<r>}  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <r> ::= {1 | 2}

The :SAVE:WMEMemory:SOURce command selects the source to be saved as a reference waveform file.

**NOTE**

Only ADD or SUBtract math operations can be saved as reference waveforms.

**NOTE**

MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

**Query Syntax** :SAVE:WMEMemory:SOURce?

The :SAVE:WMEMemory:SOURce? query returns the source to be saved as a reference waveform file.

**Return Format** <source><NL>  
 <source> ::= {CHAN<n> | FUNC | WMEM<r> | NONE}

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 380
  - "[:SAVE:WMEMemory\[:START\]](#)" on page 396
  - "[:RECall:WMEMemory<r>\[:START\]](#)" on page 377

## :SAVE:WMEMory[:START]

**N** (see [page 608](#))

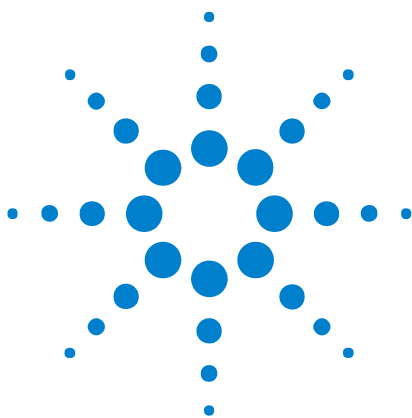
**Command Syntax** :SAVE:WMEMory[:START] [<file\_name>]  
<file\_name> ::= quoted ASCII string

The :SAVE:WMEMory[:START] command saves oscilloscope waveform data to a reference waveform file.

### NOTE

If a file extension is provided as part of a specified <file\_name>, it must be ".h5".

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 380
  - "[:SAVE:WMEMory:SOURce](#)" on page 395
  - "[:RECall:WMEMory<r>\[:START\]](#)" on page 377



## 23 :SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 398.

**Table 59** :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see <a href="#">page 399</a> )	:SYSTem:DATE? (see <a href="#">page 399</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 400</a> )	n/a	<string> ::= up to 75 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see <a href="#">page 401</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 569</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 402</a> )	:SYSTem:LOCK? (see <a href="#">page 402</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PRESet (see <a href="#">page 403</a> )	n/a	See :SYSTem:PRESet (see <a href="#">page 403</a> )
:SYSTem:PROTection:LOCK <value> (see <a href="#">page 406</a> )	:SYSTem:PROTection:LOCK? (see <a href="#">page 406</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 407</a> )	:SYSTem:SETup? (see <a href="#">page 407</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 409</a> )	:SYSTem:TIME? (see <a href="#">page 409</a> )	<time> ::= hours,minutes,seconds in NR1 format



## 23 :SYSTem Commands

**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 608](#))

**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 398
  - [":SYSTEM:TIME"](#) on page 409

## :SYSTem:DSP

**N** (see [page 608](#))

**Command Syntax** :SYSTem:DSP <string>  
<string> ::= quoted ASCII string (up to 75 characters)

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

**See Also** • ["Introduction to :SYSTem Commands"](#) on page 398



**:SYSTem:ERRor**

**C** (see [page 608](#))

**Query Syntax** :SYSTem:ERRor?

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

**Return Format** <error number>,<error string><NL>

<error number> ::= an integer error code in NR1 format

<error string> ::= quoted ASCII string containing the error message

Error messages are listed in [Chapter 30](#), “Error Messages,” starting on page 569.

- See Also**
- "Introduction to :SYSTem Commands" on page 398
  - "\*ESR (Standard Event Status Register)" on page 102
  - "\*CLS (Clear Status)" on page 99

## :SYSTem:LOCK

**N** (see [page 608](#))

**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 398

**:SYSTem:PRESet**

**C** (see page 608)

**Command Syntax** :SYSTem:PRESet

The :SYSTem:PRESet command places the instrument in a known state. This is the same as pressing the [**Default Setup**] key or [**Save/Recall**] > **Default/Erse** > **Default Setup** on the front panel.

When you perform a default setup, some user settings (like preferences) remain unchanged. To reset all user settings to their factory defaults, use the \*RST command.

Reset conditions are:

<b>Acquire Menu</b>	
Mode	Normal
Averaging	Off
# Averages	8

<b>Analog Channel Menu</b>	
Channel 1	On
Channel 2	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	10:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm (cannot be changed)
Units	Volts
Skew	0

<b>Cursor Menu</b>	
Source	Channel 1

<b>Digital Channel Menu (MSO models only)</b>	
Channel 0 - 7	Off
Labels	Off
Threshold	TTL (1.4V)

<b>Display Menu</b>	
Persistence	Off
Grid	33%

<b>Quick Meas Menu</b>	
Source	Channel 1

<b>Run Control</b>	
	Scope is running

<b>Time Base Menu</b>	
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

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

Trigger Menu	
HF Reject and noise reject	Off
Holdoff	60 ns
External probe attenuation	10:1
External Units	Volts
External Impedance	1 M Ohm (cannot be changed)

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 97
  - ["\\*RST \(Reset\)"](#) on page 109

## :SYSTem:PROTection:LOCK

**N** (see [page 608](#))

**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 398

**:SYSTem:SETup**

**C** (see [page 608](#))

**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 398
  - ["\\*LRN \(Learn Device Setup\)"](#) on page 105

**Example Code**

```
' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
' message that contains the current state of the instrument. Its
' format is a definite-length binary block, for example,
' #800075595<setup string><NL>
' where the setup string is 75595 bytes in length.
myScope.WriteString ":SYSTEM:SETUP?"
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors ' After reading query results.

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"

' Open file for output.
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult ' Write data.
Close #1 ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and
' write it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"

' Open file for input.
Open strPath For Binary Access Read As #1
Get #1, , varSetupString ' Read data.
Close #1 ' Close file.
```

## 23 :SYSTem Commands

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"  
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617



## :SYSTem:TIME

**N** (see [page 608](#))

**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>

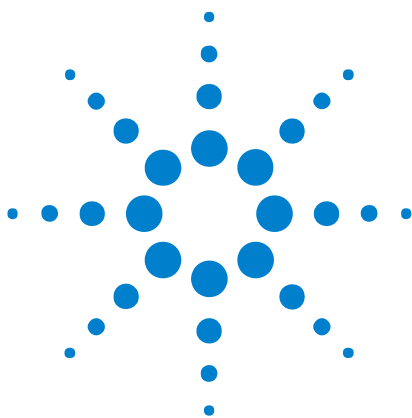
The :SYSTem:TIME? query returns the current system time.

**Return Format** <time><NL>

<time> ::= hours,minutes,seconds in NR1 format

- See Also**
- "[Introduction to :SYSTem Commands](#)" on page 398
  - "[:SYSTem:DATE](#)" on page 399





## 24 :TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 412.

**Table 60** :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 413)	:TIMebase:MODE? (see page 413)	<value> ::= {MAIN   WINDow   XY   ROLL}
:TIMebase:POSition <pos> (see page 414)	:TIMebase:POSition? (see page 414)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 415)	:TIMebase:RANGe? (see page 415)	<range_value> ::= 50 ns through 500 s in NR3 format
:TIMebase:REFerence {LEFT   CENTer   RIGHT} (see page 416)	:TIMebase:REFerence? (see page 416)	<return_value> ::= {LEFT   CENTer   RIGHT}
:TIMebase:SCALe <scale_value> (see page 417)	:TIMebase:SCALe? (see page 417)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0   OFF}   {1   ON}} (see page 418)	:TIMebase:VERNier? (see page 418)	{0   1}
:TIMebase:WINDow:POSition <pos> (see page 419)	:TIMebase:WINDow:POSition? (see page 419)	<pos> ::= time from the trigger event to the zoomed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 420)	:TIMebase:WINDow:RANGe? (see page 420)	<range value> ::= range value in seconds in NR3 format for the zoomed window
:TIMebase:WINDow:SCALe <scale_value> (see page 421)	:TIMebase:WINDow:SCALe? (see page 421)	<scale_value> ::= scale value in seconds in NR3 format for the zoomed window



**Introduction to :TIMebase Commands** The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (zoomed) time bases.

### Reporting the Setup

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

### Return Format

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

```
:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

**:TIMEbase:MODE**

**C** (see [page 608](#))

**Command Syntax** :TIMEbase:MODE <value>  
 <value> ::= {MAIN | WINDow | XY | ROLL}

The :TIMEbase:MODE command sets the current time base. There are four time base modes:

- **MAIN** – The normal time base mode is the main time base. It is the default time base mode after the \*RST (Reset) command.
- **WINDow** – In the WINDow (zoomed or delayed) time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.
- **XY** – In the XY mode, the :TIMEbase:RANGe, :TIMEbase:POSition, and :TIMEbase:REFerence commands are not available. No measurements are available in this mode.
- **ROLL** – In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMEbase:REFerence selection changes to RIGHT.

**Query Syntax** :TIMEbase:MODE?

The :TIMEbase:MODE query returns the current time base mode.

**Return Format** <value><NL>  
 <value> ::= {MAIN | WIND | XY | ROLL}

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 412
  - ["\\*RST \(Reset\)"](#) on page 109
  - [":TIMEbase:RANGe"](#) on page 415
  - [":TIMEbase:POSition"](#) on page 414
  - [":TIMEbase:REFerence"](#) on page 416

**Example Code**

```
' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
myScope.WriteString ":TIMEBASE:MODE MAIN"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:TIMEbase:POSition**

**C** (see [page 608](#))

**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 412
  - [":TIMEbase:REFerence"](#) on page 416
  - [":TIMEbase:RANGe"](#) on page 415
  - [":TIMEbase:SCALE"](#) on page 417
  - [":TIMEbase:WINDow:POSition"](#) on page 419
  - [":TIMEbase:DELay"](#) on page 566

**:TIMEbase:RANGe**

**C** (see [page 608](#))

**Command Syntax** :TIMEbase:RANGe <range\_value>

<range\_value> ::= 50 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> ::= 50 ns through 500 s in NR3 format

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 412
  - [":TIMEbase:MODE"](#) on page 413
  - [":TIMEbase:SCALE"](#) on page 417
  - [":TIMEbase:WINDow:RANGe"](#) on page 420

**Example Code**

```
' TIME_RANGE - Sets the full scale horizontal time in seconds. The
' range value is 10 times the time per division.
myScope.WriteString ":TIM:RANG 2e-3" ' Set the time range to 0.002
seconds.
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:TIMEbase:REFeRence**

**C** (see [page 608](#))

**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 412
  - [":TIMEbase:MODE"](#) on page 413

**Example Code**

```
' TIME_REFERENCE - Possible values are LEFT, CENTER, or RIGHT.
' - LEFT sets the display reference one time division from the left.
' - CENTER sets the display reference to the center of the screen.
' - RIGHT sets the display reference one time division from the right.
t.
myScope.WriteString ":TIMEbase:REFeRence CENTER" ' Set reference to center.
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617



**:TIMEbase:SCALE**

**N** (see [page 608](#))

**Command Syntax** :TIMEbase:SCALE <scale\_value>

<scale\_value> ::= 5 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> ::= 5 ns through 50 s in NR3 format

- See Also**
- "[Introduction to :TIMEbase Commands](#)" on page 412
  - "[:TIMEbase:RANGE](#)" on page 415
  - "[:TIMEbase:WINDOW:SCALE](#)" on page 421
  - "[:TIMEbase:WINDOW:RANGE](#)" on page 420

## :TIMEbase:VERNier

**N** (see [page 608](#))

**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 412

**:TIMEbase:WINDow:POSition**

**C** (see [page 608](#))

**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 412
  - [":TIMEbase:MODE"](#) on page 413
  - [":TIMEbase:POSition"](#) on page 414
  - [":TIMEbase:RANGe"](#) on page 415
  - [":TIMEbase:SCALe"](#) on page 417
  - [":TIMEbase:WINDow:RANGe"](#) on page 420
  - [":TIMEbase:WINDow:SCALe"](#) on page 421

**:TIMEbase:WINDow:RANGe**

**C** (see [page 608](#))

**Command Syntax** :TIMEbase:WINDow:RANGe <range value>

<range value> ::= range value in seconds in NR3 format

The :TIMEbase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the zoomed (delayed) window. The range is 10 times the current zoomed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMEbase:RANGe value.

**Query Syntax** :TIMEbase:WINDow:RANGe?

The :TIMEbase:WINDow:RANGe? query returns the current window timebase range setting.

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

<value> ::= range value in seconds

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 412
  - [":TIMEbase:RANGe"](#) on page 415
  - [":TIMEbase:POSition"](#) on page 414
  - [":TIMEbase:SCALE"](#) on page 417

**:TIMEbase:WINDow:SCALE**

**N** (see [page 608](#))

**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?

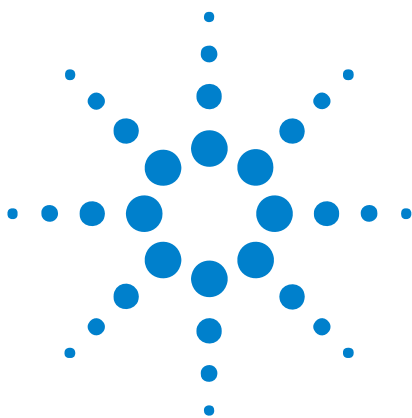
The :TIMEbase:WINDow:SCALE? query returns the current zoomed window scale setting.

**Return Format** <scale\_value><NL>

<scale\_value> ::= current seconds per division for the zoomed window

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 412
  - [":TIMEbase:RANGE"](#) on page 415
  - [":TIMEbase:POSition"](#) on page 414
  - [":TIMEbase:SCALE"](#) on page 417
  - [":TIMEbase:WINDow:RANGE"](#) on page 420





## 25 :TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "Introduction to :TRIGger Commands" on page 423
- "General :TRIGger Commands" on page 425
- ":TRIGger[:EDGE] Commands" on page 433
- ":TRIGger:GLITCh Commands" on page 439 (Pulse Width trigger)
- ":TRIGger:PATtern Commands" on page 448
- ":TRIGger:TV Commands" on page 453

### 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 432) 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 430).

- **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. You can also trigger on a specified time duration of a pattern.
- **TV triggering**— is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than  $\frac{1}{2}$  division of sync amplitude with any analog channel as the trigger source.

### Reporting the Setup

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

### Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a \*RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;  
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```



## General :TRIGger Commands

**Table 61** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}   {1   ON}} (see <a href="#">page 426</a> )	:TRIGger:HFReject? (see <a href="#">page 426</a> )	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see <a href="#">page 427</a> )	:TRIGger:HOLDoff? (see <a href="#">page 427</a> )	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:LEVel:HIGH <level>, <source> (see <a href="#">page 428</a> )	:TRIGger:LEVel:HIGH? <source> (see <a href="#">page 428</a> )	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:LEVel:LOW <level>, <source> (see <a href="#">page 429</a> )	:TRIGger:LEVel:LOW? <source> (see <a href="#">page 429</a> )	<level> ::= .75 x full-scale voltage from center screen in NR3 format. <source> ::= CHANnel<n> <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:MODE <mode> (see <a href="#">page 430</a> )	:TRIGger:MODE? (see <a href="#">page 430</a> )	<mode> ::= {EDGE   GLITCh   PATtern   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 431</a> )	:TRIGger:NREject? (see <a href="#">page 431</a> )	{0   1}
:TRIGger:SWEep <sweep> (see <a href="#">page 432</a> )	:TRIGger:SWEep? (see <a href="#">page 432</a> )	<sweep> ::= {AUTO   NORMal}

**:TRIGger:HFReject**

**C** (see [page 608](#))

**Command Syntax** :TRIGger:HFReject <value>  
 <value> ::= {{0 | OFF} | {1 | ON}}

The :TRIGger:HFReject command turns the high frequency reject filter off and on. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use this filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.

**Query Syntax** :TRIGger:HFReject?

The :TRIGger:HFReject? query returns the current high frequency reject filter mode.

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

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger\[:EDGE\]:REJECT](#)" on page 436

**:TRIGger:HOLDoff**

**C** (see [page 608](#))

**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 423

## :TRIGger:LEVel:HIGH

**N** (see [page 608](#))

**Command Syntax** :TRIGger:LEVel:HIGH <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format  
for internal triggers

<source> ::= CHANnel<n>

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

The :TRIGger:LEVel:HIGH command sets the high trigger voltage level voltage for the specified source.

**Query Syntax** :TRIGger:LEVel:HIGH? <source>

The :TRIGger:LEVel:HIGH? query returns the high trigger voltage level for the specified source.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:LEVel:LOW"](#) on page 429
  - [":TRIGger\[:EDGE\]:SOURCE"](#) on page 438

**:TRIGger:LEVel:LOW**

**N** (see [page 608](#))

**Command Syntax** :TRIGger:LEVel:LOW <level>, <source>

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format  
for internal triggers

<source> ::= CHANnel<n>

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

The :TRIGger:LEVel:LOW command sets the low trigger voltage level voltage for the specified source.

**Query Syntax** :TRIGger:LEVel:LOW? <source>

The :TRIGger:LEVel:LOW? query returns the low trigger voltage level for the specified source.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:LEVel:HIGH"](#) on page 428
  - [":TRIGger\[:EDGE\]:SOURce"](#) on page 438

**:TRIGger:MODE**

**C** (see [page 608](#))

**Command Syntax** :TRIGger:MODE <mode>  
 <mode> ::= {EDGE | GLITCh | PATTern | TV}

The :TRIGger:MODE command selects the trigger mode (trigger type).

**Query Syntax** :TRIGger:MODE?

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMEbase:MODE is ROLL or XY, the query returns "NONE".

**Return Format** <mode><NL>  
 <mode> ::= {EDGE | GLIT | PATT | TV}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger:SWEep](#)" on page 432
  - "[:TIMEbase:MODE](#)" on page 413

**Example Code**

```
' TRIGGER_MODE - Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:TRIGger:NREJect**

**C** (see [page 608](#))

**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 423

**:TRIGger:SWEEp**

**C** (see [page 608](#))

**Command Syntax** :TRIGger:SWEEp <sweep>  
 <sweep> ::= {AUTO | NORMal}

The :TRIGger:SWEEp command selects the trigger sweep mode.

When AUTO sweep mode is selected, a baseline is displayed in the absence of a signal. If a signal is present but the oscilloscope is not triggered, the unsynchronized signal is displayed instead of a baseline.

When NORMal sweep mode is selected and no trigger is present, the instrument does not sweep, and the data acquired on the previous trigger remains on the screen.

**NOTE**

This feature is called "Mode" on the instrument's front panel.

**Query Syntax** :TRIGger:SWEEp?

The :TRIGger:SWEEp? query returns the current trigger sweep mode.

**Return Format** <sweep><NL>  
 <sweep> ::= current trigger sweep mode

**See Also** • ["Introduction to :TRIGger Commands"](#) on page 423



## :TRIGger[:EDGE] Commands

**Table 62** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LFReject} (see <a href="#">page 434</a> )	:TRIGger[:EDGE]:COUPling? (see <a href="#">page 434</a> )	{AC   DC   LFReject}
:TRIGger[:EDGE]:LEVel <level> [, <source>] (see <a href="#">page 435</a> )	:TRIGger[:EDGE]:LEVel? [<source>] (see <a href="#">page 435</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. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n>   EXTErnal} for DSO models <source> ::= {CHANnel<n>   DIGital<d>   EXTErnal } for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LFReject   HFReject} (see <a href="#">page 436</a> )	:TRIGger[:EDGE]:REJect? (see <a href="#">page 436</a> )	{OFF   LFReject   HFReject}
:TRIGger[:EDGE]:SLOPe <polarity> (see <a href="#">page 437</a> )	:TRIGger[:EDGE]:SLOPe? (see <a href="#">page 437</a> )	<polarity> ::= {POSitive   NEGative   EITHer   ALTErnate}
:TRIGger[:EDGE]:SOURC e <source> (see <a href="#">page 438</a> )	:TRIGger[:EDGE]:SOURC e? (see <a href="#">page 438</a> )	<source> ::= {CHANnel<n>   EXTErnal   LINE   WGEN} for the DSO models <source> ::= {CHANnel<n>   DIGital<d>   EXTErnal   LINE   WGEN} for the MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format

**:TRIGger[:EDGE]:COUPLing**

**C** (see [page 608](#))

**Command Syntax** :TRIGger[:EDGE]:COUPLing <coupling>  
 <coupling> ::= {AC | DC | LFReject}

The :TRIGger[:EDGE]:COUPLing command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LFReject.

- AC coupling places a high-pass filter (10 Hz for analog channels, and 3.5 Hz for all External trigger inputs) in the trigger path, removing dc offset voltage from the trigger waveform. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset.
- LFReject coupling places a 50 KHz high-pass filter in the trigger path.
- DC coupling allows dc and ac signals into the trigger path.

**NOTE**

The :TRIGger[:EDGE]:COUPLing and the :TRIGger[:EDGE]:REJect selections are coupled. Changing the setting of the :TRIGger[:EDGE]:REJect can change the COUPLing setting.

**Query Syntax** :TRIGger[:EDGE]:COUPLing?

The :TRIGger[:EDGE]:COUPLing? query returns the current coupling selection.

**Return Format** <coupling><NL>  
 <coupling> ::= {AC | DC | LFR}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger:MODE](#)" on page 430
  - "[:TRIGger\[:EDGE\]:REJect](#)" on page 436

**:TRIGger[:EDGE]:LEVel**

**C** (see [page 608](#))

**Command Syntax** :TRIGger[:EDGE]:LEVel <level>  
 <level> ::= <level>[,<source>]  
 <level> ::= 0.75 x full-scale voltage from center screen in NR3 format  
 for internal triggers  
 <level> ::= ±(external range setting) in NR3 format  
 for external triggers  
 <level> ::= ±8 V for digital channels (MSO models)  
 <source> ::= {CHANnel<n> | EXTernal} for the DSO models  
 <source> ::= {CHANnel<n> | DIGital<d> | EXTernal}  
 for the MSO models  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

**NOTE**

If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

**Query Syntax** :TRIGger[:EDGE]:LEVel? [<source>]

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

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

- See Also**
- "Introduction to :TRIGger Commands" on page 423
  - ":TRIGger[:EDGE]:SOURce" on page 438
  - ":EXTernal:RANGe" on page 236
  - ":POD<n>:THReshold" on page 368
  - ":DIGital<d>:THReshold" on page 223

**:TRIGger[:EDGE]:REJect**

**C** (see [page 608](#))

**Command Syntax** :TRIGger[:EDGE]:REJect <reject>  
 <reject> ::= {OFF | LFReject | HFReject}

The :TRIGger[:EDGE]:REJect command turns the low-frequency or high-frequency reject filter on or off. You can turn on one of these filters at a time.

- The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use the high frequency reject filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.
- The low frequency reject filter adds a 50 kHz high-pass filter in series with the trigger waveform to remove any unwanted low frequency components from a trigger waveform, such as power line frequencies, that can interfere with proper triggering.

**NOTE**

The :TRIGger[:EDGE]:REJect and the :TRIGger[:EDGE]:COUPling selections are coupled. Changing the setting of the :TRIGger[:EDGE]:COUPling can change the COUPling setting.

**Query Syntax** :TRIGger[:EDGE]:REJect?

The :TRIGger[:EDGE]:REJect? query returns the current status of the reject filter.

**Return Format** <reject><NL>  
 <reject> ::= {OFF | LFR | HFR}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger:HFReject](#)" on page 426
  - "[:TRIGger\[:EDGE\]:COUPling](#)" on page 434

**:TRIGger[:EDGE]:SLOPe**

**C** (see [page 608](#))

**Command Syntax** :TRIGger[:EDGE]:SLOPe <slope>

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

The :TRIGger[:EDGE]:SLOPe command specifies the slope of the edge for the trigger. The SLOPe command is not valid in TV trigger mode. Instead, use :TRIGger:TV:POLarity to set the polarity in TV trigger mode.

**Query Syntax** :TRIGger[:EDGE]:SLOPe?

The :TRIGger[:EDGE]:SLOPe? query returns the current trigger slope.

**Return Format** <slope><NL>

<slope> ::= {NEG | POS | EITH | ALT}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger:MODE](#)" on page 430
  - "[:TRIGger:TV:POLarity](#)" on page 456

**Example Code**

```
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:TRIGger[:EDGE]:SOURce**

**C** (see [page 608](#))

**Command Syntax** :TRIGger[:EDGE]:SOURce <source>  
 <source> ::= {CHANnel<n> | EXTernal | LINE | WGEN} for the DSO models  
 <source> ::= {CHANnel<n> | DIGital<d> | EXTernal | LINE | WGEN}  
 for the MSO models

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

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

The :TRIGger[:EDGE]:SOURce command selects the input that produces the trigger.

- EXTernal – triggers on the rear panel EXT TRIG IN signal.
- LINE – triggers at the 50% level of the rising or falling edge of the AC power source signal.
- WGEN – triggers at the 50% level of the rising edge of the waveform generator output signal.

**Query Syntax** :TRIGger[:EDGE]:SOURce?

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

**Return Format** <source><NL>  
 <source> ::= {CHAN<n> | EXT | LINE | WGEN | NONE} for the DSO models  
 <source> ::= {CHAN<n> | DIG<d> | EXTernal | LINE | WGEN | NONE}  
 for the MSO models

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:MODE"](#) on page 430

**Example Code**

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces the
edge trigger. Any channel can be selected.
myScope.WriteString ":TRIGger:EDGE:SOURce CHANnel1"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

## :TRIGger:GLITCh Commands

**Table 63** :TRIGger:GLITCh Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITCh:GREAt erthan <greater_than_time>[s uffix] (see <a href="#">page 441</a> )	:TRIGger:GLITCh:GREAt erthan? (see <a href="#">page 441</a> )	<greater_than_time> ::= floating-point number in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITCh:LESSt han <less_than_time>[suff ix] (see <a href="#">page 442</a> )	:TRIGger:GLITCh:LESSt han? (see <a href="#">page 442</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 443</a> )	:TRIGger:GLITCh:LEVel ? (see <a href="#">page 443</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers (DSO models), <level> ::= ±(external range setting) in NR3 format. For digital channels (MSO models), <level> ::= ±8 V. <source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital<d>} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format
:TRIGger:GLITCh:POLAr ity <polarity> (see <a href="#">page 444</a> )	:TRIGger:GLITCh:POLAr ity? (see <a href="#">page 444</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITCh:QUALi fier <qualifier> (see <a href="#">page 445</a> )	:TRIGger:GLITCh:QUALi fier? (see <a href="#">page 445</a> )	<qualifier> ::= {GREaterthan   LESSthan   RANGE}

**Table 63** :TRIGger:GLITCh Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITCh:RANGe <less_than_time>[suffix], <greater_than_time>[suffix] (see <a href="#">page 446</a> )	:TRIGger:GLITCh:RANGe? (see <a href="#">page 446</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 447</a> )	:TRIGger:GLITCh:SOURCe? (see <a href="#">page 447</a> )	<source> ::= {CHANnel<n>   DIGital<d>} <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format



**:TRIGger:GLITch:GREaterthan**

**N** (see [page 608](#))

**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 423
  - "[:TRIGger:GLITch:SOURce](#)" on page 447
  - "[:TRIGger:GLITch:QUALifier](#)" on page 445
  - "[:TRIGger:MODE](#)" on page 430

**:TRIGger:GLITch:LESSthan**

**N** (see [page 608](#))

**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 423
  - [":TRIGger:GLITch:SOURce"](#) on page 447
  - [":TRIGger:GLITch:QUALifier"](#) on page 445
  - [":TRIGger:MODE"](#) on page 430

**:TRIGger:GLITch:LEVel**

**N** (see [page 608](#))

**Command Syntax** :TRIGger:GLITch:LEVel <level\_argument>  
 <level\_argument> ::= <level>[, <source>]  
 <level> ::= .75 x full-scale voltage from center screen in NR3 format  
 for internal triggers  
 <level> ::= ±(external range setting) in NR3 format  
 for external triggers (DSO models)  
 <level> ::= ±8 V for digital channels (MSO models)  
 <source> ::= {CHANnel<n> | EXTernal} for DSO models  
 <source> ::= {CHANnel<n> | DIGital<d>} for MSO models  
 <n> ::= 1 to (# analog channels) in NR1 format  
 <d> ::= 0 to (# digital channels - 1) in NR1 format

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

**Query Syntax** :TRIGger:GLITch:LEVel?

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:MODE"](#) on page 430
  - [":TRIGger:GLITch:SOURce"](#) on page 447
  - [":EXTernal:RANGe"](#) on page 236

## :TRIGger:GLITch:POLarity

**N** (see [page 608](#))

**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 423
  - "[:TRIGger:MODE](#)" on page 430
  - "[:TRIGger:GLITch:SOURce](#)" on page 447

**:TRIGger:GLITCh:QUALifier**

**N** (see [page 608](#))

**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 423
  - "[:TRIGger:GLITCh:SOURce](#)" on page 447
  - "[:TRIGger:MODE](#)" on page 430



**:TRIGger:GLITCh:SOURce**

**N** (see [page 608](#))

**Command Syntax** :TRIGger:GLITCh:SOURce <source>

<source> ::= {DIGital<d> | CHANnel<n>}

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

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

The :TRIGger:GLITCh:SOURce command selects the channel that produces the pulse width trigger.

**Query Syntax** :TRIGger:GLITCh:SOURce?

The :TRIGger:GLITCh:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE".

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:MODE"](#) on page 430
  - [":TRIGger:GLITCh:LEVel"](#) on page 443
  - [":TRIGger:GLITCh:POLarity"](#) on page 444
  - [":TRIGger:GLITCh:QUALifier"](#) on page 445
  - [":TRIGger:GLITCh:RANGE"](#) on page 446

**Example Code** • ["Example Code"](#) on page 438

**:TRIGger:PATtern Commands****Table 64** :TRIGger:PATtern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATtern <string>[,<edge_source>,<edge>] (see <a href="#">page 449</a> )	:TRIGger:PATtern? (see <a href="#">page 450</a> )	<string> ::= "nn...n" where n ::= {0   1   X   R   F} when <base> = ASCII <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F   X   \$} when <base> = HEX <edge_source> ::= {CHANnel<n>   NONE} for DSO models <edge_source> ::= {CHANnel<n>   DIGital<d>   NONE} for MSO models <n> ::= 1 to (# analog channels) in NR1 format <d> ::= 0 to (# digital channels - 1) in NR1 format <edge> ::= {POSitive   NEGative}
:TRIGger:PATtern:FORM at <base> (see <a href="#">page 451</a> )	:TRIGger:PATtern:FORM at? (see <a href="#">page 451</a> )	<base> ::= {ASCII   HEX}
:TRIGger:PATtern:QUAL ifier <qualifier> (see <a href="#">page 452</a> )	:TRIGger:PATtern:QUAL ifier? (see <a href="#">page 452</a> )	<qualifier> ::= ENTERed



**:TRIGger:PATtern**

**C** (see [page 608](#))

**Command Syntax** :TRIGger:PATtern <pattern>

<pattern> ::= <string>[,<edge\_source>,<edge>]

<string> ::= "nn...n" where n ::= {0 | 1 | X | R | F} when  
<base> = ASCii

<string> ::= "0xnn...n" where n ::= {0,...,9 | A,...,F | X | \$} when  
<base> = HEX

<edge\_source> ::= {CHANnel<n> | NONE} for DSO models

<edge\_source> ::= {CHANnel<n> | DIGital<d>  
| NONE} for MSO models

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

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

<edge> ::= {POSitive | NEGative}

The :TRIGger:PATtern command specifies the channel values to be used in the pattern trigger.

In the <string> parameter, each bit corresponds to a channel as described in the following table:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog + 8 digital channels (mixed-signal)</b>	Bits 0 through 7 - digital channels 0 through 7. Bits 8 through 11 - analog channels 4 through 1.
<b>2 analog + 8 digital channels (mixed-signal)</b>	Bits 0 through 7 - digital channels 0 through 7. Bits 8 and 9 - analog channels 2 and 1.
<b>4 analog channels only</b>	Bits 0 through 3 - analog channels 4 through 1.
<b>2 analog channels only</b>	Bits 0 and 1 - analog channels 2 and 1.

The format of the <string> parameter depends on the :TRIGger:PATtern:FORMat command setting:

- When the format is ASCii, the string looks just like the string you see on the oscilloscope's front panel, made up of 0, 1, X (don't care), R (rising edge), and F (falling edge) characters.
- When the format is HEX, the string begins with "0x" and contains hex digit characters or X (don't care for all four bits in the nibble).

With the hex format string, you can use the <edge\_source> and <edge> parameters to specify an edge on one of the channels.

**NOTE**

The optional <edge\_source> and <edge> parameters should be sent together or not at all. The edge can be specified in the ASCII <string> parameter. If the edge source and edge parameters are used, they take precedence.

---

You can only specify an edge on one channel. When an edge is specified, the :TRIGger:PATtern:QUALifier does not apply.

**Query Syntax** :TRIGger:PATtern?

The :TRIGger:PATtern? query returns the pattern string, edge source, and edge.

**Return Format** <string>,<edge\_source>,<edge><NL>

- See Also**
- ["Introduction to :TRIGger Commands" on page 423](#)
  - [":TRIGger:PATtern:FORMat" on page 451](#)
  - [":TRIGger:PATtern:QUALifier" on page 452](#)
  - [":TRIGger:MODE" on page 430](#)

## :TRIGger:PATtern:FORMat

**N** (see [page 608](#))

**Command Syntax** :TRIGger:PATtern:FORMat <base>

<base> ::= {ASCIi | HEX}

The :TRIGger:PATtern:FORMat command sets the entry (and query) number base used by the :TRIGger:PATtern command. The default <base> is ASCii.

**Query Syntax** :TRIGger:PATtern:FORMat?

The :TRIGger:PATtern:FORMat? query returns the currently set number base for pattern trigger patterns.

**Return Format** <base><NL>

<base> ::= {ASC | HEX}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 423
  - "[:TRIGger:PATtern](#)" on page 449

## :TRIGger:PATtern:QUALifier

**N** (see [page 608](#))

**Command Syntax** :TRIGger:PATtern:QUALifier <qualifier>  
<qualifier> ::= ENTERed

The :TRIGger:PATtern:QUALifier command qualifies when the trigger occurs.

In the InfiniiVision 2000 X-Series oscilloscopes, the trigger always occurs when the pattern is entered.

**Query Syntax** :TRIGger:PATtern:QUALifier?

The :TRIGger:PATtern:QUALifier? query returns the trigger duration qualifier.

**Return Format** <qualifier><NL>

**See Also** • ["Introduction to :TRIGger Commands"](#) on page 423

## :TRIGger:TV Commands

**Table 65** :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 454)	:TRIGger:TV:LINE? (see page 454)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 455)	:TRIGger:TV:MODE? (see page 455)	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LFIELD1   LFIELD2   LALTERNATE}
:TRIGger:TV:POLarity <polarity> (see page 456)	:TRIGger:TV:POLarity? (see page 456)	<polarity> ::= {POSitive   NEGative}
:TRIGger:TV:SOURce <source> (see page 457)	:TRIGger:TV:SOURce? (see page 457)	<source> ::= {CHANnel<n>} <n> ::= 1 to (# analog channels) in NR1 format
:TRIGger:TV:STANdard <standard> (see page 458)	:TRIGger:TV:STANdard? (see page 458)	<standard> ::= {NTSC   PAL   PALM   SECam}

**:TRIGger:TV:LINE**

**N** (see [page 608](#))

**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 66** TV Trigger Line Number Limits

TV Standard	Mode		
	LField1	LField2	LALternate
<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

**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 423
  - [":TRIGger:TV:STANdard"](#) on page 458
  - [":TRIGger:TV:MODE"](#) on page 455

**:TRIGger:TV:MODE**

**N** (see [page 608](#))

**Command Syntax** :TRIGger:TV:MODE <mode>

```
<mode> ::= {FIEld1 | FIEld2 | AFIElds | ALINes
            | LFIEld1 | LFIEld2 | LALTernate}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIElds	ALLFields, ALLFLDS
ALINes	ALLLines
LFIEld1	LINEF1, LINEFIELD1
LFIEld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt

**Query Syntax** :TRIGger:TV:MODE?

The :TRIGger:TV:MODE? query returns the TV trigger mode.

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

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LFI1 | LFI2 | LALT}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:TV:STANdard"](#) on page 458
  - [":TRIGger:MODE"](#) on page 430

## :TRIGger:TV:POLarity

**N** (see [page 608](#))

**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 423
  - [":TRIGger:MODE"](#) on page 430
  - [":TRIGger:TV:SOURce"](#) on page 457



**:TRIGger:TV:SOURce**

**N** (see [page 608](#))

**Command Syntax** :TRIGger:TV:SOURce <source>

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

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

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

**Query Syntax** :TRIGger:TV:SOURce?

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

**Return Format** <source><NL>

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 423
  - [":TRIGger:MODE"](#) on page 430
  - [":TRIGger:TV:POLarity"](#) on page 456

**Example Code**

- ["Example Code"](#) on page 438

## :TRIGger:TV:STANdard

**N** (see [page 608](#))

**Command Syntax** :TRIGger:TV:STANdard <standard>  
<standard> ::= {NTSC | PALM | PAL | SECam}

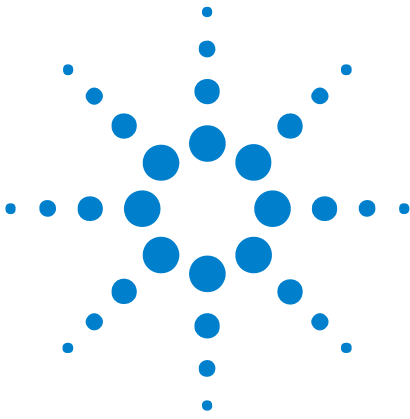
The :TRIGger:TV:STANdard command selects the video standard:

- NTSC
- PAL
- PAL-M
- SECAM

**Query Syntax** :TRIGger:TV:STANdard?

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

**Return Format** <standard><NL>  
<standard> ::= {NTSC | PALM | PAL | SEC}



## 26 :WAVeform Commands

Provide access to waveform data. See "[Introduction to :WAVeform Commands](#)" on page 461.

**Table 67** :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see <a href="#">page 467</a> )	:WAVeform:BYTeorder? (see <a href="#">page 467</a> )	<value> ::= {LSBFirst   MSBFirst}
n/a	:WAVeform:COUNT? (see <a href="#">page 468</a> )	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see <a href="#">page 469</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 471</a> )	:WAVeform:FORMat? (see <a href="#">page 471</a> )	<value> ::= {WORD   BYTE   ASCII}
:WAVeform:POINTs <# points> (see <a href="#">page 472</a> )	:WAVeform:POINTs? (see <a href="#">page 472</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}



**Table 67** :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:POINTs:MODE <points_mode> (see page 474)	:WAVEform:POINTs:MODE ? (see page 474)	<points_mode> ::= {NORMAL   MAXimum   RAW}
n/a	:WAVEform:PREamble? (see page 476)	<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>• 3 for AVERAge type</li><li>• 4 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:C OUNT? (see page 479)	<count> ::= an integer from 2 to 1000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:T TAG? (see page 480)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see page 481)	:WAVEform:SOURce? (see page 481)	<source> ::= {CHANnel<n>   FUNCTION   MATH} for DSO models <source> ::= {CHANnel<n>   POD{1   2}   BUS{1   2}   FUNCTION   MATH} for MSO models <n> ::= 1 to (# analog channels) in NR1 format
:WAVEform:SOURce:SUBS ource <subsource> (see page 485)	:WAVEform:SOURce:SUBS ource? (see page 485)	<subsource> ::= {SUB0   RX   MOSI}
n/a	:WAVEform:TYPE? (see page 486)	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVEform:UNSigned {{0   OFF}   {1   ON}} (see page 487)	:WAVEform:UNSigned? (see page 487)	{0   1}

**Table 67** :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:VIEW <view> (see <a href="#">page 488</a> )	:WAVEform:VIEW? (see <a href="#">page 488</a> )	<view> ::= {MAIN}
n/a	:WAVEform:XINCrement? (see <a href="#">page 489</a> )	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVEform:XORigin? (see <a href="#">page 490</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVEform:XREFerence? (see <a href="#">page 491</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVEform:YINCrement? (see <a href="#">page 492</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVEform:YORigin? (see <a href="#">page 493</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVEform:YREFerence? (see <a href="#">page 494</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 469](#)) and :WAVEform:PREAmble (see [page 476](#)). The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

#### Data Acquisition Types

There are four types of waveform acquisitions that can be selected for analog channels with the :ACQUIRE:TYPE command (see [page 169](#)): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always acquired using NORMal. When the data is acquired using the :DIGitize command (see [page 133](#)) or :RUN command (see [page 150](#)), 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 469](#)) 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 162](#)).

### Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINTS command (see [page 472](#)). If :WAVeform:POINTS MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes or as many as 8,000,000 for a digital channel on the mixed signal oscilloscope. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVeform:POINTS may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVeform:POINTS must be an even divisor of 1,000 or be set to MAXimum. :WAVeform:POINTS determines the increment between time buckets that will be transferred. If POINTs = MAXimum, the data cannot be decimated. For example:

- :WAVeform:POINTS 1000 – returns time buckets 0, 1, 2, 3, 4 ..., 999.
- :WAVeform:POINTS 500 – returns time buckets 0, 2, 4, 6, 8 ..., 998.
- :WAVeform:POINTS 250 – returns time buckets 0, 4, 8, 12, 16 ..., 996.
- :WAVeform:POINTS 100 – returns time buckets 0, 10, 20, 30, 40 ..., 990.

### Analog Channel Data

#### NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket  $n - 1$ , where  $n$

is the number returned by the :WAVeform:POINts? query (see [page 472](#)). 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 160](#)). 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 472](#)). 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 472](#)). 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 169](#)), the value returned by the :WAVeform:XINCrement query (see [page 489](#)) should be doubled to find the time difference between the min-max pairs.

### **HRESolution Data**

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

### **Data Conversion**

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

$$\text{voltage} = [(\text{data value} - \text{yreference}) * \text{yincrement}] + \text{yorigin}$$

If the :WAVeform:FORMat data format is ASCii (see [page 471](#)), the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

$$\text{time} = [(\text{data point number} - \text{xreference}) * \text{xincrement}] + \text{xorigin}$$

This would result in the following calculation for time bucket 3:

$$\text{time} = [(3 - 0) * 2 \text{ ns}] + 16 \text{ ns} = 22 \text{ ns}$$

In :ACQUIRE:TYPE PEAK mode (see [page 169](#)), because data is acquired in max-min pairs, modify the previous time formula to the following:

$$\text{time} = [(\text{data pair number} - \text{xreference}) * \text{xincrement} * 2] + \text{xorigin}$$

### Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCii (see ":WAVeform:FORMat" on [page 471](#)). 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 487](#)) 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 471](#)) 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 467](#)) and :WAVeform:UNSigned (see [page 487](#)) have no effect when the format is ASCii.

#### Data Format for Transfer - WORD format

WORD format (see [":WAVeform:FORMat"](#) on [page 471](#)) 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 472](#)). 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 467](#)) 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 471](#)) 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 467](#)) has no effect when the data format is BYTE.

**Digital Channel Data (MSO models only)**

The waveform record for digital channels is similar to that of analog channels. The main difference is that the data points represent either DIGital0,...,7 (POD1), DIGital8,...,15 (POD2), or any grouping of digital channels (BUS1 or BUS2).

For digital channels, :WAVeform:UNSigned (see [page 487](#)) must be set to ON.

**Digital Channel POD Data Format**

Data for digital channels is only available in groups of 8 bits (Pod1 = D0 - D7, Pod2 = D8 - D15). The bytes are organized as:

:WAVeform:SOURce	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
POD1	D7	D6	D5	D4	D3	D2	D1	D0
POD2	D15	D14	D13	D12	D11	D10	D9	D8

If the :WAVeform:FORMat is WORD (see [page 471](#)) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see [page 467](#)) controls which byte is 0.

If a digital channel is not displayed, its bit value in the pod data byte is not defined.

**Digital Channel BUS Data Format**

Digital channel BUS definitions can include any or all of the digital channels. Therefore, data is always returned as 16-bit values. :BUS commands (see [page 171](#)) are used to select the digital channels for a bus.

**Reporting the Setup**

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

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS
NONE
```

**:WAVeform:BYTeorder**

**C** (see [page 608](#))

**Command Syntax** :WAVeform:BYTeorder <value>  
 <value> ::= {LSBFirst | MSBFirst}

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFirst sets the most significant byte to be transmitted first. The parameter LSBFirst sets the least significant byte to be transmitted first. This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected. The default setting is LSBFirst.

**Query Syntax** :WAVeform:BYTeorder?

The :WAVeform:BYTeorder query returns the current output sequence.

**Return Format** <value><NL>  
 <value> ::= {LSBF | MSBF}

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 461
  - [":WAVeform:DATA"](#) on page 469
  - [":WAVeform:FORMat"](#) on page 471
  - [":WAVeform:PREamble"](#) on page 476

- Example Code**
- ["Example Code"](#) on page 482
  - ["Example Code"](#) on page 477

## :WAVeform:COUNT

**C** (see [page 608](#))

**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 461
  - "[:ACQUIRE:COUNT](#)" on page 160
  - "[:ACQUIRE:TYPE](#)" on page 169

**:WAVeform:DATA**

**C** (see [page 608](#))

**Query Syntax** :WAVeform:DATA?

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSigned, :WAVeform:BYTeorder, :WAVeform:FORMat, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINts command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 – Hole. Holes are locations where data has not yet been acquired.

Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.

- 0x01 or 0x0001 – Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFFFF – Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

**Return Format** <binary block data><NL>

- See Also**
- For a more detailed description of the data returned for different acquisition types, see: "[Introduction to :WAVeform Commands](#)" on [page 461](#)
  - "[:WAVeform:UNSigned](#)" on [page 487](#)
  - "[:WAVeform:BYTeorder](#)" on [page 467](#)
  - "[:WAVeform:FORMat](#)" on [page 471](#)
  - "[:WAVeform:POINts](#)" on [page 472](#)
  - "[:WAVeform:PREamble](#)" on [page 476](#)
  - "[:WAVeform:SOURce](#)" on [page 481](#)
  - "[:WAVeform:TYPE](#)" on [page 486](#)

**Example Code**

```
' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.
' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
```

```

'
' <header><waveform_data><NL>
'
' Where:
' <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 _
            + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) _
            * sngYIncrement + sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) _
            * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

**:WAVEform:FORMat**

**C** (see [page 608](#))

**Command Syntax** :WAVEform:FORMat <value>

<value> ::= {WORD | BYTE | ASCii}

The :WAVEform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

ASCII formatted data is transferred ASCII text.

- WORD formatted data transfers 16-bit data as two bytes. The :WAVEform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVEform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

**Query Syntax** :WAVEform:FORMat?

The :WAVEform:FORMat query returns the current output format for the transfer of waveform data.

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

<value> ::= {WORD | BYTE | ASC}

- See Also**
- ["Introduction to :WAVEform Commands"](#) on page 461
  - [":WAVEform:BYTeorder"](#) on page 467
  - [":WAVEform:SOURce"](#) on page 481
  - [":WAVEform:DATA"](#) on page 469
  - [":WAVEform:PREamble"](#) on page 476

**Example Code** • ["Example Code"](#) on page 482

**:WAVeform:POINts**

**C** (see [page 608](#))

**Command Syntax** :WAVeform:POINts <# points>

```
<# points> ::= {100 | 250 | 500 | 1000 | <points mode>}
              if waveform points mode is NORMAl

<# points> ::= {100 | 250 | 500 | 1000 | 2000 | 5000 | 10000 | 20000
              | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
              | 4000000 | 8000000 | <points mode>}
              if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMAl | MAXimum | RAW}
```

**NOTE**

The <points\_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog or digital sources, the records that can be transferred depend on the waveform points mode. The maximum number of points returned for math (function) waveforms is determined by the NORMAl waveform points mode. See the :WAVeform:POINts:MODE command (see [page 474](#)) for more information.

Only data visible on the display will be returned.

**Query Syntax** :WAVeform:POINts?

The :WAVeform:POINts query returns the number of waveform points to be transferred when using the :WAVeform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVeform:POINts:MODE command (see [page 474](#)) 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 | 5000 | 10000 | 20000
              | 50000 | 100000 | 200000 | 500000 | 1000000 | 2000000
              | 4000000 | 8000000 | <maximum # points>}
              if waveform points mode is MAXimum or RAW
```

**NOTE**

If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.



- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 461
  - [":ACQUIRE:POINTS"](#) on page 162
  - [":WAVEFORM:DATA"](#) on page 469
  - [":WAVEFORM:SOURCE"](#) on page 481
  - [":WAVEFORM:VIEW"](#) on page 488
  - [":WAVEFORM:PREAmble"](#) on page 476
  - [":WAVEFORM:POINTS:MODE"](#) on page 474

**Example Code**

```
' WAVE_POINTS - Specifies the number of points to be transferred  
' using the ":WAVEFORM:DATA?" query.  
myScope.WriteString ":WAVEFORM:POINTS 1000"
```

See complete example programs at: [Chapter 34](#), "Programming Examples," starting on page 617

**:WAVEform:POINts:MODE**

**N** (see [page 608](#))

**Command Syntax** :WAVEform:POINts:MODE <points\_mode>

<points\_mode> ::= {NORMal | MAXimum | RAW}

The :WAVEform:POINts:MODE command sets the data record to be transferred with the :WAVEform:DATA? query.

For the analog or digital sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQUIRE:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 62,500-point (maximum) representation of the raw acquisition record. The measurement record can be retrieved from any source.

If the <points\_mode> is NORMal the measurement record is retrieved.

If the <points\_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points\_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

**Considerations  
for MAXimum or  
RAW data  
retrieval**

- The instrument must be stopped (see the :STOP command (see [page 154](#)) or the :DIGitize command (see [page 133](#)) in the root subsystem) in order to return more than the *measurement record*.
- :TIMEbase:MODE must be set to MAIN.
- :ACQUIRE:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQUIRE:COUNT must be set to 1 in order to return more than the *measurement record*.
- MAXimum or RAW will allow up to 100,000 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 461
  - [":WAVEform:DATA"](#) on page 469
  - [":ACQUIRE:POINTs"](#) on page 162
  - [":WAVEform:VIEW"](#) on page 488
  - [":WAVEform:PREamble"](#) on page 476
  - [":WAVEform:POINTs"](#) on page 472
  - [":TIMEbase:MODE"](#) on page 413
  - [":ACQUIRE:TYPE"](#) on page 169
  - [":ACQUIRE:COUNt"](#) on page 160

**:WAVEform:PREamble**

**C** (see page 608)

**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.

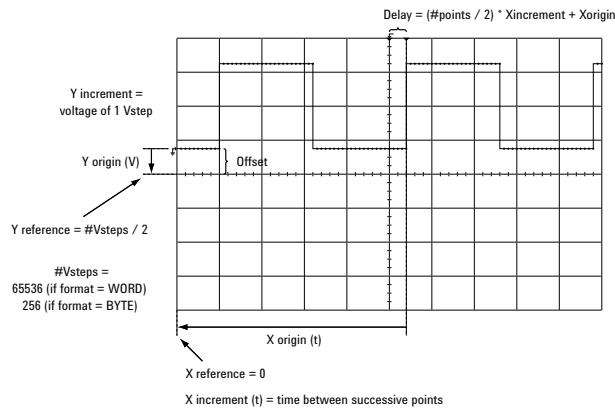
**Return Format** <preamble\_block><NL>

```
<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 461
  - ":ACQuire:COUNT" on page 160
  - ":ACQuire:POINTs" on page 162
  - ":ACQuire:TYPE" on page 169

- ":DIGitize" on page 133
- ":WAVEform:COUNT" on page 468
- ":WAVEform:DATA" on page 469
- ":WAVEform:FORMat" on page 471
- ":WAVEform:POINTs" on page 472
- ":WAVEform:TYPE" on page 486
- ":WAVEform:XINCrement" on page 489
- ":WAVEform:XORigin" on page 490
- ":WAVEform:XREFerence" on page 491
- ":WAVEform:YINCrement" on page 492
- ":WAVEform:YORigin" on page 493
- ":WAVEform:YREFerence" on page 494

### Example Code

```

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'   POINTS      : int32 - number of data points transferred.
'   COUNT       : int32 - 1 and is always 1.
'   XINCREMENT  : float64 - time difference between data points.
'   XORIGIN     : float64 - always the first data point in memory.
'   XREFERENCE  : int32 - specifies the data point associated with
'                   x-origin.
'   YINCREMENT  : float32 - voltage diff between data points.
'   YORIGIN     : float32 - value is the voltage at center screen.
'   YREFERENCE  : int32 - specifies the data point where y-origin
'                   occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)

```

```
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

**:WAVeform:SEGMENTed:COUNT**

**N** (see [page 608](#))

**Query Syntax** :WAVeform:SEGMENTed:COUNT?

**NOTE**

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

The :WAVeform:SEGMENTed:COUNT query returns the number of memory segments in the acquired data. You can use the :WAVeform:SEGMENTed:COUNT? query while segments are being acquired (although :DIGitize blocks subsequent queries until the full segmented acquisition is complete).

The segmented memory acquisition mode is enabled with the :ACQUIRE:MODE command. The number of segments to acquire is set using the :ACQUIRE:SEGMENTed:COUNT command, and data is acquired using the :DIGitize, :SINGLE, or :RUN commands.

**Return Format** <count> ::= an integer from 2 to 1000 in NR1 format (count set by :ACQUIRE:SEGMENTed:COUNT).

- See Also**
- [":ACQUIRE:MODE"](#) on page 161
  - [":ACQUIRE:SEGMENTed:COUNT"](#) on page 164
  - [":DIGitize"](#) on page 133
  - [":SINGLE"](#) on page 152
  - [":RUN"](#) on page 150
  - ["Introduction to :WAVeform Commands"](#) on page 461

**Example Code** • ["Example Code"](#) on page 165

**:WAVeform:SEGmented:TTAG****N** (see [page 608](#))**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 165
  - ["Introduction to :WAVeform Commands"](#) on page 461

**Example Code** • ["Example Code"](#) on page 165



**:WAVeform:SOURce**

**C** (see [page 608](#))

**Command Syntax** :WAVeform:SOURce <source>

```
<source> ::= {CHANnel<n> | FUNCTION | MATH | WMEMory<r>}
           for DSO models
```

```
<source> ::= {CHANnel<n> | POD{1 | 2} | BUS{1 | 2} | FUNCTION
           | MATH | WMEMory<r>}
           for MSO models
```

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

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

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, reference waveform, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply, and FFT (Fast Fourier Transform) operations.

With MSO oscilloscope models, you can choose a POD or BUS as the waveform source. There are some differences between POD and BUS when formatting and getting data from the oscilloscope:

- When POD1 or POD2 is selected as the waveform source, you can choose the BYTE, WORD, or ASCii formats (see ":WAVeform:FORMat" on [page 471](#)).

When the WORD format is chosen, every other data byte will be 0. The setting of :WAVeform:BYTeorder controls which byte is 0.

When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with unsigned decimal values separated by commas.

- When BUS1 or BUS2 is selected as the waveform source, you can choose the WORD or ASCii formats (but not BYTE because bus values are always returned as 16-bit values).

When the ASCii format is chosen, the :WAVeform:DATA? query returns a string with timestamps and hexadecimal bus values, for example:  
-5.000000000000e-08,0x1938,-4.990000000000e-08,0xff38,...

**Query Syntax** :WAVeform:SOURce?

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

**NOTE**

MATH is an alias for FUNCTION. The :WAVeform:SOURce? Query returns FUNC if the source is FUNCTION or MATH.

**Return Format** <source><NL>

```

<source> ::= {CHAN<n> | FUNC | WMEM<r>} for DSO models

<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC
              | WMEM<r>} for MSO models

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

<r> ::= {1 | 2}

```

- See Also**
- ["Introduction to :WAVEform Commands" on page 461](#)
  - [":DIGitize" on page 133](#)
  - [":WAVEform:FORMat" on page 471](#)
  - [":WAVEform:BYTeorder" on page 467](#)
  - [":WAVEform:DATA" on page 469](#)
  - [":WAVEform:PREamble" on page 476](#)

**Example Code**

```

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query. Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output. This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString ":WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
myScope.WriteString ":WAVEFORM:FORMAT BYTE"
lngVSteps = 256
intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'   FORMAT          : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'   TYPE            : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'   POINTS          : int32 - number of data points transferred.
'   COUNT           : int32 - 1 and is always 1.
'   XINCREMENT      : float64 - time difference between data points.

```

```

'   XORIGIN      : float64 - always the first data point in memory.
'   XREFERENCE   : int32 - specifies the data point associated with
'                   x-origin.
'   YINCREMENT   : float32 - voltage diff between data points.
'   YORIGIN      : float32 - value is the voltage at center screen.
'   YREFERENCE   : int32 - specifies the data point where y-origin
'                   occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'   FormatNumber(dblXIncrement * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'   FormatNumber(dblXOrigin * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'   CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'   FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'   FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'   CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
'   FormatNumber(lngVSteps * sngYIncrement / 8) + _
'   " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
'   FormatNumber((lngVSteps / 2 - lngYReference) * _
'   sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
'   FormatNumber(lngPoints * dblXIncrement / 10 * _

```

```

        1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
        FormatNumber(((lngPoints / 2 - lngXReference) * _
        dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'     <header><waveform_data><NL>
'
' Where:
'     <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 _
            + varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) _
            * sngYIncrement + sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) _
            * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

**:WAVeform:SOURce:SUBSource**

**C** (see [page 608](#))

**Command Syntax** :WAVeform:SOURce:SUBSource <subsource>

<subsource> ::= {SUB0 | RX | MOSI}

This command lets you choose from multiple data sets when an oscilloscope supports them. The InfiniiVision 2000 X-Series oscilloscopes do not support multiple data sets, so SUB0 is the only valid subsource.

**Query Syntax** :WAVeform:SOURce:SUBSource?

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

**Return Format** <subsource><NL>

<subsource> ::= SUB0

- See Also**
- "[Introduction to :WAVeform Commands](#)" on page 461
  - "[:WAVeform:SOURce](#)" on page 481

**:WAVeform:TYPE****C** (see [page 608](#))**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>

&lt;mode&gt; ::= {NORM | PEAK | AVER | HRES}

**NOTE**

If the :WAVeform:SOURce is POD1 or POD2, the type is always NORM.

- 
- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 461
  - [":ACQUIRE:TYPE"](#) on page 169
  - [":WAVeform:DATA"](#) on page 469
  - [":WAVeform:PREamble"](#) on page 476
  - [":WAVeform:SOURce"](#) on page 481

**:WAVeform:UNSigned**

**C** (see [page 608](#))

**Command Syntax** :WAVeform:UNSigned <unsigned>  
 <unsigned> ::= {{0 | OFF} | {1 | ON}}

The :WAVeform:UNSigned command turns unsigned mode on or off for the currently selected waveform. Use the WAVeform:UNSigned command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

If :WAVeform:SOURce is set to POD1, POD2, BUS1, or BUS2, WAVeform:UNSigned must be set to ON.

**Query Syntax** :WAVeform:UNSigned?

The :WAVeform:UNSigned? query returns the status of unsigned mode for the currently selected waveform.

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

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 461
  - [":WAVeform:SOURce"](#) on page 481

## :WAVeform:VIEW

**C** (see [page 608](#))

**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 461
  - "[:WAVeform:POINTS](#)" on page 472



**:WAVeform:XINCrement****C** (see [page 608](#))**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 461
  - [":WAVeform:PREamble"](#) on page 476

**Example Code** • ["Example Code"](#) on page 477

**:WAVeform:XORigin****C** (see [page 608](#))**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 461
  - "[:WAVeform:PREamble](#)" on page 476
  - "[:WAVeform:XREFerence](#)" on page 491

- Example Code**
- "[Example Code](#)" on page 477

**:WAVeform:XREFerence****C** (see [page 608](#))**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 461
  - "[:WAVeform:PREamble](#)" on page 476
  - "[:WAVeform:XORigin](#)" on page 490

**Example Code** • "[Example Code](#)" on page 477

**:WAVeform:YINCrement****C** (see [page 608](#))**Query Syntax** :WAVeform:YINCrement?

The :WAVeform:YINCrement? query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values. The y-increment for digital waveforms is always "1".

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

<value> ::= y-increment value in the current preamble in 32-bit floating point NR3 format

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 461
  - [":WAVeform:PREamble"](#) on page 476

**Example Code**

- ["Example Code"](#) on page 477

**:WAVeform:YORigin**

**C** (see [page 608](#))

**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 461
  - [":WAVeform:PREamble"](#) on page 476
  - [":WAVeform:YREFerence"](#) on page 494

- Example Code**
- ["Example Code"](#) on page 477

**:WAVeform:YREFerence**

**C** (see [page 608](#))

**Query Syntax** :WAVeform:YREFerence?

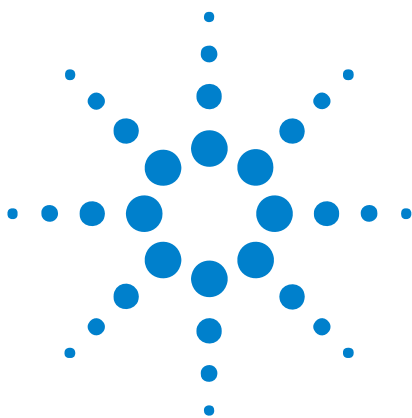
The :WAVeform:YREFerence? query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCii.

**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 461
  - [":WAVeform:PREamble"](#) on page 476
  - [":WAVeform:YORigin"](#) on page 493

**Example Code** • ["Example Code"](#) on page 477



## 27 :WGEN Commands

When the built-in waveform generator is licensed (Option WGN), you can use it to output sine, square, ramp, pulse, DC, and noise waveforms. The :WGEN commands are used to select the waveform function and parameters. See "[Introduction to :WGEN Commands](#)" on page 496.

**Table 68** :WGEN Commands Summary

Command	Query	Options and Query Returns
:WGEN:FREquency <frequency> (see <a href="#">page 497</a> )	:WGEN:FREquency? (see <a href="#">page 497</a> )	<frequency> ::= frequency in Hz in NR3 format
:WGEN:FUNction <signal> (see <a href="#">page 498</a> )	:WGEN:FUNction? (see <a href="#">page 499</a> )	<signal> ::= {SINusoid   SQUare   RAMP   PULSe   NOISe   DC}
:WGEN:FUNction:PULSe:WIDTh <width> (see <a href="#">page 500</a> )	:WGEN:FUNction:PULSe:WIDTh? (see <a href="#">page 500</a> )	<width> ::= pulse width in seconds in NR3 format
:WGEN:FUNction:RAMP:SYMMetry <percent> (see <a href="#">page 501</a> )	:WGEN:FUNction:RAMP:SYMMetry? (see <a href="#">page 501</a> )	<percent> ::= symmetry percentage from 0% to 100% in NR3 format
:WGEN:FUNction:SQUare:DCYClE <percent> (see <a href="#">page 502</a> )	:WGEN:FUNction:SQUare:DCYClE? (see <a href="#">page 502</a> )	<percent> ::= duty cycle percentage from 20% to 80% in NR3 format
:WGEN:OUTPut {{0   OFF}   {1   ON}} (see <a href="#">page 503</a> )	:WGEN:OUTPut? (see <a href="#">page 503</a> )	{0   1}
:WGEN:OUTPut:LOAD <impedance> (see <a href="#">page 504</a> )	:WGEN:OUTPut:LOAD? (see <a href="#">page 504</a> )	<impedance> ::= {ONEMeg   FIFTy}
:WGEN:PERiod <period> (see <a href="#">page 505</a> )	:WGEN:PERiod? (see <a href="#">page 505</a> )	<period> ::= period in seconds in NR3 format
:WGEN:RST (see <a href="#">page 506</a> )	n/a	n/a



**Table 68** :WGEN Commands Summary (continued)

Command	Query	Options and Query Returns
:WGEN:VOLTage <amplitude> (see page 507)	:WGEN:VOLTage? (see page 507)	<amplitude> ::= amplitude in volts in NR3 format
:WGEN:VOLTage:HIGH <high> (see page 508)	:WGEN:VOLTage:HIGH? (see page 508)	<high> ::= high-level voltage in volts, in NR3 format
:WGEN:VOLTage:LOW <low> (see page 509)	:WGEN:VOLTage:LOW? (see page 509)	<low> ::= low-level voltage in volts, in NR3 format
:WGEN:VOLTage:OFFSet <offset> (see page 510)	:WGEN:VOLTage:OFFSet? (see page 510)	<offset> ::= offset in volts in NR3 format

**Introduction to  
:WGEN  
Commands**

The :WGEN subsystem provides commands to select the waveform generator function and parameters.

**Reporting the Setup**

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

**Return Format**

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

```
:WGEN:FUNC SIN;OUTP 0;FREQ +1.0000E+03;VOLT +500.0E-03;VOLT:OFFS
+0.0E+00;:WGEN:OUTP:LOAD ONEM
```



**:WGEN:FREQuency**

**N** (see [page 608](#))

**Command Syntax** :WGEN:FREQuency <frequency>

<frequency> ::= frequency in Hz in NR3 format

For Sine, Square, Ramp, and Pulse waveforms, the :WGEN:FREQuency command specifies the frequency of the waveform.

You can also specify the frequency indirectly using the :WGEN:PERiod command.

**Query Syntax** :WGEN:FREQuency?

The :WGEN:FREQuency? query returns the currently set waveform generator frequency.

**Return Format** <frequency><NL>

<frequency> ::= frequency in Hz in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTion"](#) on page 498
  - [":WGEN:PERiod"](#) on page 505

**:WGEN:FUNction**

**N** (see page 608)

**Command Syntax** :WGEN:FUNction <signal>

<signal> ::= {SINusoid | SQUare | RAMP | PULSe | NOISE | DC}

The :WGEN:FUNction command selects the type of waveform:

Waveform Type	Characteristics
SINusoid	<p>Use these commands to set the sine signal parameters:</p> <ul style="list-style-type: none"> <li>• ":WGEN:FREQuency" on page 497</li> <li>• ":WGEN:PERiod" on page 505</li> <li>• ":WGEN:VOLTage" on page 507</li> <li>• ":WGEN:VOLTage:OFFSet" on page 510</li> <li>• ":WGEN:VOLTage:HIGh" on page 508</li> <li>• ":WGEN:VOLTage:LOW" on page 509</li> </ul> <p>The frequency can be adjusted from 100 mHz to 20 MHz.</p>
SQUare	<p>Use these commands to set the square wave signal parameters:</p> <ul style="list-style-type: none"> <li>• ":WGEN:FREQuency" on page 497</li> <li>• ":WGEN:PERiod" on page 505</li> <li>• ":WGEN:VOLTage" on page 507</li> <li>• ":WGEN:VOLTage:OFFSet" on page 510</li> <li>• ":WGEN:VOLTage:HIGh" on page 508</li> <li>• ":WGEN:VOLTage:LOW" on page 509</li> <li>• ":WGEN:FUNction:SQUare:DCYCLE" on page 502</li> </ul> <p>The frequency can be adjusted from 100 mHz to 10 MHz. The duty cycle can be adjusted from 20% to 80%.</p>
RAMP	<p>Use these commands to set the ramp signal parameters:</p> <ul style="list-style-type: none"> <li>• ":WGEN:FREQuency" on page 497</li> <li>• ":WGEN:PERiod" on page 505</li> <li>• ":WGEN:VOLTage" on page 507</li> <li>• ":WGEN:VOLTage:OFFSet" on page 510</li> <li>• ":WGEN:VOLTage:HIGh" on page 508</li> <li>• ":WGEN:VOLTage:LOW" on page 509</li> <li>• ":WGEN:FUNction:RAMP:SYMMetry" on page 501</li> </ul> <p>The frequency can be adjusted from 100 mHz to 100 kHz. Symmetry represents the amount of time per cycle that the ramp waveform is rising and can be adjusted from 0% to 100%.</p>

Waveform Type	Characteristics
PULSe	<p>Use these commands to set the pulse signal parameters:</p> <ul style="list-style-type: none"> <li>• <a href="#">":WGEN:FREQuency"</a> on page 497</li> <li>• <a href="#">":WGEN:PERiod"</a> on page 505</li> <li>• <a href="#">":WGEN:VOLTage"</a> on page 507</li> <li>• <a href="#">":WGEN:VOLTage:OFFSet"</a> on page 510</li> <li>• <a href="#">":WGEN:VOLTage:HIGh"</a> on page 508</li> <li>• <a href="#">":WGEN:VOLTage:LOW"</a> on page 509</li> <li>• <a href="#">":WGEN:FUNcTion:PULSe:WIDTh"</a> on page 500</li> </ul> <p>The frequency can be adjusted from 100 mHz to 10 MHz. The pulse width can be adjusted from 20 ns to the period minus 20 ns.</p>
DC	<p>Use this command to set the DC level:</p> <ul style="list-style-type: none"> <li>• <a href="#">":WGEN:VOLTage:OFFSet"</a> on page 510</li> </ul>
NOISe	<p>Use these commands to set the noise signal parameters:</p> <ul style="list-style-type: none"> <li>• <a href="#">":WGEN:VOLTage"</a> on page 507</li> <li>• <a href="#">":WGEN:VOLTage:OFFSet"</a> on page 510</li> <li>• <a href="#">":WGEN:VOLTage:HIGh"</a> on page 508</li> <li>• <a href="#">":WGEN:VOLTage:LOW"</a> on page 509</li> </ul>

For all waveform types, the output amplitude, into 50  $\Omega$ , can be adjusted from 10 mVpp to 2.5 Vpp (or from 20 mVpp to 5 Vpp into and open-circuit load).

**Query Syntax** :WGEN:FUNcTion?

The :WGEN:FUNcTion? query returns the currently selected signal type.

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

<signal> ::= {SIN | SQU | RAMP | PULS | NOIS | DC}

**See Also** • ["Introduction to :WGEN Commands"](#) on page 496

## :WGEN:FUNCTION:PULSE:WIDTH

**N** (see [page 608](#))

**Command Syntax** :WGEN:FUNCTION:PULSE:WIDTH <width>

<width> ::= pulse width in seconds in NR3 format

For Pulse waveforms, the :WGEN:FUNCTION:PULSE:WIDTH command specifies the width of the pulse.

The pulse width can be adjusted from 20 ns to the period minus 20 ns.

**Query Syntax** :WGEN:FUNCTION:PULSE:WIDTH?

The :WGEN:FUNCTION:PULSE:WIDTH? query returns the currently set pulse width.

**Return Format** <width><NL>

<width> ::= pulse width in seconds in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498

**:WGEN:FUNCTION:RAMP:SYMMetry**

**N** (see [page 608](#))

**Command Syntax** :WGEN:FUNCTION:RAMP:SYMMetry <percent>

<percent> ::= symmetry percentage from 0% to 100% in NR3 format

For Ramp waveforms, the :WGEN:FUNCTION:RAMP:SYMMetry command specifies the symmetry of the waveform.

Symmetry represents the amount of time per cycle that the ramp waveform is rising.

**Query Syntax** :WGEN:FUNCTION:RAMP:SYMMetry?

The :WGEN:FUNCTION:RAMP:SYMMetry? query returns the currently set ramp symmetry.

**Return Format** <percent><NL>

<percent> ::= symmetry percentage from 0% to 100% in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498

## :WGEN:FUNCTION:SQUare:DCYCLE

**N** (see [page 608](#))

**Command Syntax** :WGEN:FUNCTION:SQUare:DCYCLE <percent>

<percent> ::= duty cycle percentage from 20% to 80% in NR3 format

For Square waveforms, the :WGEN:FUNCTION:SQUare:DCYCLE command specifies the square wave duty cycle.

Duty cycle is the percentage of the period that the waveform is high.

**Query Syntax** :WGEN:FUNCTION:SQUare:DCYCLE?

The :WGEN:FUNCTION:SQUare:DCYCLE? query returns the currently set square wave duty cycle.

**Return Format** <percent><NL>

<percent> ::= duty cycle percentage from 20% to 80% in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498

**:WGEN:OUTPut**

**N** (see [page 608](#))

**Command Syntax** :WGEN:OUTPut <on\_off>

<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :WGEN:OUTPut command specifies whether the waveform generator signal output is ON (1) or OFF (0).

**Query Syntax** :WGEN:OUTPut?

The :WGEN:OUTPut? query returns the current state of the waveform generator output setting.

**Return Format** <on\_off><NL>

<on\_off> ::= {1 | 0}

**See Also** • ["Introduction to :WGEN Commands"](#) on page 496

**:WGEN:OUTPut:LOAD**

**N** (see [page 608](#))

**Command Syntax** :WGEN:OUTPut:LOAD <impedance>

<impedance> ::= {ONEMeg | FIFTy}

The :WGEN:OUTPut:LOAD command selects the expected output load impedance.

The output impedance of the Gen Out BNC is fixed at 50 ohms. However, the output load selection lets the waveform generator display the correct amplitude and offset levels for the expected output load.

If the actual load impedance is different than the selected value, the displayed amplitude and offset levels will be incorrect.

**Query Syntax** :WGEN:OUTPut:LOAD?

The :WGEN:OUTPut:LOAD? query returns the current expected output load impedance.

**Return Format** <impedance><NL>

<impedance> ::= {ONEM | FIFT}

**See Also** • ["Introduction to :WGEN Commands"](#) on page 496



**:WGEN:PERiod**

**N** (see [page 608](#))

**Command Syntax** :WGEN:PERiod <period>

<period> ::= period in seconds in NR3 format

For Sine, Square, Ramp, and Pulse waveforms, the :WGEN:PERiod command specifies the period of the waveform.

You can also specify the period indirectly using the :WGEN:FREQuency command.

**Query Syntax** :WGEN:PERiod?

The :WGEN:PERiod? query returns the currently set waveform generator period.

**Return Format** <period><NL>

<period> ::= period in seconds in NR3 format

- See Also**
- "[Introduction to :WGEN Commands](#)" on page 496
  - "[:WGEN:FUNCTION](#)" on page 498
  - "[:WGEN:FREQuency](#)" on page 497

## :WGEN:RST

**N** (see [page 608](#))

**Command Syntax** :WGEN:RST

The :WGEN:RST command restores the waveform generator factory default settings (1 kHz sine wave, 500 mVpp, 0 V offset).

- See Also**
- "[Introduction to :WGEN Commands](#)" on page 496
  - "[:WGEN:FUNCTION](#)" on page 498
  - "[:WGEN:FREQUENCY](#)" on page 497

**:WGEN:VOLTage**

**N** (see [page 608](#))

**Command Syntax** :WGEN:VOLTage <amplitude>

<amplitude> ::= amplitude in volts in NR3 format

For Sine, Square, Ramp, Pulse, and Noise waveforms, the :WGEN:VOLTage command specifies the waveform's amplitude. Use the :WGEN:VOLTage:OFFSet command to specify the offset voltage.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

**Query Syntax** :WGEN:VOLTage?

The :WGEN:VOLTage? query returns the currently specified waveform amplitude.

**Return Format** <amplitude><NL>

<amplitude> ::= amplitude in volts in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498
  - [":WGEN:VOLTage:OFFSet"](#) on page 510
  - [":WGEN:VOLTage:HIGH"](#) on page 508
  - [":WGEN:VOLTage:LOW"](#) on page 509

**:WGEN:VOLTage:HIGH**

**N** (see [page 608](#))

**Command Syntax** :WGEN:VOLTage:HIGH <high>

<high> ::= high-level voltage in volts, in NR3 format

For Sine, Square, Ramp, Pulse, and Noise waveforms, the :WGEN:VOLTage:HIGH command specifies the waveform's high-level voltage. Use the :WGEN:VOLTage:LOW command to specify the low-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

**Query Syntax** :WGEN:VOLTage:HIGH?

The :WGEN:VOLTage:HIGH? query returns the currently specified waveform high-level voltage.

**Return Format** <high><NL>

<high> ::= high-level voltage in volts, in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498
  - [":WGEN:VOLTage:LOW"](#) on page 509
  - [":WGEN:VOLTage"](#) on page 507
  - [":WGEN:VOLTage:OFFSet"](#) on page 510

**:WGEN:VOLTage:LOW**

**N** (see [page 608](#))

**Command Syntax** :WGEN:VOLTage:LOW <low>

<low> ::= low-level voltage in volts, in NR3 format

For Sine, Square, Ramp, Pulse, and Noise waveforms, the :WGEN:VOLTage:LOW command specifies the waveform's low-level voltage. Use the :WGEN:VOLTage:HIGH command to specify the high-level voltage.

You can also specify the high-level and low-level voltages indirectly using the :WGEN:VOLTage and :WGEN:VOLTage:OFFSet commands. For example, a high-level voltage of 4 V and a low-level voltage of -1 V is the same as an amplitude of 5 V and an offset of 1 V.

**Query Syntax** :WGEN:VOLTage:LOW?

The :WGEN:VOLTage:LOW? query returns the currently specified waveform low-level voltage.

**Return Format** <low><NL>

<low> ::= low-level voltage in volts, in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498
  - [":WGEN:VOLTage:LOW"](#) on page 509
  - [":WGEN:VOLTage"](#) on page 507
  - [":WGEN:VOLTage:OFFSet"](#) on page 510

**:WGEN:VOLTage:OFFSet**

**N** (see [page 608](#))

**Command Syntax** :WGEN:VOLTage:OFFSet <offset>

<offset> ::= offset in volts in NR3 format

For Sine, Square, Ramp, Pulse, and Noise waveforms, the :WGEN:VOLTage:OFFSet command specifies the waveform's offset voltage. Use the :WGEN:VOLTage command to specify the amplitude.

You can also specify the amplitude and offset indirectly using the :WGEN:VOLTage:HIGH and :WGEN:VOLTage:LOW commands. For example, an amplitude of 5 V and an offset of 1 V is the same as a high-level voltage of 4 V and a low-level voltage of -1 V.

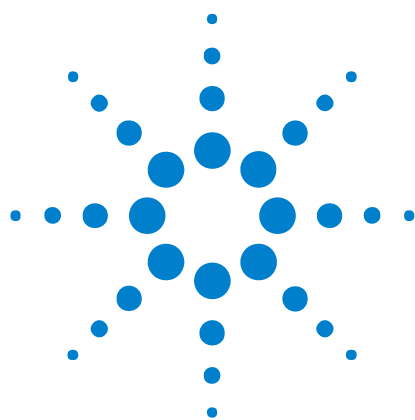
**Query Syntax** :WGEN:VOLTage:OFFSet?

The :WGEN:VOLTage:OFFSet? query returns the currently specified waveform offset voltage.

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

<offset> ::= offset in volts in NR3 format

- See Also**
- ["Introduction to :WGEN Commands"](#) on page 496
  - [":WGEN:FUNCTION"](#) on page 498
  - [":WGEN:VOLTage"](#) on page 507
  - [":WGEN:VOLTage:HIGH"](#) on page 508
  - [":WGEN:VOLTage:LOW"](#) on page 509



## 28 :WMEemory<r> Commands

Control reference waveforms.

**Table 69** :WMEemory<r> Commands Summary

Command	Query	Options and Query Returns
:WMEemory<r>:CLEar (see <a href="#">page 513</a> )	n/a	<r> ::= 1-2 in NR1 format
:WMEemory<r>:DISPlay { {0   OFF}   {1   ON} } (see <a href="#">page 514</a> )	:WMEemory<r>:DISPlay? (see <a href="#">page 514</a> )	<r> ::= 1-2 in NR1 format {0   1}
:WMEemory<r>:LABel <string> (see <a href="#">page 515</a> )	:WMEemory<r>:LABel? (see <a href="#">page 515</a> )	<r> ::= 1-2 in NR1 format <string> ::= any series of 10 or less ASCII characters enclosed in quotation marks
:WMEemory<r>:SAVE <source> (see <a href="#">page 516</a> )	n/a	<r> ::= 1-2 in NR1 format <source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1 to (# analog channels) in NR1 format NOTE: Only ADD or SUBtract math operations can be saved as reference waveforms.
:WMEemory<r>:SKEW <skew> (see <a href="#">page 517</a> )	:WMEemory<r>:SKEW? (see <a href="#">page 517</a> )	<r> ::= 1-2 in NR1 format <skew> ::= time in seconds in NR3 format
:WMEemory<r>:YOFFset <offset>[suffix] (see <a href="#">page 518</a> )	:WMEemory<r>:YOFFset? (see <a href="#">page 518</a> )	<r> ::= 1-2 in NR1 format <offset> ::= vertical offset value in NR3 format [suffix] ::= {V   mV}



**Table 69** :WMEemory<r> Commands Summary (continued)

Command	Query	Options and Query Returns
:WMEemory<r>:YRANge <range>[suffix] (see page 519)	:WMEemory<r>:YRANge? (see page 519)	<r> ::= 1-2 in NR1 format <range> ::= vertical full-scale range value in NR3 format [suffix] ::= {V   mV}
:WMEemory<r>:YSCale <scale>[suffix] (see page 520)	:WMEemory<r>:YSCale? (see page 520)	<r> ::= 1-2 in NR1 format <scale> ::= vertical units per division value in NR3 format [suffix] ::= {V   mV}



**:WMEemory<r>:CLEar**

**N** (see page 608)

**Command Syntax** :WMEemory<r>:CLEar

<r> ::= 1-2 in NR1 format

The :WMEemory<r>:CLEar command clears the specified reference waveform location.

- See Also**
- [Chapter 28, “:WMEemory<r> Commands,”](#) starting on page 511
  - [":WMEemory<r>:SAVE"](#) on page 516
  - [":WMEemory<r>:DISPlay"](#) on page 514

## :WMEemory<r>:DISPlay

**N** (see page 608)

**Command Syntax** :WMEemory<r>:DISPlay <on\_off>

<r> ::= 1-2 in NR1 format

<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :WMEemory<r>:DISPlay command turns the display of the specified reference waveform on or off.

There are two reference waveform locations, but only one reference waveform can be displayed at a time. That means, if :WMEemory1:DISPlay is ON, sending the :WMEemory2:DISPlay ON command will automatically set :WMEemory1:DISPlay OFF.

**Query Syntax** :WMEemory<r>:DISPlay?

The :WMEemory<r>:DISPlay? query returns the current display setting for the reference waveform.

**Return Format** <on\_off><NL>

<on\_off> ::= {1 | 0}

- See Also**
- [Chapter 28, “:WMEemory<r> Commands,”](#) starting on page 511
  - [":WMEemory<r>:CLEar"](#) on page 513
  - [":WMEemory<r>:LABel"](#) on page 515

**:WMEemory<r>:LABel**

**N** (see [page 608](#))

**Command Syntax** :WMEemory<r>:LABel <string>  
 <r> ::= 1-2 in NR1 format  
 <string> ::= quoted ASCII string

**NOTE**

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

The :WMEemory<r>:LABel command sets the reference waveform label to the string that follows.

Setting a label for a reference waveform also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

**Query Syntax** :WMEemory<r>:LABel?

The :WMEemory<r>:LABel? query returns the label associated with a particular reference waveform.

**Return Format** <string><NL>  
 <string> ::= quoted ASCII string

**See Also**

- [Chapter 28](#), “:WMEemory<r> Commands,” starting on page 511
- [":WMEemory<r>:DISPlay"](#) on page 514

## :WMEemory<r>:SAVE

**N** (see [page 608](#))

**Command Syntax** :WMEemory<r>:SAVE <source>  
<r> ::= 1-2 in NR1 format  
<source> ::= {CHANnel<n> | FUNCTION | MATH}  
<n> ::= 1 to (# analog channels) in NR1 format

The :WMEemory<r>:SAVE command copies the analog channel or math function waveform to the specified reference waveform location.

### NOTE

Only ADD or SUBtract math operations can be saved as reference waveforms.

- See Also**
- [Chapter 28, “:WMEemory<r> Commands,”](#) starting on page 511
  - [“:WMEemory<r>:DISPlay”](#) on page 514

**:WMEemory<r>:SKEW**

**N** (see [page 608](#))

**Command Syntax** :WMEemory<r>:SKEW <skew>

<r> ::= 1-2 in NR1 format

<skew> ::= time in seconds in NR3 format

The :WMEemory<r>:SKEW command sets the skew factor for the specified reference waveform.

**Query Syntax** :WMEemory<r>:SKEW?

The :WMEemory<r>:SKEW? query returns the current skew setting for the selected reference waveform.

**Return Format** <skew><NL>

<skew> ::= time in seconds in NR3 format

- See Also**
- [Chapter 28, “:WMEemory<r> Commands,”](#) starting on page 511
  - [":WMEemory<r>:DISPlay"](#) on page 514
  - [":WMEemory<r>:YOFFset"](#) on page 518
  - [":WMEemory<r>:YRANge"](#) on page 519
  - [":WMEemory<r>:YSCale"](#) on page 520

**:WMEemory<r>:YOFFset**

**N** (see page 608)

**Command Syntax** :WMEemory<r>:YOFFset <offset> [<suffix>]

<r> ::= 1-2 in NR1 format

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

<suffix> ::= {V | mV}

The :WMEemory<r>:YOFFset command sets the value that is represented at center screen for the selected reference waveform.

The range of legal values varies with the value set by the :WMEemory<r>:YRANge or :WMEemory<r>:YSCale commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are affected by the probe attenuation setting.

**Query Syntax** :WMEemory<r>:YOFFset?

The :WMEemory<r>:YOFFset? query returns the current offset value for the selected reference waveform.

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

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

- See Also**
- [Chapter 28, “:WMEemory<r> Commands,”](#) starting on page 511
  - [":WMEemory<r>:DISPlay"](#) on page 514
  - [":WMEemory<r>:YRANge"](#) on page 519
  - [":WMEemory<r>:YSCale"](#) on page 520
  - [":WMEemory<r>:SKEW"](#) on page 517

**:WMEemory<r>:YRANge**

**N** (see [page 608](#))

**Command Syntax** :WMEemory<r>:YRANge <range>[<suffix>]

<r> ::= 1-2 in NR1 format

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

The :WMEemory<r>:YRANge command defines the full-scale vertical axis of the selected reference waveform.

Legal values for the range are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

**Query Syntax** :WMEemory<r>:YRANge?

The :WMEemory<r>:YRANge? query returns the current full-scale range setting for the specified reference waveform.

**Return Format** <range><NL>

<range> ::= vertical full-scale range value in NR3 format

- See Also**
- [Chapter 28](#), “:WMEemory<r> Commands,” starting on page 511
  - “:WMEemory<r>:DISPlay” on page 514
  - “:WMEemory<r>:YOFFset” on page 518
  - “:WMEemory<r>:SKEW” on page 517
  - “:WMEemory<r>:YSCale” on page 520

**:WMEemory<r>:YScale**

**N** (see [page 608](#))

**Command Syntax** :WMEemory<r>:YScale <scale>[<suffix>]

<r> ::= 1-2 in NR1 format

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

<suffix> ::= {V | mV}

The :WMEemory<r>:YScale command sets the vertical scale, or units per division, of the selected reference waveform.

Legal values for the scale are copied from the original source waveform (that is, the analog channel or math function waveform that was originally saved as a reference waveform).

**Query Syntax** :WMEemory<r>:YScale?

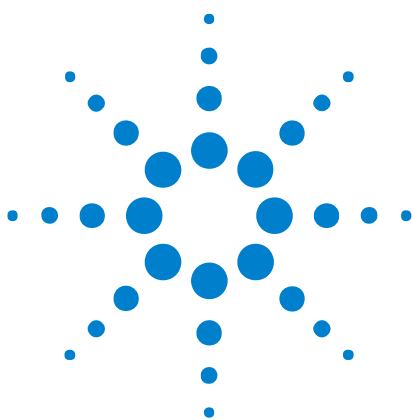
The :WMEemory<r>:YScale? query returns the current scale setting for the specified reference waveform.

**Return Format** <scale><NL>

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

- See Also**
- [Chapter 28](#), “:WMEemory<r> Commands,” starting on page 511
  - “:WMEemory<r>:DISPlay” on page 514
  - “:WMEemory<r>:YOFFset” on page 518
  - “:WMEemory<r>:YRANge” on page 519
  - “:WMEemory<r>:SKEW” on page 517





## 29 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 608).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see <a href="#">page 194</a> )	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see <a href="#">page 195</a> )	
ANALog<n>:INVert	:CHANnel<n>:INVert (see <a href="#">page 198</a> )	
ANALog<n>:LABel	:CHANnel<n>:LABel (see <a href="#">page 199</a> )	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see <a href="#">page 200</a> )	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see <a href="#">page 201</a> )	
ANALog<n>:PMODE	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see <a href="#">page 207</a> )	
:CHANnel:ACTivity (see <a href="#">page 526</a> )	:ACTivity (see <a href="#">page 125</a> )	
:CHANnel:LABel (see <a href="#">page 527</a> )	:CHANnel<n>:LABel (see <a href="#">page 199</a> ) or :DIGital<d>:LABel (see <a href="#">page 220</a> )	use CHANnel<n>:LABel for analog channels and use DIGital<n>:LABel for digital channels
:CHANnel:THReshold (see <a href="#">page 528</a> )	:POD<n>:THReshold (see <a href="#">page 368</a> ) or :DIGital<d>:THReshold (see <a href="#">page 223</a> )	
:CHANnel2:SKEW (see <a href="#">page 529</a> )	:CHANnel<n>:PROBe:SKEW (see <a href="#">page 204</a> )	



## 29 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel<n>:INPut (see <a href="#">page 530</a> )	:CHANnel<n>:IMPedance (see <a href="#">page 197</a> )	
:CHANnel<n>:PMODE (see <a href="#">page 531</a> )	none	
:DISPlay:CONNect (see <a href="#">page 532</a> )	:DISPlay:VECTors (see <a href="#">page 232</a> )	
:DISPlay:ORDer (see <a href="#">page 533</a> )	none	
:ERASe (see <a href="#">page 534</a> )	:DISPlay:CLEar (see <a href="#">page 227</a> )	
:EXTernal:PMODE (see <a href="#">page 535</a> )	none	
FUNcTion1, FUNcTion2	:FUNcTion Commands (see <a href="#">page 239</a> )	ADD not included
:FUNcTion:SOURce (see <a href="#">page 536</a> )	:FUNcTion:SOURce1 (see <a href="#">page 252</a> )	Obsolete command has ADD, SUBTract, and MULTIpLy parameters; current command has GOFT parameter.
:FUNcTion:VIEW (see <a href="#">page 537</a> )	:FUNcTion:DISPlay (see <a href="#">page 243</a> )	
:HARDcopy:DESTination (see <a href="#">page 538</a> )	:HARDcopy:FILEname (see <a href="#">page 539</a> )	
:HARDcopy:FILEname (see <a href="#">page 539</a> )	:RECall:FILEname (see <a href="#">page 373</a> ) :SAVE:FILEname (see <a href="#">page 373</a> )	
:HARDcopy:GRAYscale (see <a href="#">page 540</a> )	:HARDcopy:PALette (see <a href="#">page 271</a> )	
:HARDcopy:IGColors (see <a href="#">page 541</a> )	:HARDcopy:INKSaver (see <a href="#">page 263</a> )	
:HARDcopy:PDRiver (see <a href="#">page 542</a> )	:HARDcopy:APRinter (see <a href="#">page 260</a> )	
:MEASure:LOWer (see <a href="#">page 543</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 298</a> )	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see <a href="#">page 544</a> )	:MEASure:CLEar (see <a href="#">page 297</a> )	
:MEASure:TDELta (see <a href="#">page 545</a> )	:MARKer:XDELta (see <a href="#">page 282</a> )	
:MEASure:THResholds (see <a href="#">page 546</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 298</a> )	MEASure:DEFine:THResholds can define absolute values or percentage

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:TStArt (see <a href="#">page 547</a> )	:MARKer:X1Position (see <a href="#">page 278</a> )	
:MEASure:TStOp (see <a href="#">page 548</a> )	:MARKer:X2Position (see <a href="#">page 280</a> )	
:MEASure:TVOLt (see <a href="#">page 549</a> )	:MEASure:TVALue (see <a href="#">page 319</a> )	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see <a href="#">page 551</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 298</a> )	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see <a href="#">page 552</a> )	:MARKer:YDELta (see <a href="#">page 285</a> )	
:MEASure:VStArt (see <a href="#">page 553</a> )	:MARKer:Y1Position (see <a href="#">page 283</a> )	
:MEASure:VStOp (see <a href="#">page 554</a> )	:MARKer:Y2Position (see <a href="#">page 284</a> )	
:MTESt:AMASk:{SAVE   StORe} (see <a href="#">page 555</a> )	:SAVE:MASk[:StArt] (see <a href="#">page 388</a> )	
:MTESt:AVERage (see <a href="#">page 556</a> )	:ACQuire:TYPE AVERage (see <a href="#">page 169</a> )	
:MTESt:AVERage:COUNt (see <a href="#">page 557</a> )	:ACQuire:COUNt (see <a href="#">page 160</a> )	
:MTESt:LOAD (see <a href="#">page 558</a> )	:RECall:MASk[:StArt] (see <a href="#">page 374</a> )	
:MTESt:RUMode (see <a href="#">page 559</a> )	:MTESt:RMODE (see <a href="#">page 350</a> )	
:MTESt:RUMode:SOFailure (see <a href="#">page 560</a> )	:MTESt:RMODE:FACTION:StOP (see <a href="#">page 354</a> )	
:MTESt:{StArt   StOP} (see <a href="#">page 561</a> )	:RUN (see <a href="#">page 150</a> ) or :StOP (see <a href="#">page 154</a> )	
:MTESt:TRIGger:SOURce (see <a href="#">page 562</a> )	:TRIGger Commands (see <a href="#">page 423</a> )	There are various commands for setting the source with different types of triggers.
:PRINt? (see <a href="#">page 563</a> )	:DISPlay:DATA? (see <a href="#">page 228</a> )	
:SAVE:IMAGe:AREA (see <a href="#">page 565</a> )	none	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TIMebase:DElay (see <a href="#">page 566</a> )	:TIMebase:POSition (see <a href="#">page 414</a> ) or :TIMebase:WINDow:POSition (see <a href="#">page 419</a> )	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of zoomed (delayed) time base window.
:TRIGger:THReshold (see <a href="#">page 567</a> )	:POD<n>:THReshold (see <a href="#">page 368</a> ) or :DIGital<d>:THReshold (see <a href="#">page 223</a> )	
:TRIGger:TV:TVMode (see <a href="#">page 568</a> )	:TRIGger:TV:MODE (see <a href="#">page 455</a> )	

**Discontinued Commands**

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 2000 X-Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERsistence INFinite (see <a href="#">page 231</a> )	
CHANnel:MATH	:FUNCTion:OPERation (see <a href="#">page 248</a> )	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTectioN (see <a href="#">page 206</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:FREeze	none	
DISPlay:GRID	none	
DISPlay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCTion:MOVE	none	

Discontinued Command	Current Command Equivalent	Comments
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 118</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 455</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 2000 X-Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

## :CHANnel:ACTivity

**O** (see [page 608](#))

**Command Syntax** :CHANnel:ACTivity

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

### NOTE

The :CHANnel:ACTivity command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :ACTivity command (see [page 125](#)) instead.

---

**Query Syntax** :CHANnel:ACTivity?

The :CHANnel:ACTivity? query returns the active edges since the last clear, and returns the current logic levels.

**Return Format** <edges>, <levels><NL>

<edges> ::= presence of edges (32-bit integer in NR1 format).

<levels> ::= logical highs or lows (32-bit integer in NR1 format).

### NOTE

A bit equal to zero indicates that no edges were detected at the specified threshold since the last clear on that channel. Edges may have occurred that were not detected because of the threshold setting.

---

A bit equal to one indicates that edges have been detected at the specified threshold since the last clear on that channel.

## :CHANnel:LABel

**O** (see [page 608](#))

**Command Syntax** :CHANnel:LABel <source\_text><string>  
 <source\_text> ::= {CHANnel1 | CHANnel2 | DIGital<d>}  
 <d> ::= 0 to (# digital channels - 1) in NR1 format  
 <string> ::= quoted ASCII string

The :CHANnel:LABel command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

**NOTE**

The :CHANnel:LABel command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABel command (see [page 199](#)) or :DIGital<n>:LABel command (see [page 220](#)).

**Query Syntax** :CHANnel:LABel?

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

**Return Format** <string><NL>  
 <string> ::= quoted ASCII string

**:CHANnel:THReshold**

**O** (see [page 608](#))

**Command Syntax** :CHANnel:THReshold <channel group>, <threshold type> [, <value>]  
 <channel group> ::= {POD1 | POD2}  
 <threshold type> ::= {CMOS | ECL | TTL | USERdef}  
 <value> ::= voltage for USERdef in NR3 format [volt\_type]  
 [volt\_type] ::= {V | mV (-3) | uV (-6)}

The :CHANnel:THReshold command sets the threshold for a group of channels. The threshold is either set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is ignored.

**NOTE**

The :CHANnel:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold command (see [page 368](#)) or :DIGital<n>:THReshold command (see [page 223](#)).

**Query Syntax** :CHANnel:THReshold? <channel group>

The :CHANnel:THReshold? query returns the voltage and threshold text for a specific group of channels.

**Return Format** <threshold type> [, <value>]<NL>  
 <threshold type> ::= {CMOS | ECL | TTL | USERdef}  
 <value> ::= voltage for USERdef (float 32 NR3)

**NOTE**

- CMOS = 2.5V
- TTL = 1.5V
- ECL = -1.3V
- USERdef ::= -6.0V to 6.0V



## :CHANnel2:SKEW

**O** (see [page 608](#))

**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 204](#)) 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 192

**:CHANnel<n>:INPut**

**O** (see [page 608](#))

**Command Syntax** :CHANnel<n>:INPut <impedance>  
 <impedance> ::= {ONEMeg | FIFTy}  
 <n> ::= 1 to (# analog channels) in NR1 format

The :CHANnel<n>:INPut command selects the input impedance setting for the specified channel. The legal values for this command are ONEMeg (1 M $\Omega$ ) and FIFTy (50 $\Omega$ ).

**NOTE**

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see [page 197](#)) 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

**O** (see [page 608](#))

**Command Syntax** :CHANnel<n>:PMODE <pmode value>  
 <pmode value> ::= {AUTO | MANual}  
 <n> ::= 1 to (# analog channels) in NR1 format

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODE sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

**NOTE**

The :CHANnel<n>:PMODE command is an obsolete command provided for compatibility to previous oscilloscopes.

**Query Syntax** :CHANnel<n>:PMODE?

The :CHANnel<n>:PMODE? query returns AUT if an autosense probe is attached and MAN otherwise.

**Return Format** <pmode value><NL>  
 <pmode value> ::= {AUT | MAN}

**:DISPlay:CONNect**

**O** (see [page 608](#))

**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 232](#)) 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 232](#)

## :DISPlay:ORDer

**O** (see [page 608](#))

**Query Syntax** :DISPlay:ORDer?

The :DISPlay:ORDer? query returns a list of digital channel numbers in screen order, from top to bottom, separated by commas. Busing is displayed as digital channels with no separator. For example, in the following list, the bus consists of digital channels 4 and 5: DIG1, DIG4 DIG5, DIG7.

**NOTE**

The :DISPlay:ORDer command is an obsolete command provided for compatibility to previous oscilloscopes. This command is only available on the MSO models.

**Return Format** <order><NL>  
 <order> ::= Unquoted ASCII string

**NOTE**

A return value is included for each digital channel. A return value of NONE indicates that a channel is turned off.

**See Also** • [":DIGital<d>:POSition"](#) on page 221

**Example Code**

```
' DISP_ORDER - Set the order the channels are displayed on the
' analyzer. You can enter between 1 and 32 channels at one time.
' If you leave out channels, they will not be displayed.

' Display ONLY channel 0 and channel 10 in that order.
myScope.WriteString ":DISPLAY:ORDER 0,10"
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

## **:ERASe**

**O** (see [page 608](#))

**Command Syntax** :ERASe

The :ERASe command erases the screen.

### **NOTE**

The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISplay:CLEar command (see [page 227](#)) instead.

---

**:EXternal:PMODE**

**O** (see [page 608](#))

**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**

**O** (see [page 608](#))

**Command Syntax** :FUNCTION:SOURce <value>  
 <value> ::= {CHANnel<n> | ADD | SUBtract | MULTiply}  
 <n> ::= 1 to (# analog channels) in NR1 format

The :FUNCTION:SOURce command is only used when an FFT (Fast Fourier Transform) 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 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 252](#)) instead.

**Query Syntax** :FUNCTION:SOURce?

The :FUNCTION:SOURce? query returns the current source for function operations.

**Return Format** <value><NL>  
 <value> ::= {CHAN<n> | ADD | SUBT | MULT}  
 <n> ::= 1 to (# analog channels) in NR1 format

**See Also**

- "[Introduction to :FUNCTION Commands](#)" on page 240
- "[:FUNCTION:OPERation](#)" on page 248



## :FUNCTION:VIEW

**O** (see [page 608](#))

**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 243](#)) 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

**O** (see [page 608](#))

**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 539](#)) 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 258

**:HARDcopy:FILENAME**

**O** (see [page 608](#))

**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 382](#)) and :RECall:FILENAME command (see [page 373](#)) 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 258

## :HARDcopy:GRAYscale

**O** (see [page 608](#))

**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 271](#)) 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 258

## :HARDcopy:IGColors

**O** (see [page 608](#))

**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 263](#)) 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 258

**:HARDcopy:PDRiver**

**O** (see [page 608](#))

**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 260](#)) 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 258

## :MEASure:LOWer

**O** (see [page 608](#))

**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 298](#)) 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 294
  - [":MEASure:THResholds"](#) on page 546
  - [":MEASure:UPPer"](#) on page 551

## :MEASure:SCRatch

**O** (see [page 608](#))

**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 297](#)) instead.

---



## :MEASure:TDELta

**O** (see [page 608](#))

**Query Syntax** :MEASure:TDELta?

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

$Tdelta = Tstop - Tstart$

Tstart is the time at the start marker (X1 cursor) and Tstop is the time at the stop marker (X2 cursor). No measurement is made when the :MEASure:TDELta? query is received by the oscilloscope. The delta time value that is output is the current value. This is the same value as the front-panel cursors delta X value.

### NOTE

The :MEASure:TDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:XDELta command (see [page 282](#)) instead.

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

<value> ::= time difference between start and stop markers in NR3 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 276
  - "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MARKer:X1Position](#)" on page 278
  - "[:MARKer:X2Position](#)" on page 280
  - "[:MARKer:XDELta](#)" on page 282
  - "[:MEASure:TSTArt](#)" on page 547
  - "[:MEASure:TSTOp](#)" on page 548

## :MEASure:THResholds

**O** (see [page 608](#))

**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 298](#)) 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 294](#)
  - [":MEASure:LOWer"](#) on [page 543](#)
  - [":MEASure:UPPer"](#) on [page 551](#)

## :MEASure:TSTArt

**O** (see [page 608](#))

**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 610](#)). 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 278](#)) 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 276
  - ["Introduction to :MEASure Commands"](#) on page 294
  - [":MARKer:X1Position"](#) on page 278
  - [":MARKer:X2Position"](#) on page 280
  - [":MARKer:XDELta"](#) on page 282
  - [":MEASure:TDELta"](#) on page 545
  - [":MEASure:TSTOP"](#) on page 548

**:MEASure:TSTOp**

**O** (see [page 608](#))

**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 610](#)). 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 280](#)) 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 276
  - "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MARKer:X1Position](#)" on page 278
  - "[:MARKer:X2Position](#)" on page 280
  - "[:MARKer:XDELta](#)" on page 282
  - "[:MEASure:TDELta](#)" on page 545
  - "[:MEASure:TSTArt](#)" on page 547

## :MEASure:TVOLt

**O** (see [page 608](#))

**Query Syntax** :MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH}

<digital channels> ::= {DIGital<d>} for the MSO models

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

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

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

**NOTE**

The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command (see [page 319](#)).

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

## 29 Obsolete and Discontinued Commands

<value> ::= time in seconds of the specified voltage crossing  
in NR3 format

**:MEASure:UPPer**

**O** (see [page 608](#))

**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 298](#)) 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 294
  - "[:MEASure:LOWer](#)" on page 543
  - "[:MEASure:THResholds](#)" on page 546

**:MEASure:VDELta**

**O** (see [page 608](#))

**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 285](#)) instead.

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

<value> ::= delta V value in NR1 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 276
  - "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MARKer:Y1Position](#)" on page 283
  - "[:MARKer:Y2Position](#)" on page 284
  - "[:MARKer:YDELta](#)" on page 285
  - "[:MEASure:TDELta](#)" on page 545
  - "[:MEASure:TSTArt](#)" on page 547



## :MEASure:VSTArt

**O** (see [page 608](#))

**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 610](#)). 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 283](#)) 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 276
  - "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MARKer:Y1Position](#)" on page 283
  - "[:MARKer:Y2Position](#)" on page 284
  - "[:MARKer:YDELta](#)" on page 285
  - "[:MARKer:X1Y1source](#)" on page 279
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:TDELta](#)" on page 545
  - "[:MEASure:TSTArt](#)" on page 547

**:MEASure:VSTOp**

**O** (see [page 608](#))

**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 610](#)). 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 284](#)) 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 276
  - "[Introduction to :MEASure Commands](#)" on page 294
  - "[:MARKer:Y1Position](#)" on page 283
  - "[:MARKer:Y2Position](#)" on page 284
  - "[:MARKer:YDELta](#)" on page 285
  - "[:MARKer:X2Y2source](#)" on page 281
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:TDELta](#)" on page 545
  - "[:MEASure:TSTArt](#)" on page 547

**:MTESt:AMASk:{SAVE | STORe}**

**O** (see [page 608](#))

**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 388](#)) instead.

**See Also** • "Introduction to :MTESt Commands" on page 333

**:MTESt:AVERage**

**O** (see [page 608](#))

**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 169](#)) 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 333
  - "[:MTESt:AVERage:COUNT](#)" on page 557

**:MTESt:AVERAge:COUNT**

**O** (see [page 608](#))

**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 160](#)) 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 333](#)
  - "[:MTESt:AVERAge](#)" on [page 556](#)

## :MTEST:LOAD

**O** (see [page 608](#))

**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 374](#)) instead.

- See Also**
- "Introduction to :MTEST Commands" on page 333
  - ":MTEST:AMASK:{SAVE | STORE}" on page 555

## :MTEST:RUMode

**O** (see [page 608](#))

**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 350](#)) 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 333
  - [":MTEST:RUMode:SOFailure"](#) on page 560

## :MTESt:RUMode:SOFailure

**O** (see [page 608](#))

**Command Syntax** :MTESt:RUMode:SOFailure <on\_off>  
<on\_off> ::= {{1 | ON} | {0 | OFF}}

The :MTESt:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

### NOTE

The :MTESt:RUMode:SOFailure command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MTESt:RMODE:FACTion:STOP command (see [page 354](#)) 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 333
  - "[:MTESt:RUMode](#)" on page 559



**:MTEST:{START | STOP}**

**O** (see [page 608](#))

**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 150](#)) and :STOP command (see [page 154](#)) instead.

**See Also** • "Introduction to :MTEST Commands" on page 333

## :MTESt:TRIGger:SOURce

**O** (see [page 608](#))

**Command Syntax** :MTESt:TRIGger:SOURce <source>

<source> ::= CHANnel<n>

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

The :MTESt:TRIGger:SOURce command sets the channel to use as the trigger.

### NOTE

The :MTESt:TRIGger:SOURce command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the trigger source commands (see [page 423](#)) instead.

**Query Syntax** :MTESt:TRIGger:SOURce?

The :MTESt:TRIGger:SOURce? query returns the currently selected trigger source.

**Return Format** <source> ::= CHAN<n>

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

**See Also** • ["Introduction to :MTESt Commands"](#) on page 333

## :PRINt?

**O** (see [page 608](#))

### 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 228](#)) 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 258
  - [":HARDcopy:FACTors"](#) on page 261
  - [":HARDcopy:GRAYscale"](#) on page 540
  - [":DISPlay:DATA"](#) on page 228

**:SAVE:IMAGe:AREA**

**O** (see [page 608](#))

**Query Syntax** :SAVE:IMAGe:AREA?

The :SAVE:IMAGe:AREA? query returns the selected image area.

When saving images, this query returns SCR (screen). When saving setups or waveform data, this query returns GRAT (graticule) even though graticule images are not saved.

**Return Format** <area><NL>

<area> ::= {GRAT | SCR}

- See Also**
- "[Introduction to :SAVE Commands](#)" on page 380
  - "[:SAVE:IMAGe\[:START\]](#)" on page 383
  - "[:SAVE:IMAGe:FACTors](#)" on page 384
  - "[:SAVE:IMAGe:FORMat](#)" on page 385
  - "[:SAVE:IMAGe:INKSaver](#)" on page 386
  - "[:SAVE:IMAGe:PALette](#)" on page 387

**:TIMEbase:DElay**

**O** (see [page 608](#))

**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 416](#)).

**NOTE**

The :TIMEbase:DElay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POSition command (see [page 414](#)) instead.

**Query Syntax** :TIMEbase:DElay?

The :TIMEbase:DElay query returns the current delay value.

**Return Format** <delay\_value><NL>

<delay\_value> ::= time from trigger to display reference in seconds in NR3 format.

**Example Code**

```
' TIMEBASE_DELAY - Sets the time base delay. This delay
' is the internal time between the trigger event and the
' onscreen delay reference point.

' Set time base delay to 0.0.
myScope.WriteString ":TIMEBASE:DELAY 0.0"
```

See complete example programs at: [Chapter 34](#), “Programming Examples,” starting on page 617

## :TRIGger:THReshold

**O** (see [page 608](#))

**Command Syntax** :TRIGger:THReshold <channel group>, <threshold type> [, <value>]  
 <channel group> ::= {POD1 | POD2}  
 <threshold type> ::= {CMOS | ECL | TTL | USERdef}  
 <value> ::= voltage for USERdef (floating-point number) [Volt type]  
 [Volt type] ::= {V | mV | uV}

The :TRIGger:THReshold command sets the threshold (trigger level) for a pod of 8 digital channels (either digital channels 0 through 7 or 8 through 15). The threshold can be set to a predefined value or to a user-defined value. For the predefined value, the voltage parameter is not required.

**NOTE** This command is only available on the MSO models.

**NOTE** The :TRIGger:THReshold command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :POD<n>:THReshold command (see [page 368](#)), :DIGital<d>:THReshold command (see [page 223](#)), or :TRIGger[:EDGE]:LEVel command (see [page 435](#)).

**Query Syntax** :TRIGger:THReshold? <channel group>

The :TRIGger:THReshold? query returns the voltage and threshold text for analog channel 1 or 2, or POD1 or POD2.

**Return Format** <threshold type>[, <value>]<NL>  
 <threshold type> ::= {CMOS | ECL | TTL | USER}  
 CMOS ::= 2.5V  
 TTL ::= 1.5V  
 ECL ::= -1.3V  
 USERdef ::= range from -8.0V to +8.0V.  
 <value> ::= voltage for USERdef (a floating-point number in NR1).

**:TRIGger:TV:TVMode**

**O** (see [page 608](#))

**Command Syntax** :TRIGger:TV:TVMode <mode>

```
<mode> ::= {FIEld1 | FIEld2 | AFIElds | ALINes | LINE | VERTical
           | LFIeld1 | LFIeld2 | LALTernate | LVERTical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANdard is GENERic. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENERic (see [page 458](#)).

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIeld	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERTical	LINEVert

**NOTE**

The :TRIGger:TV:TVMode command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see [page 455](#)) instead.

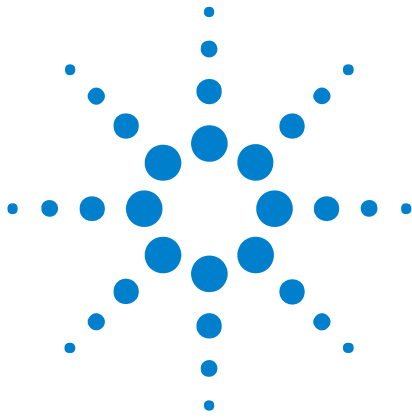
**Query Syntax** :TRIGger:TV:TVMode?

The :TRIGger:TV:TVMode? query returns the TV trigger mode.

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

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
           | LALT | LVER}
```





## 30 Error Messages

**-440, Query UNTERMINATED after indefinite response**

**-430, Query DEADLOCKED**

**-420, Query UNTERMINATED**

**-410, Query INTERRUPTED**

**-400, Query error**

**-340, Calibration failed**

**-330, Self-test failed**

**-321, Out of memory**

**-320, Storage fault**

**-315, Configuration memory lost**



**-314, Save/recall memory lost**

**-313, Calibration memory lost**

**-311, Memory error**

**-310, System error**

**-300, Device specific error**

**-278, Macro header not found**

**-277, Macro redefinition not allowed**

**-276, Macro recursion error**

**-273, Illegal macro label**

**-272, Macro execution error**

**-258, Media protected**

**-257, File name error**

**-256, File name not found**

**-255, Directory full**

**-254, Media full**

**-253, Corrupt media**

**-252, Missing media**

**-251, Missing mass storage**

**-250, Mass storage error**

**-241, Hardware missing**

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

**-240, Hardware error**

**-231, Data questionable**

**-230, Data corrupt or stale**

**-224, Illegal parameter value**

**-223, Too much data**

**-222, Data out of range**

**-221, Settings conflict**

**-220, Parameter error**

**-200, Execution error**

**-183, Invalid inside macro definition**

**-181, Invalid outside macro definition**

**-178, Expression data not allowed**

**-171, Invalid expression**

**-170, Expression error**

**-168, Block data not allowed**

**-161, Invalid block data**

**-158, String data not allowed**

**-151, Invalid string data**

**-150, String data error**

**-148, Character data not allowed**

**-138, Suffix not allowed**

**-134, Suffix too long**

**-131, Invalid suffix**

**-128, Numeric data not allowed**

**-124, Too many digits**

**-123, Exponent too large**

**-121, Invalid character in number**

**-120, Numeric data error**

**-114, Header suffix out of range**

**-113, Undefined header**

**-112, Program mnemonic too long**

**-109, Missing parameter**

**-108, Parameter not allowed**

**-105, GET not allowed**

**-104, Data type error**

**-103, Invalid separator**

**-102, Syntax error**

**-101, Invalid character**

**-100, Command error**

**+10, Software Fault Occurred**

**+100, File Exists**

**+101, End-Of-File Found**

**+102, Read Error**

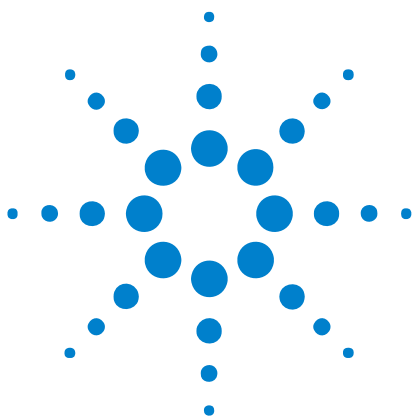
**+103, Write Error****+104, Illegal Operation****+105, Print Canceled****+106, Print Initialization Failed****+107, Invalid Trace File****+108, Compression Error****+109, No Data For Operation**

A remote operation wants some information, but there is no information available. For example, you may request a stored TIFF image using the :DISPlay:DATA? query, but there may be no image stored.

**+112, Unknown File Type****+113, Directory Not Supported**







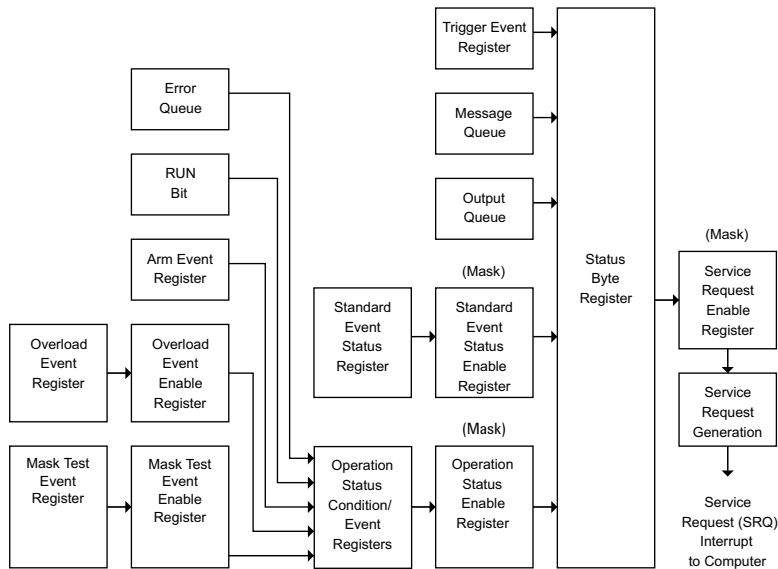
## 31 Status Reporting

Status Reporting Data Structures	579
Status Byte Register (STB)	581
Service Request Enable Register (SRE)	583
Trigger Event Register (TER)	584
Output Queue	585
Message Queue	586
(Standard) Event Status Register (ESR)	587
(Standard) Event Status Enable Register (ESE)	588
Error Queue	589
Operation Status Event Register (:OPERRegister[:EVENTt])	590
Operation Status Condition Register (:OPERRegister:CONDition)	591
Arm Event Register (AER)	592
Overload Event Register (:OVLRegister)	593
Mask Test Event Event Register (:MTERRegister[:EVENTt])	594
Clearing Registers and Queues	595
Status Reporting Decision Chart	596

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting (for example, the Status Byte Register and the Standard Event Status Register). There are also instrument-defined structures and bits (for example, the Operation Status Event Register and the Overload Event Register).

An overview of the oscilloscope's status reporting structure is shown in the following block diagram. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.





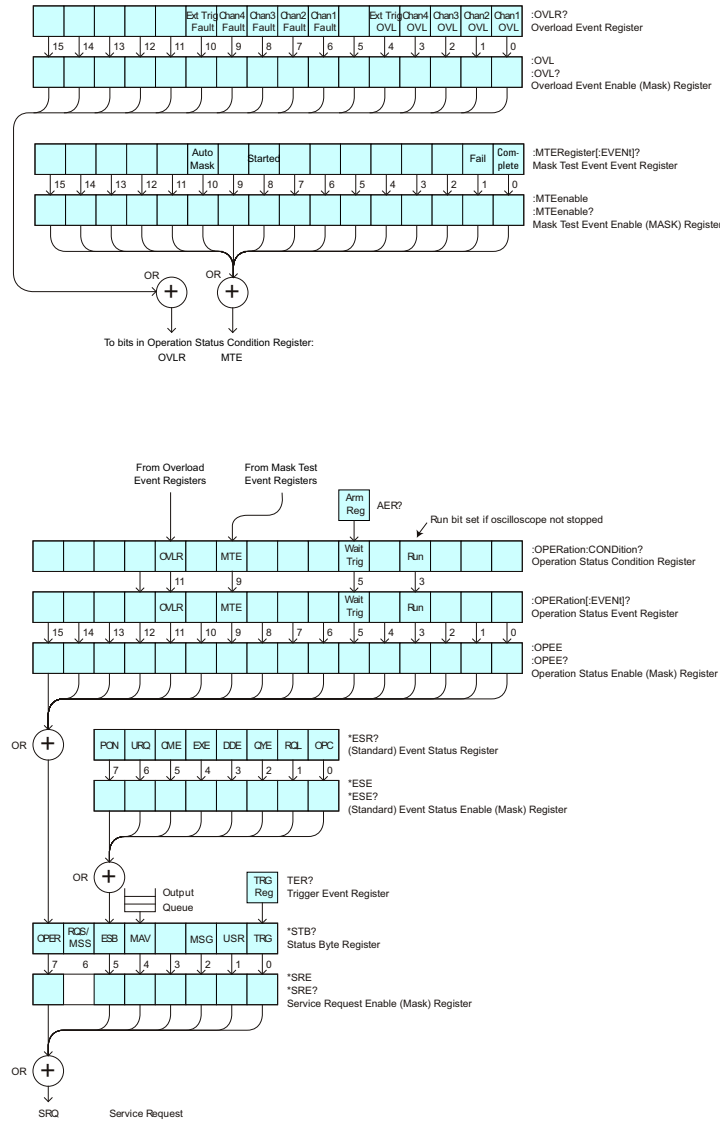
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled with the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The \*CLS command clears all event registers and all queues except the output queue. If you send \*CLS immediately after a program message terminator, the output queue is also cleared.

## Status Reporting Data Structures

The following figure shows how the status register bits are masked and logically OR'ed to generate service requests (SRQ) on particular events.



The status register bits are described in more detail in the following tables:

- Table 34
- Table 32
- Table 39
- Table 40

- [Table 42](#)
- [Table 37](#)

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 (:OPERRegister[: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.
OVLRL bit	bit 11	Is set whenever a 50Ω input overload occurs.

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

## Operation Status Condition Register (:OPERRegister: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.
OVLRL bit	bit 11	Is set whenever a 50 $\Omega$ input overload occurs.

The :OPERRegister: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 Fault	bit 6	Fault has occurred on Channel 1 input.
Channel 2 Fault	bit 7	Fault has occurred on Channel 2 input.
Channel 3 Fault	bit 8	Fault has occurred on Channel 3 input.
Channel 4 Fault	bit 9	Fault has occurred on Channel 4 input.
External Trigger Fault	bit 10	Fault has occurred on External Trigger input.

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

The Mask Test Event Event Register register hosts these bits:

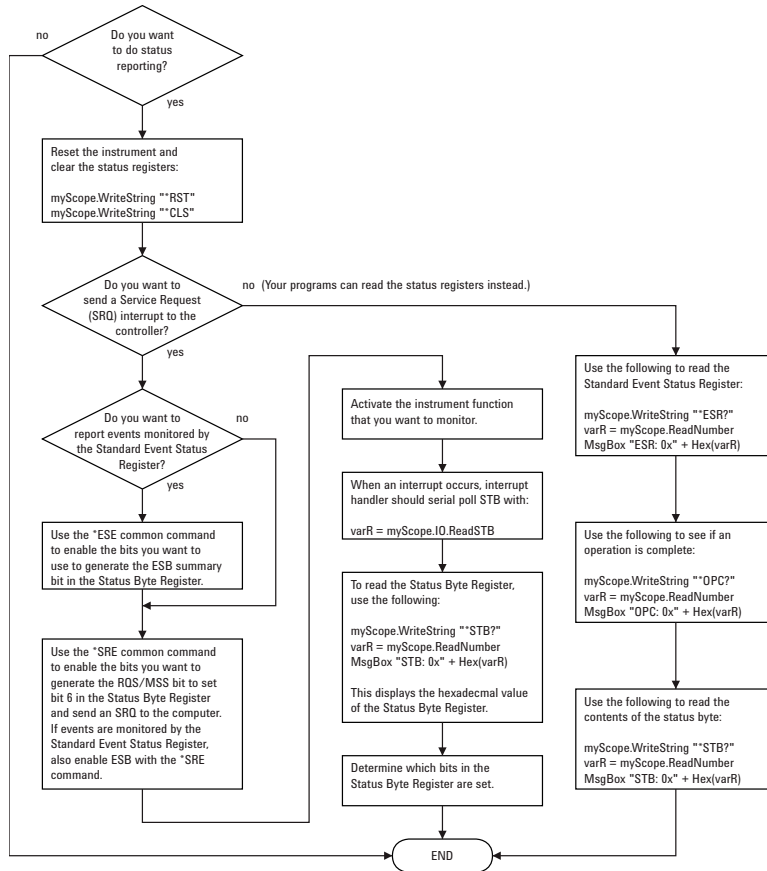
Name	Location	Description
Complete	bit 0	Is set when the mask test is complete.
Fail	bit 1	Is set when there is a mask test failure.
Started	bit 8	Is set when mask testing is started.
Auto Mask	bit 10	Is set when auto mask creation is completed.

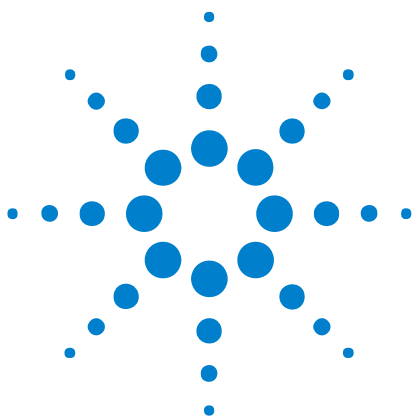
The :MTERegister[:EVENT]? query returns the value of, and clears, the Mask Test Event Event Register.

## 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





## 32 Synchronizing Acquisitions

- Synchronization in the Programming Flow [598](#)
- Blocking Synchronization [599](#)
- Polling Synchronization With Timeout [600](#)
- Synchronizing with a Single-Shot Device Under Test (DUT) [602](#)
- Synchronization with an Averaging Acquisition [604](#)

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 598](#)).
- 2 Acquire a waveform (see [page 598](#)).
- 3 Retrieve results (see [page 598](#)).

### Set Up the Oscilloscope

Before making changes to the oscilloscope setup, it is best to make sure it is stopped using the :STOP command followed by the \*OPC? query.

#### NOTE

It is not necessary to use \*OPC?, hard coded waits, or status checking when setting up the oscilloscope. After the oscilloscope is configured, it is ready for an acquisition.

### Acquire a Waveform

When acquiring a waveform there are two possible methods used to wait for the acquisition to complete. These methods are blocking and polling. The table below details when each method should be chosen and why.

	Blocking Wait	Polling Wait
<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 599.	See " <a href="#">Polling Synchronization With Timeout</a> " on page 600.

### 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 ":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

```

## Synchronizing with a Single-Shot Device Under Test (DUT)

The examples in ["Blocking Synchronization"](#) on page 599 and ["Polling Synchronization With Timeout"](#) on page 600 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 600 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"

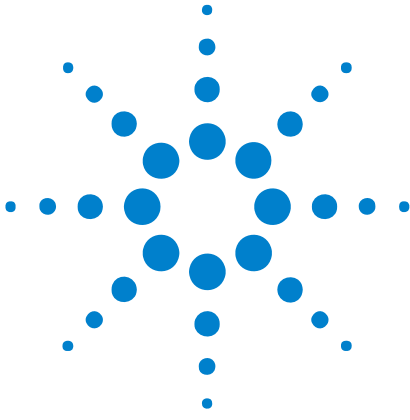
Exit Sub

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

End Sub

```





## 33

# More About Oscilloscope Commands

Command Classifications [608](#)

Valid Command/Query Strings [609](#)

Query Return Values [615](#)

All Oscilloscope Commands Are Sequential [616](#)



## 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 608
- "Non-Core Commands" on page 608
- "Obsolete Commands" on page 608

### **C** Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent InfiniiVision oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

### **N** Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all Agilent InfiniiVision oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

### **O** Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

- [Chapter 29](#), "Obsolete and Discontinued Commands," starting on page 521

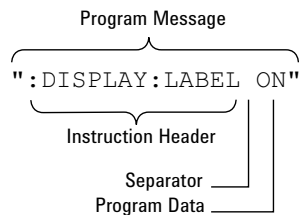


## Valid Command/Query Strings

- "Program Message Syntax" on page 609
- "Duplicate Mnemonics" on page 613
- "Tree Traversal Rules and Multiple Commands" on page 613

### 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 610), and upper and/or lower case ASCII characters may be used. (Query responses, however, are always returned in upper case.)

Instructions are composed of two main parts:

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

**Instruction Header** The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument.

`" :DISPlay:LABel ON"` is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, `" :DISPlay:LABel?"`. Many instructions can be used as either commands or queries, depending

on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- "Simple Command Headers" on page 611
- "Compound Command Headers" on page 611
- "Common Command Headers" on page 611

**White Space  
(Separator)**

White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. White space is defined as one or more space characters. ASCII defines a space to be character 32 (in decimal).

**Program Data**

Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. "Program Data Syntax Rules" on page 612 describes all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas(.). Spaces can be added around the commas to improve readability.

**Program  
Message  
Terminator**

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

**NOTE**

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

#### Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGe	RANG
PATtern	PATT
TIMebase	TIM
DELay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

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

### Simple Command Headers

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

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

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGitize CHANnel1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

### Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLimit ON

### Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (\*) and the command header. \*CLS is an example of a common command header.

### Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data><terminator>
```

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

```
<program mnemonic><separator><data>,<data><terminator>
```

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

#### Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEbase:MODE command can be set to normal, zoomed (delayed), XY, or ROLL. The character program data in this case may be MAIN, WINDow, XY, or ROLL. The command :TIMEbase:MODE WINDow sets the time base mode to zoomed.

The available mnemonics for character program data are always included with the command's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

#### Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEbase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

$$28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.$$

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

## Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGE may be used to change the vertical range or to change the horizontal range:

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

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMEbase:RANGe 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMEbase are subsystem selectors and determine which range is being modified.

## Tree Traversal Rules and Multiple Commands

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

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.
- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEbase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Agilent VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

```
:<subsystem>:<function><separator><data>;<function><separator><data><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**

```
myScope.WriteString ":TIMEbase:REFerence CENTER;POSition 0.00001"
```

or

```
myScope.WriteString ":TIMEbase:REFerence CENTER"  
myScope.WriteString ":TIMEbase:POSition 0.00001"
```

**NOTE**

In the first line of example 2, the subsystem selector is implied for the POSition command in the compound command. The POSition command must be in the same program message as the REFerence command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMEbase: before the POSition command as shown in the second part of example 2. The space after POSition is required.

**Example 3:  
Selecting  
Multiple  
Subsystems**

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

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

For example:

```
myScope.WriteString ":TIMEbase:REFerence CENTER;:DISPlay:VECTors ON"
```

**NOTE**

The leading colon before DISPlay:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:VECTors ON command. The space between REFerence and CENTer is required; so is the space between VECTors and ON.

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

## Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMEbase:RANGe? places the current time base setting in the output queue. When using the Agilent VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String
myScope.WriteString ":TIMEbase:RANGe?"
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

### NOTE

**Read Query Results Before Sending Another Command.** Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

### Infinity Representation

The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

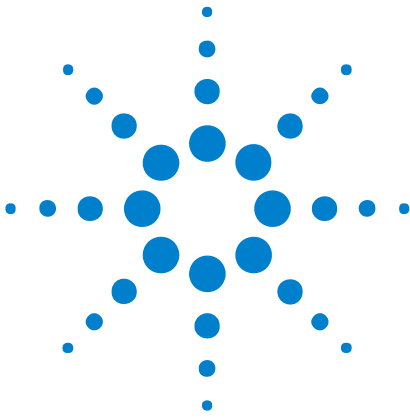
## All Oscilloscope Commands Are Sequential

IEEE 488.2 makes the distinction between sequential and overlapped commands:

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

All of the oscilloscope commands are sequential.





## 34 Programming Examples

VISA COM Examples [618](#)

VISA Examples [645](#)

SICL Examples [686](#)

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.



## VISA COM Examples

- ["VISA COM Example in Visual Basic"](#) on page 618
- ["VISA COM Example in C#"](#) on page 627
- ["VISA COM Example in Visual Basic .NET"](#) on page 636

### VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Reference the Agilent VISA COM library:
  - a Choose **Tools>References...** from the main menu.
  - b In the References dialog, check the "VISA COM 3.0 Type Library".
  - c Click **OK**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```
'
' Agilent VISA COM Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' -----

Option Explicit

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

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -----

Sub Main()

    On Error GoTo VisaComError
```

```

' Create the VISA COM I/O resource.
Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488
Set myScope.IO = _
    myMgr.Open("USB0::0x0957::0x17A6::US50210029::0::INSTR")
myScope.IO.Clear ' Clear the interface.
myScope.IO.Timeout = 10000 ' Set I/O communication timeout.

' Initialize - start from a known state.
Initialize

' Capture data.
Capture

' Analyze the captured waveform.
Analyze

Exit Sub

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

End Sub

'
' Initialize the oscilloscope to a known state.
' -----

Private Sub Initialize()

    On Error GoTo VisaComError

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    Debug.Print "Identification string: " + strQueryResult

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

Exit Sub

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

End Sub

'
' Capture the waveform.
' -----

Private Sub Capture()

    On Error GoTo VisaComError

```

```

' Use auto-scale to automatically configure oscilloscope.
' -----
DoCommand ":AUToscale"

' Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
    DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
    DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
    DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
    DoQueryString(":TRIGger:EDGE:SLOPe?")

' Save oscilloscope configuration.
' -----
varQueryResult = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varQueryResult ' Write data.
Close hFile ' Close file.
Debug.Print "Setup bytes saved: " + CStr(LenB(varQueryResult))

' Change settings with individual commands:
' -----

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and offset.
DoCommand ":TIMEbase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALe?")

DoCommand ":TIMEbase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSition?")

```

```

' Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESOLUTION).
DoCommand ":ACQUIRE:TYPE NORMAL"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQUIRE:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile ' Open file for input.
Get hFile, , varSetupString ' Read data.
Close hFile ' Close file.
' Write learn string back to oscilloscope using ":SYSTEM:SETUP"
' command:
DoCommandIEEEBlock ":SYSTEM:SETUP", varSetupString
Debug.Print "Setup bytes restored: " + CStr(LenB(varSetupString))

' Capture an acquisition using :DIGITIZE.
' -----
DoCommand ":DIGITIZE CHANNEL1"

Exit Sub

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

End Sub

'
' Analyze the captured waveform.
' -----

Private Sub Analyze()

On Error GoTo VisaComError

' Make a couple of measurements.
' -----
DoCommand ":MEASURE:SOURCE CHANNEL1"
Debug.Print "Measure source: " + _
    DoQueryString(":MEASURE:SOURCE?")

DoCommand ":MEASURE:FREQUENCY"
varQueryResult = DoQueryNumber(":MEASURE:FREQUENCY?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"

DoCommand ":MEASURE:VAMPLITUDE"
varQueryResult = DoQueryNumber(":MEASURE:VAMPLITUDE?")
MsgBox "Vertical amplitude:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' Download the screen image.
' -----
' Get screen image.

```

```

DoCommand ":HARDcopy:INKSaver OFF"
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG, COLor")

' Save screen image to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath ' Remove file if it exists.
End If

Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , byteData ' Write data.
Close hFile ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
    " bytes) written to " + strPath

' Download waveform data.
' -----

' Set the waveform points mode.
DoCommand ":WAVeform:POINts:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINts:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVeform:PREAmble?")

```

```

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 4 Then
    Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMAl"
ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERAge"
ElseIf intType = 3 Then
    Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
varQueryResult = DoQueryIEEEBlock_UI1(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(UBound(varQueryResult) + 1)

```

```

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
Dim lngI As Long

For lngI = 0 To UBound(varQueryResult)
    lngDataValue = varQueryResult(lngI)

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngDataValue - lngYReference) * _
            sngYIncrement) + sngYOrigin)

Next lngI

' Close output file.
Close hFile ' Close file.
MsgBox "Waveform format BYTE data written to " + _
    "c:\scope\data\waveform_data.csv."

Exit Sub

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

End Sub

Private Sub DoCommand(command As String)

    On Error GoTo VisaComError

    myScope.WriteString command
    CheckInstrumentErrors

Exit Sub

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)

    On Error GoTo VisaComError

```



```

Dim strErrors As String

myScope.WriteIEEEBlock command, data
CheckInstrumentErrors

Exit Sub

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Sub

Private Function DoQueryString(query As String) As String

    On Error GoTo VisaComError

    myScope.WriteString query
    DoQueryString = myScope.ReadString
    CheckInstrumentErrors

    Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Function DoQueryNumber(query As String) As Variant

    On Error GoTo VisaComError

    myScope.WriteString query
    DoQueryNumber = myScope.ReadNumber
    CheckInstrumentErrors

    Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

    On Error GoTo VisaComError

    Dim strErrors As String

```

```

myScope.WriteString query
DoQueryNumbers = myScope.ReadList
CheckInstrumentErrors

Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

On Error GoTo VisaComError

myScope.WriteString query
DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckInstrumentErrors

Exit Function

VisaComError:
MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
    Err.Source + ", " + _
    Err.Description, vbExclamation, "VISA COM Error"
End

End Function

Private Sub CheckInstrumentErrors()

On Error GoTo VisaComError

Dim strErrVal As String
Dim strOut As String

myScope.WriteString ":SYSTem:ERROr?" ' Query any errors data.
strErrVal = myScope.ReadString ' Read: Errnum,"Error String".
While Val(strErrVal) <> 0 ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    myScope.WriteString ":SYSTem:ERROr?" ' Request error message.
    strErrVal = myScope.ReadString ' Read error message.
Wend

If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    myScope.FlushWrite (False)
    myScope.FlushRead

End If

Exit Sub

VisaComError:

```

```

        MsgBox "VISA COM Error: " + vbCrLf + Err.Description
    End Sub

```

## VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual C#, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add a reference to the VISA COM 3.0 Type Library:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add Reference...**
  - c In the Add Reference dialog, select the **COM** tab.
  - d Select **VISA COM 3.0 Type Library**; then click **OK**.
- 6 Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```

/*
 * Agilent VISA COM Example in C#
 * -----
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * -----
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
    class VisaComInstrumentApp
    {
        private static VisaComInstrument myScope;

        public static void Main(string[] args)
        {
            try
            {
                myScope = new
                    VisaComInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR")

```

```

);

    myScope.SetTimeoutSeconds(10);

    // Initialize - start from a known state.
    Initialize();

    // Capture data.
    Capture();

    // Analyze the captured waveform.
    Analyze();
}
catch (System.ApplicationException err)
{
    Console.WriteLine("*** VISA COM Error : " + err.Message);
}
catch (System.SystemException err)
{
    Console.WriteLine("*** System Error Message : " + err.Message);
}
catch (System.Exception err)
{
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
}
finally
{
    myScope.Close();
}
}

/*
 * Initialize the oscilloscope to a known state.
 * -----
 */
private static void Initialize()
{
    string strResults;

    // Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?");
    Console.WriteLine("*IDN? result is: {0}", strResults);

    // Clear status and load the default setup.
    myScope.DoCommand("*CLS");
    myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -----
 */
private static void Capture()
{
    // Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale");
}

```

```

// Set trigger mode (EDGE, PULSe, PATTeRn, etc., and input source.
myScope.DoCommand(":TRIGger:MODE EDGE");
Console.WriteLine("Trigger mode: {0}",
    myScope.DoQueryString(":TRIGger:MODE?"));

// Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
Console.WriteLine("Trigger edge source: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
Console.WriteLine("Trigger edge level: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

// Save oscilloscope configuration.
byte[] ResultsArray; // Results array.
int nLength; // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
nLength = ResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet -1.5");
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"));

// Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:SCALe 0.0002");
Console.WriteLine("Timebase scale: {0}",
    myScope.DoQueryString(":TIMebase:SCALe?"));

myScope.DoCommand(":TIMebase:POSition 0.0");
Console.WriteLine("Timebase position: {0}",
    myScope.DoQueryString(":TIMebase:POSition?"));

// Set the acquisition type (NORMal, PEAK, AVERAge, or HRESolution
).
myScope.DoCommand(":ACQuire:TYPE NORMal");

```

```

Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQUIRE:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] dataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.stp";
dataArray = File.ReadAllBytes(strPath);
nBytesWritten = dataArray.Length;

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTEM:SETUP", dataArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGITIZE.
myScope.DoCommand(":DIGITIZE CHANNEL1");
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    byte[] resultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // -----
    myScope.DoCommand(":MEASURE:SOURCE CHANNEL1");
    Console.WriteLine("Measure source: {0}",
        myScope.DoQueryString(":MEASURE:SOURCE?"));

    double fResult;
    myScope.DoCommand(":MEASURE:FREQUENCY");
    fResult = myScope.DoQueryNumber(":MEASURE:FREQUENCY?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    myScope.DoCommand(":MEASURE:VAMPLITUDE");
    fResult = myScope.DoQueryNumber(":MEASURE:VAMPLITUDE?");
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

    // Download the screen image.
    // -----
    myScope.DoCommand(":HARDCOPY:INKSAVER OFF");

    // Get the screen data.
    resultsArray =
        myScope.DoQueryIEEEBlock(":DISPLAY:DATA? PNG, COLOR");
    nLength = resultsArray.Length;

    // Store the screen data to a file.
    strPath = "c:\\scope\\data\\screen.png";
    FileStream fStream = File.Open(strPath, FileMode.Create);

```

```

fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// -----

// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTs:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINTs:MODE?"));

// Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINTs?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREAmble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
    Console.WriteLine("Acquire type: NORMal");
}
else if (fType == 1.0)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
    Console.WriteLine("Acquire type: AVERage");
}

```

```

    }
    else if (fType == 3.0)
    {
        Console.WriteLine("Acquire type: HRESolution");
    }

    double fPoints = fResultsArray[2];
    Console.WriteLine("Waveform points: {0:e}", fPoints);

    double fCount = fResultsArray[3];
    Console.WriteLine("Waveform average count: {0:e}", fCount);

    double fXincrement = fResultsArray[4];
    Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

    double fXorigin = fResultsArray[5];
    Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

    double fXreference = fResultsArray[6];
    Console.WriteLine("Waveform X reference: {0:e}", fXreference);

    double fYincrement = fResultsArray[7];
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

    double fYorigin = fResultsArray[8];
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

    double fYreference = fResultsArray[9];
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

    // Read waveform data.
    ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?");
    nLength = ResultsArray.Length;
    Console.WriteLine("Number of data values: {0}", nLength);

    // Set up output file:
    strPath = "c:\\scope\\data\\waveform_data.csv";
    if (File.Exists(strPath)) File.Delete(strPath);

    // Open file for output.
    StreamWriter writer = File.CreateText(strPath);

    // Output waveform data in CSV format.
    for (int i = 0; i < nLength - 1; i++)
        writer.WriteLine("{0:f9}, {1:f6}",
            fXorigin + ((float)i * fXincrement),
            (((float)ResultsArray[i] - fYreference)
            * fYincrement) + fYorigin);

    // Close output file.
    writer.Close();
    Console.WriteLine("Waveform format BYTE data written to {0}",
        strPath);
}
}

class VisaComInstrument

```



```

{
private ResourceManagerClass m_ResourceManager;
private FormattedIO488Class m_IoObject;
private string m_strVisaAddress;

// Constructor.
public VisaComInstrument(string strVisaAddress)
{
// Save VISA address in member variable.
m_strVisaAddress = strVisaAddress;

// Open the default VISA COM IO object.
OpenIo();

// Clear the interface.
m_IoObject.IO.Clear();
}

public void DoCommand(string strCommand)
{
// Send the command.
m_IoObject.WriteString(strCommand, true);

// Check for inst errors.
CheckInstrumentErrors(strCommand);
}

public void DoCommandIEEEBlock(string strCommand,
byte[] DataArray)
{
// Send the command to the device.
m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

// Check for inst errors.
CheckInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
// Send the query.
m_IoObject.WriteString(strQuery, true);

// Get the result string.
string strResults;
strResults = m_IoObject.ReadString();

// Check for inst errors.
CheckInstrumentErrors(strQuery);

// Return results string.
return strResults;
}

public double DoQueryNumber(string strQuery)
{
// Send the query.
m_IoObject.WriteString(strQuery, true);
}

```

```

    // Get the result number.
    double fResult;
    fResult = (double)m_IoObject.ReadNumber(
        IEEEASCIIType.ASCIIType_R8, true);

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return result number.
    return fResult;
}

public double[] DoQueryNumbers(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result numbers.
    double[] fResultsArray;
    fResultsArray = (double[])m_IoObject.ReadList(
        IEEEASCIIType.ASCIIType_R8, ",;");

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return result numbers.
    return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the results array.
    System.Threading.Thread.Sleep(2000); // Delay before reading.
    byte[] ResultsArray;
    ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
        IEEEBinaryType.BinaryType_UI1, false, true);

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return results array.
    return ResultsArray;
}

private void CheckInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    string strInstrumentError;
    bool bFirstError = true;

    do // While not "0,No error".
    {
        m_IoObject.WriteString(":SYSTEM:ERROR?", true);

```

```

        strInstrumentError = m_IoObject.ReadString();

        if (!strInstrumentError.ToString().StartsWith("+0, "))
        {
            if (bFirstError)
            {
                Console.WriteLine("ERROR(s) for command '{0}': ",
                    strCommand);
                bFirstError = false;
            }
            Console.Write(strInstrumentError);
        }
    } while (!strInstrumentError.ToString().StartsWith("+0, "));
}

private void OpenIo()
{
    m_ResourceManager = new ResourceManagerClass();
    m_IoObject = new FormattedIO488Class();

    // Open the default VISA COM IO object.
    try
    {
        m_IoObject.IO =
            (IMessage)m_ResourceManager.Open(m_strVisaAddress,
                AccessMode.NO_LOCK, 0, "");
    }
    catch (Exception e)
    {
        Console.WriteLine("An error occurred: {0}", e.Message);
    }
}

public void SetTimeoutSeconds(int nSeconds)
{
    m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
    try
    {
        m_IoObject.IO.Close();
    }
    catch { }

    try
    {
        Marshal.ReleaseComObject(m_IoObject);
    }
    catch { }

    try
    {
        Marshal.ReleaseComObject(m_ResourceManager);
    }
    catch { }
}

```

```

    }
  }
}

```

## VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual Basic, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add a reference to the VISA COM 3.0 Type Library:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add Reference...**
  - c In the Add Reference dialog, select the **COM** tab.
  - d Select **VISA COM 3.0 Type Library**; then click **OK**.
  - e Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.
- 6 Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```

'
' Agilent VISA COM Example in Visual Basic .NET
' -----
' This program illustrates a few commonly used programming
' features of your Agilent oscilloscope.
' -----

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New _
          VisaComInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR"

```

```

)
    myScope.SetTimeoutSeconds(10)

    ' Initialize - start from a known state.
    Initialize()

    ' Capture data.
    Capture()

    ' Analyze the captured waveform.
    Analyze()

Catch err As System.ApplicationException
    Console.WriteLine("*** VISA Error Message : " + err.Message)
Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
Finally
    myScope.Close()
End Try
End Sub

' Initialize the oscilloscope to a known state.
' -----

Private Shared Sub Initialize()
    Dim strResults As String

    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Clear status and load the default setup.
    myScope.DoCommand("*CLS")
    myScope.DoCommand("*RST")

End Sub

' Capture the waveform.
' -----

Private Shared Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE")
    Console.WriteLine("Trigger mode: {0}", _
        myScope.DoQueryString(":TRIGger:MODE?"))

    ' Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
    Console.WriteLine("Trigger edge source: {0}", _
        myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

```

```

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte() ' Results array.
Dim nLength As Integer ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMebase:SCALe?"))

myScope.DoCommand(":TIMebase:POSition 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMebase:POSition?"))

' Set the acquisition type (NORMal, PEAK, AVERAge, or HRESolution)

myScope.DoCommand(":ACQuire:TYPE NORMal")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim dataArray As Byte()
Dim nBytesWritten As Integer

' Read setup string from file.

```

```

strPath = "c:\scope\config\setup.stp"
dataArray = File.ReadAllBytes(strPath)
nBytesWritten = dataArray.Length

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", dataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGitize.
myScope.DoCommand(":DIGitize CHANnel1")

End Sub

' Analyze the captured waveform.
' -----

Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte() ' Results array.
    Dim nLength As Integer ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
        myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    myScope.DoCommand(":MEASure:VAMPlitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

    ' Download the screen image.
    ' -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF")

    ' Get the screen data.
    ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor
")
    nLength = ResultsArray.Length

    ' Store the screen data to a file.
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Screen image ({0} bytes) written to {1}", _
        nLength, strPath)

    ' Download waveform data.
    ' -----

```

```

' Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINTs:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVEform:POINTs:MODE?"))

' Get the number of waveform points available.
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVEform:POINTs?"))

' Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVEform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVEform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
    Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
    Console.WriteLine("Acquire type: NORMAl")
ElseIf fType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
    Console.WriteLine("Acquire type: AVERAge")
ElseIf fType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)

```



```

Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?")
nLength = ResultsArray.Length
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
        fXorigin + (CSng(index) * fXincrement), _
        ((CSng(ResultsArray(index)) - fYreference) _
        * fYincrement) + fYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
    strPath)

End Sub

End Class

Class VisaComInstrument
    Private m_ResourceManager As ResourceManagerClass
    Private m_IOException As FormattedIO488Class
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)

        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA COM IO object.
        OpenIo()
    End Sub
End Class

```

```

    ' Clear the interface.
    m_IoObject.IO.Clear()

End Sub

Public Sub DoCommand(ByVal strCommand As String)

    ' Send the command.
    m_IoObject.WriteString(strCommand, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal dataArray As Byte())

    ' Send the command to the device.
    m_IoObject.WriteIEEEBlock(strCommand, dataArray, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result string.
    Dim strResults As String
    strResults = m_IoObject.ReadString()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return results string.
    Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result number.
    Dim fResult As Double
    fResult = _
        CDb1(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return result number.
    Return fResult
End Function

```

```

Public Function DoQueryNumbers(ByVal strQuery As String) As _
    Double()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result numbers.
    Dim fResultsArray As Double()
    fResultsArray = _
        m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return result numbers.
    Return fResultsArray
End Function

Public _
    Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the results array.
    System.Threading.Thread.Sleep(2000) ' Delay before reading data.
    Dim ResultsArray As Byte()
    ResultsArray = _
        m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
            False, True)

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return results array.
    Return ResultsArray
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As String
    Dim bFirstError As Boolean = True
    Do ' While not "0,No error".
        m_IoObject.WriteString(":SYSTem:ERRor?", True)
        strInstrumentError = m_IoObject.ReadString()

        If Not strInstrumentError.ToString().StartsWith("+0,") Then
            If bFirstError Then
                Console.WriteLine("ERROR(s) for command '{0}': ", _
                    strCommand)
                bFirstError = False
            End If
            Console.Write(strInstrumentError)
        End If
    Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenIo()
    m_ResourceManager = New ResourceManagerClass()

```

## 34 Programming Examples

```
m_IoObject = New FormattedIO488Class()

' Open the default VISA COM IO object.
Try
    m_IoObject.IO = _
        DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
            AccessMode.NO_LOCK, 0, ""), IMessage)
Catch e As Exception
    Console.WriteLine("An error occurred: {0}", e.Message)
End Try
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    m_IoObject.IO.Timeout = nSeconds * 1000
End Sub

Public Sub Close()
    Try
        m_IoObject.IO.Close()
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_IoObject)
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_ResourceManager)
    Catch
    End Try
End Sub
End Class
End Namespace
```

## VISA Examples

- "VISA Example in C" on page 645
- "VISA Example in Visual Basic" on page 654
- "VISA Example in C#" on page 664
- "VISA Example in Visual Basic .NET" on page 675

### VISA Example in C

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual C++, Win32, Win32 Console Application project.
- 3 In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
- 4 Cut-and-paste the code that follows into a file named "example.c" in the project directory.
- 5 In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
- 6 Edit the program to use the VISA address of your oscilloscope.
- 7 Choose **Project > Properties...** In the Property Pages dialog, update these project settings:
  - a Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.
  - b Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
  - c Click **OK** to close the Property Pages dialog.
- 8 Add the include files and library files search paths:
  - a Choose **Tools > Options...**
  - b In the Options dialog, select **VC++ Directories** under Projects and Solutions.
  - c Show directories for **Include files**, and add the include directory (for example, Program Files\IVI Foundation\VISA\WinNT\include).
  - d Show directories for **Library files**, and add the library files directory (for example, Program Files\IVI Foundation\VISA\WinNT\lib\msc).
  - e Click **OK** to close the Options dialog.
- 9 Build and run the program.

```
/*
 * Agilent VISA Example in C
 * -----
```

```

* This program illustrates a few commonly-used programming
* features of your Agilent oscilloscope.
*/

#include <stdio.h>           /* For printf(). */
#include <string.h>         /* For strcpy(), strcat(). */
#include <time.h>           /* For clock(). */
#include <visa.h>           /* Agilent VISA routines. */

#define VISA_ADDRESS "USB0::0x0957::0x17A6::US50210029::0::INSTR"
#define IEEEBLOCK_SPACE 500000

/* Function prototypes */
void initialize(void);      /* Initialize to known state. */
void capture(void);        /* Capture the waveform. */
void analyze(void);        /* Analyze the captured waveform. */

void do_command(char *command); /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query); /* Query for string. */
void do_query_number(char *query); /* Query for number. */
void do_query_numbers(char *query); /* Query for numbers. */
int do_query_ieeeblock(char *query); /* Query for IEEE block. */
void check_instrument_errors(); /* Check for inst errors. */
void error_handler(); /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi; /* Device session ID. */
ViStatus err; /* VISA function return value. */
char str_result[256] = {0}; /* Result from do_query_string(). */
double num_result; /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /* Result from
do_query_ieeeblock(). */
double dbl_results[10]; /* Result from do_query_numbers(). */

/* Main Program
* ----- */
void main(void)
{
    /* Open the default resource manager session. */
    err = viOpenDefaultRM(&defaultRM);
    if (err != VI_SUCCESS) error_handler();

    /* Open the session using the oscilloscope's VISA address. */
    err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
    if (err != VI_SUCCESS) error_handler();

    /* Set the I/O timeout to fifteen seconds. */
    err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
    if (err != VI_SUCCESS) error_handler();

    /* Initialize - start from a known state. */
    initialize();

    /* Capture data. */
    capture();
}

```

```

/* Analyze the captured waveform. */
analyze();

/* Close the vi session and the resource manager session. */
viClose(vi);
viClose(defaultRM);
}

/* Initialize the oscilloscope to a known state.
 * ----- */
void initialize (void)
{
    /* Clear the interface. */
    err = viClear(vi);
    if (err != VI_SUCCESS) error_handler();

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * ----- */
void capture (void)
{
    int num_bytes;
    FILE *fp;

    /* Use auto-scale to automatically configure oscilloscope. */
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATtern, etc., and input source. */
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURCe CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);

    do_command(":TRIGger:EDGE:LEVel 1.5");
    do_query_string(":TRIGger:EDGE:LEVel?");
    printf("Trigger edge level: %s\n", str_result);

    do_command(":TRIGger:EDGE:SLOPe POSitive");
    do_query_string(":TRIGger:EDGE:SLOPe?");
    printf("Trigger edge slope: %s\n", str_result);

    /* Save oscilloscope configuration. */

    /* Read system setup. */
    num_bytes = do_query_ieeeblock(":SYSTem:SETup?");

```

```

printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALE 0.05");
do_query_string(":CHANnel1:SCALE?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and offset. */
do_command(":TIMEbase:SCALE 0.0002");
do_query_string(":TIMEbase:SCALE?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMEbase:POSition 0.0");
do_query_string(":TIMEbase:POSition?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESOLUTION). *
/
do_command(":ACquire:TYPE NORMAL");
do_query_string(":ACquire:TYPE?");
printf("Acquire type: %s\n", str_result);

/* Or, configure by loading a previously saved setup. */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTEM:SETup", num_bytes);
printf("Restored setup string (%d bytes).\n", num_bytes);

/* Capture an acquisition using :DIGitize. */
do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
* ----- */
void analyze (void)

```



```

{
double wav_format;
double acq_type;
double wav_points;
double avg_count;
double x_increment;
double x_origin;
double x_reference;
double y_increment;
double y_origin;
double y_reference;

FILE *fp;
int num_bytes; /* Number of bytes returned from instrument. */
int i;

/* Make a couple of measurements.
 * ----- */
do_command(":MEASure:SOURce CHANnel1");
do_query_string(":MEASure:SOURce?");
printf("Measure source: %s\n", str_result);

do_command(":MEASure:FREQuency");
do_query_number(":MEASure:FREQuency?");
printf("Frequency: %.4f kHz\n", num_result / 1000);

do_command(":MEASure:VAMPlitude");
do_query_number(":MEASure:VAMPlitude?");
printf("Vertical amplitude: %.2f V\n", num_result);

/* Download the screen image.
 * ----- */
do_command(":HARDcopy:INKSaver OFF");

/* Read screen image. */
num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLor");
printf("Screen image bytes: %d\n", num_bytes);

/* Write screen image bytes to file. */
fp = fopen("c:\\scope\\data\\screen.png", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
fp);
fclose(fp);
printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.bmp.\n");

/* Download waveform data.
 * ----- */

/* Set the waveform points mode. */
do_command(":WAVEform:POINTs:MODE RAW");
do_query_string(":WAVEform:POINTs:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_query_string(":WAVEform:POINTs?");
printf("Waveform points available: %s\n", str_result);

```

```

/* Set the waveform source. */
do_command(":WAVEform:SOURce CHANnel");
do_query_string(":WAVEform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVEform:FORMat BYTE");
do_query_string(":WAVEform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVEform:PREAmble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}
else if (acq_type == 3.0)
{
    printf("Acquire type: HRESolution\n");
}

wav_points = dbl_results[2];
printf("Waveform points: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);

x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);

```

```

x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);

y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);

y_origin = dbl_results[8];
printf("Waveform Y origin: %e\n", y_origin);

y_reference = dbl_results[9];
printf("Waveform Y reference: %e\n", y_reference);

/* Read waveform data. */
num_bytes = do_query_ieeeblock(":WAVEform:DATA?");
printf("Number of data values: %d\n", num_bytes);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

/* Output waveform data in CSV format. */
for (i = 0; i < num_bytes - 1; i++)
{
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
        x_origin + ((float)i * x_increment),
        (((float)ieeeblock_data[i] - y_reference) * y_increment)
        + y_origin);
}

/* Close output file. */
fclose(fp);
printf("Waveform format BYTE data written to ");
printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ----- */
void do_command(command)
char *command;
{
    char message[80];

    strcpy(message, command);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ----- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{

```

```

    char message[80];
    int data_length;

    strcpy(message, command);
    strcat(message, " #8%08d");
    err = viPrintf(vi, message, num_bytes);
    if (err != VI_SUCCESS) error_handler();

    err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();

    return(data_length);
}

/* Query for a string result.
 * ----- */
void do_query_string(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%t", str_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for a number result.
 * ----- */
void do_query_number(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%lf", &num_result);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for numbers result.
 * ----- */
void do_query_numbers(query)

```

```

char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");

    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%,10lf\n", dbl_results);
    if (err != VI_SUCCESS) error_handler();

    check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ----- */
int do_query_ieeeblock(query)
char *query;
{
    char message[80];
    int data_length;

    strcpy(message, query);
    strcat(message, "\n");
    err = viPrintf(vi, message);
    if (err != VI_SUCCESS) error_handler();

    data_length = IEEEBLOCK_SPACE;
    err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
    if (err != VI_SUCCESS) error_handler();

    if (data_length == IEEEBLOCK_SPACE )
    {
        printf("IEEE block buffer full: ");
        printf("May not have received all data.\n");
    }

    check_instrument_errors();

    return(data_length);
}

/* Check for instrument errors.
 * ----- */
void check_instrument_errors()
{
    char str_err_val[256] = {0};
    char str_out[800] = "";

    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
    while(strncmp(str_err_val, "+0,No error", 3) != 0 )
    {
        strcat(str_out, ", ");
        strcat(str_out, str_err_val);
    }
}

```

```

    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
}

if (strcmp(str_out, "") != 0)
{
    printf("INST Error%s\n", str_out);
    err = viFlush(vi, VI_READ_BUF);
    if (err != VI_SUCCESS) error_handler();
    err = viFlush(vi, VI_WRITE_BUF);
    if (err != VI_SUCCESS) error_handler();
}
}

/* Handle VISA errors.
 * ----- */
void error_handler()
{
    char err_msg[1024] = {0};

    viStatusDesc(vi, err, err_msg);
    printf("VISA Error: %s\n", err_msg);
    if (err < VI_SUCCESS)
    {
        exit(1);
    }
}

```

## VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Add the visa32.bas file to your project:
  - a Choose **File>Import File...**
  - b Navigate to the header file, visa32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include), select it, and click **Open**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```

'
' Agilent VISA Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming

```

```

' features of your Agilent oscilloscope.
' -----

Option Explicit

Public err As Long      ' Error returned by VISA function calls.
Public drm As Long      ' Session to Default Resource Manager.
Public vi As Long       ' Session to instrument.

' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -----

Sub Main()

    ' Open the default resource manager session.
    err = viOpenDefaultRM(drm)
    If (err <> VI_SUCCESS) Then HandleVISAError drm

    ' Open the session using the oscilloscope's VISA address.
    err = viOpen(drm, _
        "USB0::0x0957::0x17A6::US50210029::0::INSTR", 0, 15000, vi)
    If (err <> VI_SUCCESS) Then HandleVISAError drm

    ' Set the I/O timeout to ten seconds.
    err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 10000)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    ' Initialize - start from a known state.
    Initialize

    ' Capture data.
    Capture

    ' Analyze the captured waveform.
    Analyze

    ' Close the vi session and the resource manager session.
    err = viClose(vi)
    err = viClose(drm)

```

```

End Sub

'
' Initialize the oscilloscope to a known state.
' -----

Private Sub Initialize()

    ' Clear the interface.
    err = viClear(vi)
    If Not (err = VI_SUCCESS) Then HandleVISAError vi

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

End Sub

'
' Capture the waveform.
' -----

Private Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    ' -----
    DoCommand ":AUToscale"

    ' Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
    DoCommand ":TRIGger:MODE EDGE"
    Debug.Print "Trigger mode: " + _
        DoQueryString(":TRIGger:MODE?")

    ' Set EDGE trigger parameters.
    DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
    Debug.Print "Trigger edge source: " + _
        DoQueryString(":TRIGger:EDGE:SOURce?")

    DoCommand ":TRIGger:EDGE:LEVel 1.5"
    Debug.Print "Trigger edge level: " + _
        DoQueryString(":TRIGger:EDGE:LEVel?")

    DoCommand ":TRIGger:EDGE:SLOPe POSitive"
    Debug.Print "Trigger edge slope: " + _
        DoQueryString(":TRIGger:EDGE:SLOPe?")

    ' Save oscilloscope configuration.
    ' -----
    Dim lngSetupStringSize As Long
    lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
    Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

```



```

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
If Len(Dir(strPath)) Then
    Kill strPath ' Remove file if it exists.
End If

' Open file for output.
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI) ' Write data.
Next lngI
Close hFile ' Close file.

' Change settings with individual commands:
' -----

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALE 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALE?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and position.
DoCommand ":TIMEbase:SCALE 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALE?")

DoCommand ":TIMEbase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSition?")

' Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESOLUTION).
DoCommand ":ACQuire:TYPE NORMAL"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile) ' Length of file.
Get hFile, , byteArray ' Read data.
Close hFile ' Close file.
' Write learn string back to oscilloscope using ":SYSTEM:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTEM:SETup", lngSetupFileSize)
Debug.Print "Setup bytes restored: " + CStr(lngRestored)

```

```

' Capture an acquisition using :DIGitize.
' -----
DoCommand ":DIGitize CHANnel1"

End Sub

'
' Analyze the captured waveform.
' -----

Private Sub Analyze()

' Make a couple of measurements.
' -----
DoCommand ":MEASure:SOURce CHANnel1"
Debug.Print "Measure source: " + _
    DoQueryString(":MEASure:SOURce?")

DoCommand ":MEASure:FREQuency"
dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(dblQueryResult / 1000, 4) + " kHz"

DoCommand ":MEASure:VAMPlitude"
dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
MsgBox "Vertial amplitude:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

' Download the screen image.
' -----
DoCommand ":HARDcopy:INKSaver OFF"

' Get screen image.
Dim lngBlockSize As Long
lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, COlor")
Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

' Save screen image to a file:
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
    Kill strPath ' Remove file if it exists.
End If
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngBlockSize - 1
    Put hFile, , byteArray(lngI) ' Write data.
Next lngI
Close hFile ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----

```

```

' Set the waveform points mode.
DoCommand ":WAVEform:POINTs:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVEform:POINTs:MODE?")

' Get the number of waveform points available.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVEform:POINTs?")

' Set the waveform source.
DoCommand ":WAVEform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVEform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVEform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVEform:FORMat?")

' Display the waveform settings:
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim lngYOrigin As Long
Dim lngYReference As Long
Dim strOutput As String

Dim lngNumNumbers As Long
lngNumNumbers = DoQueryNumbers(":WAVEform:PREamble?")

intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
lngYOrigin = dblArray(8)
lngYReference = dblArray(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
    Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMal"

```

```

ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then
    Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(lngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVEform:DATA?")
Debug.Print "Number of data values: " + CStr(lngNumBytes)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long

For lngI = 0 To lngNumBytes - 1
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngDataValue - lngYReference) _
            * sngYIncrement) + lngYOrigin)
Next lngI

```

```

' Close output file.
Close hFile ' Close file.
MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

End Sub

Private Sub DoCommand(command As String)

    err = viVPrintf(vi, command + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    CheckInstrumentErrors

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

    retCount = lngBlockSize

    Dim strCommandAndLength As String
    strCommandAndLength = command + " %#" + _
        Format(lngBlockSize) + "b"

    err = viVPrintf(vi, strCommandAndLength + vbLf, paramsArray(1))
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    DoCommandIEEEBlock = retCount

    CheckInstrumentErrors

End Function

Private Function DoQueryString(query As String) As String

    Dim strResult As String * 200

    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strResult)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    DoQueryString = strResult

    CheckInstrumentErrors

End Function

Private Function DoQueryNumber(query As String) As Variant

    Dim dblResult As Double

    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

```

```

    err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblResult))
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    DoQueryNumber = dblResult

    CheckInstrumentErrors

End Function

Private Function DoQueryNumbers(query As String) As Long

    Dim dblResult As Double

    ' Send query.
    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    ' Set up paramsArray for multiple parameter query returning array.
    paramsArray(0) = VarPtr(retCount)
    paramsArray(1) = VarPtr(dblArray(0))

    ' Set retCount to max number of elements array can hold.
    retCount = DblArraySize

    ' Read numbers.
    err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    ' retCount is now actual number of values returned by query.
    DoQueryNumbers = retCount

    CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

    ' Send query.
    err = viVPrintf(vi, query + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    ' Set up paramsArray for multiple parameter query returning array.
    paramsArray(0) = VarPtr(retCount)
    paramsArray(1) = VarPtr(byteArray(0))

    ' Set retCount to max number of elements array can hold.
    retCount = ByteArraySize

    ' Get unsigned integer bytes.
    err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_READ_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_WRITE_BUF)

```

```

If (err <> VI_SUCCESS) Then HandleVISAError vi

' retCount is now actual number of bytes returned by query.
DoQueryIEEEBlock_Bytes = retCount

CheckInstrumentErrors

End Function

Private Sub CheckInstrumentErrors()

On Error GoTo ErrorHandler

Dim strErrVal As String * 200
Dim strOut As String

err = viVPrintf(vi, ":SYSTEM:ERROR?" + vbLf, 0) ' Query any errors.
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viVScanf(vi, "%t", strErrVal) ' Read: Errnum,"Error String".
If (err <> VI_SUCCESS) Then HandleVISAError vi

While Val(strErrVal) <> 0 ' End if find: 0,"No Error".
strOut = strOut + "INST Error: " + strErrVal

err = viVPrintf(vi, ":SYSTEM:ERROR?" + vbLf, 0) ' Request error.
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viVScanf(vi, "%t", strErrVal) ' Read error message.
If (err <> VI_SUCCESS) Then HandleVISAError vi

Wend

If Not strOut = "" Then
MsgBox strOut, vbExclamation, "INST Error Messages"

err = viFlush(vi, VI_READ_BUF)
If (err <> VI_SUCCESS) Then HandleVISAError vi

err = viFlush(vi, VI_WRITE_BUF)
If (err <> VI_SUCCESS) Then HandleVISAError vi

End If

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Sub HandleVISAError(session As Long)

Dim strVisaErr As String * 200
Call viStatusDesc(session, err, strVisaErr)

```

```

MsgBox "*** VISA Error : " + strVisaErr, vbExclamation

' If the error is not a warning, close the session.
If err < VI_SUCCESS Then
    If session <> 0 Then Call viClose(session)
    End
End If

End Sub

```

## VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual C#, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add Agilent's VISA header file to your project:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Click **Add** and then click **Add Existing Item...**
  - c Navigate to the header file, visa32.cs (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include directory), select it, but *do not click the Open button*.
  - d Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

- 6 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```

/*
 * Agilent VISA Example in C#
 * -----
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * -----
 */

using System;
using System.IO;
using System.Text;

```



```

namespace InfiniiVision
{
    class VisaInstrumentApp
    {
        private static VisaInstrument myScope;

        public static void Main(string[] args)
        {
            try
            {
                myScope = new
                    VisaInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR");
                myScope.SetTimeoutSeconds(10);

                // Initialize - start from a known state.
                Initialize();

                // Capture data.
                Capture();

                // Analyze the captured waveform.
                Analyze();
            }
            catch (System.ApplicationException err)
            {
                Console.WriteLine("*** VISA Error Message : " + err.Message);
            }
            catch (System.SystemException err)
            {
                Console.WriteLine("*** System Error Message : " + err.Message);
            }
            catch (System.Exception err)
            {
                System.Diagnostics.Debug.Fail("Unexpected Error");
                Console.WriteLine("*** Unexpected Error : " + err.Message);
            }
            finally
            {
                myScope.Close();
            }
        }
    }

    /*
    * Initialize the oscilloscope to a known state.
    * -----
    */
    private static void Initialize()
    {
        StringBuilder strResults;

        // Get and display the device's *IDN? string.
        strResults = myScope.DoQueryString("*IDN?");
        Console.WriteLine("*IDN? result is: {0}", strResults);

        // Clear status and load the default setup.
    }
}

```

```

    myScope.DoCommand("*CLS");
    myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -----
 */
private static void Capture()
{
    // Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale");

    // Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE");
    Console.WriteLine("Trigger mode: {0}",
        myScope.DoQueryString(":TRIGger:MODE?"));

    // Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
    Console.WriteLine("Trigger edge source: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

    myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
    Console.WriteLine("Trigger edge level: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

    myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
    Console.WriteLine("Trigger edge slope: {0}",
        myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

    // Save oscilloscope configuration.
    byte[] ResultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Query and read setup string.
    nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?",
        out ResultsArray);

    // Write setup string to file.
    strPath = "c:\\scope\\config\\setup.stp";
    FileStream fStream = File.Open(strPath, FileMode.Create);
    fStream.Write(ResultsArray, 0, nLength);
    fStream.Close();
    Console.WriteLine("Setup bytes saved: {0}", nLength);

    // Change settings with individual commands:

    // Set vertical scale and offset.
    myScope.DoCommand(":CHANnel1:SCALe 0.05");
    Console.WriteLine("Channel 1 vertical scale: {0}",
        myScope.DoQueryString(":CHANnel1:SCALe?"));

    myScope.DoCommand(":CHANnel1:OFFSet -1.5");
    Console.WriteLine("Channel 1 vertical offset: {0}",
        myScope.DoQueryString(":CHANnel1:OFFSet?"));
}

```

```

// Set horizontal scale and position.
myScope.DoCommand(":TIMEbase:SCALE 0.0002");
Console.WriteLine("Timebase scale: {0}",
    myScope.DoQueryString(":TIMEbase:SCALE?"));

myScope.DoCommand(":TIMEbase:POSition 0.0");
Console.WriteLine("Timebase position: {0}",
    myScope.DoQueryString(":TIMEbase:POSition?"));

// Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESolution
).
myScope.DoCommand(":ACquire:TYPE NORMal");
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACquire:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] dataArray;
int nBytesWritten;

// Read setup string from file.
strPath = "c:\\scope\\config\\setup.stp";
dataArray = File.ReadAllBytes(strPath);

// Restore setup string.
nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTEM:SETup",
    dataArray);
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

// Capture an acquisition using :DIGitize.
myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze the captured waveform.
 * -----
 */
private static void Analyze()
{
    byte[] resultsArray; // Results array.
    int nLength; // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1");
    Console.WriteLine("Measure source: {0}",
        myScope.DoQueryString(":MEASure:SOURce?"));

    double fResult;
    myScope.DoCommand(":MEASure:FREquency");
    fResult = myScope.DoQueryNumber(":MEASure:FREquency?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    myScope.DoCommand(":MEASure:VAMplitude");
    fResult = myScope.DoQueryNumber(":MEASure:VAMplitude?");
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);
}

```

```

// Download the screen image.
// -----
myScope.DoCommand(":HARDcopy:INKSaver OFF");

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COlor",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// -----

// Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINts:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVEform:POINts:MODE?"));

// Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINts 10240");
Console.WriteLine("Waveform points available: {0}",
    myScope.DoQueryString(":WAVEform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVEform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVEform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREAmble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
    Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
    Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
    Console.WriteLine("Waveform format: ASCii");
}

```

```

double fType = fResultsArray[1];
if (fType == 0.0)
{
    Console.WriteLine("Acquire type: NORMAl");
}
else if (fType == 1.0)
{
    Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
    Console.WriteLine("Acquire type: AVERAge");
}
else if (fType == 3.0)
{
    Console.WriteLine("Acquire type: HRESolution");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Waveform X reference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

// Read waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVEform:DATA?",
    out ResultsArray);
Console.WriteLine("Number of data values: {0}", nLength);

// Set up output file:
strPath = "c:\\scope\\data\\waveform_data.csv";
if (File.Exists(strPath)) File.Delete(strPath);

// Open file for output.
StreamWriter writer = File.CreateText(strPath);

// Output waveform data in CSV format.
for (int i = 0; i < nLength - 1; i++)

```

```

        writer.WriteLine("{0:f9}, {1:f6}",
            fXorigin + ((float)i * fXincrement),
            (((float)ResultsArray[i] - fYreference) *
            fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format BYTE data written to {0}",
            strPath);
    }
}

class VisaInstrument
{
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        VisaSendCommandOrQuery(strCommand);

        // Check for inst errors.
        CheckInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
        byte[] dataArray)
    {
        // Send the command to the device.
        string strCommandAndLength;
        int nViStatus, nLength, nBytesWritten;

        nLength = dataArray.Length;
        strCommandAndLength = String.Format("{0} #8%08d",
            strCommand);

        // Write first part of command to formatted I/O write buffer.
        nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,

```

```

        nLength);
    CheckVisaStatus(nViStatus);

    // Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, dataArray, nLength,
        out nBytesWritten);
    CheckVisaStatus(nViStatus);

    // Check for inst errors.
    CheckInstrumentErrors(strCommand);

    return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}

public double DoQueryNumber(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultNumber();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double[] fResultsArray;
    fResultsArray = VisaGetResultNumbers();

    // Check for inst errors.
    CheckInstrumentErrors(strQuery);
}

```

```

        // Return string results.
        return fResultsArray;
    }

    public int DoQueryIEEEBlock(string strQuery,
        out byte[] ResultsArray)
    {
        // Send the query.
        VisaSendCommandOrQuery(strQuery);

        // Get the result string.
        int length; // Number of bytes returned from instrument.
        length = VisaGetResultIEEEBlock(out ResultsArray);

        // Check for inst errors.
        CheckInstrumentErrors(strQuery);

        // Return string results.
        return length;
    }

    private void VisaSendCommandOrQuery(string strCommandOrQuery)
    {
        // Send command or query to the device.
        string strWithNewline;
        strWithNewline = String.Format("{0}\n", strCommandOrQuery);
        int nViStatus;
        nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
        CheckVisaStatus(nViStatus);
    }

    private StringBuilder VisaGetResultString()
    {
        StringBuilder strResults = new StringBuilder(1000);

        // Read return value string from the device.
        int nViStatus;
        nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
        CheckVisaStatus(nViStatus);

        return strResults;
    }

    private double VisaGetResultNumber()
    {
        double fResults = 0;

        // Read return value string from the device.
        int nViStatus;
        nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
        CheckVisaStatus(nViStatus);

        return fResults;
    }

    private double[] VisaGetResultNumbers()

```



```

{
    double[] fResultsArray;
    fResultsArray = new double[10];

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
        fResultsArray);
    CheckVisaStatus(nViStatus);

    return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
    // Results array, big enough to hold a PNG.
    ResultsArray = new byte[300000];
    int length; // Number of bytes returned from instrument.

    // Set the default number of bytes that will be contained in
    // the ResultsArray to 300,000 (300kB).
    length = 300000;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
        ResultsArray);
    CheckVisaStatus(nViStatus);

    // Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
    CheckVisaStatus(nViStatus);

    nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
    CheckVisaStatus(nViStatus);

    return length;
}

private void CheckInstrumentErrors(string strCommand)
{
    // Check for instrument errors.
    StringBuilder strInstrumentError = new StringBuilder(1000);
    bool bFirstError = true;

    do // While not "0,No error"
    {
        VisaSendCommandOrQuery(":SYSTem:ERRor?");
        strInstrumentError = VisaGetString();

        if (!strInstrumentError.ToString().StartsWith("+0, "))
        {
            if (bFirstError)
            {
                Console.WriteLine("ERROR(s) for command '{0}': ",
                    strCommand);
                bFirstError = false;
            }
        }
    }
}

```

```

        }
        Console.WriteLine(strInstrumentError);
    }
} while (!strInstrumentError.ToString().StartsWith("+0, "));
}

private void OpenResourceManager()
{
    int nViStatus;
    nViStatus =
        visa32.viOpenDefaultRM(out this.m_nResourceManager);
    if (nViStatus < visa32.VI_SUCCESS)
        throw new
            ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
    int nViStatus;
    nViStatus = visa32.viOpen(this.m_nResourceManager,
        this.m_strVisaAddress, visa32.VI_NO_LOCK,
        visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
    CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
    int nViStatus;
    nViStatus = visa32.viSetAttribute(this.m_nSession,
        visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
    CheckVisaStatus(nViStatus);
}

public void CheckVisaStatus(int nViStatus)
{
    // If VISA error, throw exception.
    if (nViStatus < visa32.VI_SUCCESS)
    {
        StringBuilder strError = new StringBuilder(256);
        visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
            strError);
        throw new ApplicationException(strError.ToString());
    }
}

public void Close()
{
    if (m_nSession != 0)
        visa32.viClose(m_nSession);
    if (m_nResourceManager != 0)
        visa32.viClose(m_nResourceManager);
}
}
}

```

## VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual Basic, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the Visual Basic .NET source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add Agilent's VISA header file to your project:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add** and then choose **Add Existing Item...**
  - c Navigate to the header file, visa32.vb (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include directory), select it, but *do not click the Open button*.
  - d Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

  - e Right-click the project again and choose **Properties**; then, select "InfiniiVision.VisaInstrumentApp" as the **Startup object**.
- 6 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA Example in Visual Basic .NET
' -----
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' -----

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
```

```

        New VisaInstrument("USB0::0x0957::0x17A6::US50210029::0::INSTR
")
myScope.SetTimeoutSeconds(10)

' Initialize - start from a known state.
Initialize()

' Capture data.
Capture()

' Analyze the captured waveform.
Analyze()

Catch err As System.ApplicationException
    Console.WriteLine("*** VISA Error Message : " + err.Message)
Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
Catch err As System.Exception
    Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
End Try
End Sub

'
' Initialize the oscilloscope to a known state.
' -----

Private Shared Sub Initialize()
    Dim strResults As StringBuilder

    ' Get and display the device's *IDN? string.
    strResults = myScope.DoQueryString("*IDN?")
    Console.WriteLine("*IDN? result is: {0}", strResults)

    ' Clear status and load the default setup.
    myScope.DoCommand("*CLS")
    myScope.DoCommand("*RST")

End Sub

'
' Capture the waveform.
' -----

Private Shared Sub Capture()

    ' Use auto-scale to automatically configure oscilloscope.
    myScope.DoCommand(":AUToscale")

    ' Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
    myScope.DoCommand(":TRIGger:MODE EDGE")
    Console.WriteLine("Trigger mode: {0}", _
        myScope.DoQueryString(":TRIGger:MODE?"))

    ' Set EDGE trigger parameters.
    myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
    Console.WriteLine("Trigger edge source: {0}", _

```

```

        myScope.DoQueryString(":TRIGger:EDGE:SOURce?")

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte() ' Results array.
Dim nLength As Integer ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?", _
    ResultsArray)

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and position.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMebase:SCALe?"))

myScope.DoCommand(":TIMebase:POSition 0.0")
Console.WriteLine("Timebase position: {0}", _
    myScope.DoQueryString(":TIMebase:POSition?"))

' Set the acquisition type (NORMal, PEAK, AVERAge, or HRESolution)

myScope.DoCommand(":ACQuire:TYPE NORMal")
Console.WriteLine("Acquire type: {0}", _
    myScope.DoQueryString(":ACQuire:TYPE?"))

' Or, configure by loading a previously saved setup.
Dim dataArray As Byte()
Dim nBytesWritten As Integer

```

```

' Read setup string from file.
strPath = "c:\scope\config\setup.stp"
dataArray = File.ReadAllBytes(strPath)

' Restore setup string.
nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTEM:SETup", _
    dataArray)
Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

' Capture an acquisition using :DIGitize.
myScope.DoCommand(":DIGitize CHANnel1")

End Sub

'
' Analyze the captured waveform.
' -----

Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte() ' Results array.
    Dim nLength As Integer ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' -----
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
        myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    myScope.DoCommand(":MEASure:VAMPlitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
    Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

    ' Download the screen image.
    ' -----
    myScope.DoCommand(":HARDcopy:INKSaver OFF")

    ' Get the screen data.
    nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, COLor", _
        ResultsArray)

    ' Store the screen data to a file.
    strPath = "c:\scope\data\screen.png"
    Dim fStream As FileStream
    fStream = File.Open(strPath, FileMode.Create)
    fStream.Write(ResultsArray, 0, nLength)
    fStream.Close()
    Console.WriteLine("Screen image ({0} bytes) written to {1}", _
        nLength, strPath)

    ' Download waveform data.

```

```

' -----

' Set the waveform points mode.
myScope.DoCommand(":WAVEform:POINts:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVEform:POINts:MODE?"))

' Get the number of waveform points available.
myScope.DoCommand(":WAVEform:POINts 10240")
Console.WriteLine("Waveform points available: {0}", _
    myScope.DoQueryString(":WAVEform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVEform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVEform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVEform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVEform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
    Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
    Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
    Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
    Console.WriteLine("Acquire type: NORMAl")
ElseIf fType = 1 Then
    Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
    Console.WriteLine("Acquire type: AVERAge")
ElseIf fType = 3 Then
    Console.WriteLine("Acquire type: HRESolution")
End If

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Waveform points: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Waveform average count: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

```

```

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Waveform X reference: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

' Get the waveform data.
nLength = myScope.DoQueryIEEEBlock(":WAVEform:DATA?", _
    ResultsArray)
Console.WriteLine("Number of data values: {0}", nLength)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"
If File.Exists(strPath) Then
    File.Delete(strPath)
End If

' Open file for output.
Dim writer As StreamWriter = File.CreateText(strPath)

' Output waveform data in CSV format.
For index As Integer = 0 To nLength - 1
    ' Write time value, voltage value.
    writer.WriteLine("{0:f9}, {1:f6}", _
        fXorigin + (CSng(index) * fXincrement), _
        ((CSng(ResultsArray(index)) - fYreference) _
        * fYincrement) + fYorigin)
Next

' Close output file.
writer.Close()
Console.WriteLine("Waveform format BYTE data written to {0}", _
    strPath)

End Sub

End Class

Class VisaInstrument
Private m_nResourceManager As Integer
Private m_nSession As Integer
Private m_strVisaAddress As String

' Constructor.
Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA resource manager.
    OpenResourceManager()

```



```

    ' Open a VISA resource session.
    OpenSession()

    ' Clear the interface.
    Dim nViStatus As Integer
    nViStatus = visa32.viClear(m_nSession)
End Sub

Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    VisaSendCommandOrQuery(strCommand)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal dataArray As Byte()) As Integer

    ' Send the command to the device.
    Dim strCommandAndLength As String
    Dim nViStatus As Integer
    Dim nLength As Integer
    Dim nBytesWritten As Integer

    nLength = dataArray.Length
    strCommandAndLength = [String].Format("{0} #8{1:D8}", _
        strCommand, nLength)

    ' Write first part of command to formatted I/O write buffer.
    nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
    CheckVisaStatus(nViStatus)

    ' Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, dataArray, nLength, _
        nBytesWritten)
    CheckVisaStatus(nViStatus)

    ' Check for inst errors.
    CheckInstrumentErrors(strCommand)

    Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
    As StringBuilder
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim strResults As New StringBuilder(1000)
    strResults = VisaGetResultString()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

```

```

    ' Return string results.
    Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResults As Double
    fResults = VisaGetResultNumber()

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResultsArray As Double()
    fResultsArray = VisaGetResultNumbers()

    ' Check for instrument errors (another command and result).
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
    ByRef ResultsArray As Byte()) As Integer
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    System.Threading.Thread.Sleep(2000) ' Delay before reading data.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(ResultsArray)

    ' Check for inst errors.
    CheckInstrumentErrors(strQuery)

    ' Return string results.
    Return length
End Function

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
    As String)
    ' Send command or query to the device.

```

```

    Dim strWithNewline As String
    strWithNewline = [String].Format("{0}" & Chr(10) & "", _
        strCommandOrQuery)
    Dim nViStatus As Integer
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
    CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
    Dim strResults As New StringBuilder(1000)

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
    CheckVisaStatus(nViStatus)

    Return strResults
End Function

Private Function VisaGetResultNumber() As Double
    Dim fResults As Double = 0

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
    CheckVisaStatus(nViStatus)

    Return fResults
End Function

Private Function VisaGetResultNumbers() As Double()
    Dim fResultsArray As Double()
    fResultsArray = New Double(9) {}

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, _
        "%,10lf" & Chr(10) & "", fResultsArray)
    CheckVisaStatus(nViStatus)

    Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
    As Byte()) As Integer
    ' Results array, big enough to hold a PNG.
    ResultsArray = New Byte(299999) {}
    Dim length As Integer
    ' Number of bytes returned from instrument.
    ' Set the default number of bytes that will be contained in
    ' the ResultsArray to 300,000 (300kB).
    length = 300000

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
        ResultsArray)

```

```

CheckVisaStatus(nViStatus)

' Write and read buffers need to be flushed after IEEE block?
nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
CheckVisaStatus(nViStatus)

nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
CheckVisaStatus(nViStatus)

Return length
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
' Check for instrument errors.
Dim strInstrumentError As New StringBuilder(1000)
Dim bFirstError As Boolean = True
Do ' While not "0,No error"
    VisaSendCommandOrQuery(":SYSTem:ERRor?")
    strInstrumentError = VisaGetString()

    If Not strInstrumentError.ToString().StartsWith("+0,") Then
        If bFirstError Then
            Console.WriteLine("ERROR(s) for command '{0}': ", _
                strCommand)
            bFirstError = False
        End If
        Console.Write(strInstrumentError)
    End If
Loop While Not strInstrumentError.ToString().StartsWith("+0,")
End Sub

Private Sub OpenResourceManager()
Dim nViStatus As Integer
nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
If nViStatus < visa32.VI_SUCCESS Then
    Throw New _
        ApplicationException("Failed to open Resource Manager")
End If
End Sub

Private Sub OpenSession()
Dim nViStatus As Integer
nViStatus = visa32.viOpen(Me.m_nResourceManager, _
    Me.m_strVisaAddress, visa32.VI_NO_LOCK, _
    visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
Dim nViStatus As Integer
nViStatus = visa32.viSetAttribute(Me.m_nSession, _
    visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
CheckVisaStatus(nViStatus)
End Sub

Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
' If VISA error, throw exception.

```

```
    If nViStatus < visa32.VI_SUCCESS Then
        Dim strError As New StringBuilder(256)
        visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
        Throw New ApplicationException(strError.ToString())
    End If
End Sub

Public Sub Close()
    If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
    End If
    If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
    End If
End Sub
End Class
End Namespace
```

## SICL Examples

- "SICL Example in C" on page 686
- "SICL Example in Visual Basic" on page 695

### SICL Example in C

To compile and run this example in Microsoft Visual Studio 2008:

- 1 Open Visual Studio.
- 2 Create a new Visual C++, Win32, Win32 Console Application project.
- 3 In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
- 4 Cut-and-paste the code that follows into a file named "example.c" in the project directory.
- 5 In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
- 6 Edit the program to use the SICL address of your oscilloscope.
- 7 Choose **Project > Properties....** In the Property Pages dialog, update these project settings:
  - a Under Configuration Properties, Linker, Input, add "sicl32.lib" to the Additional Dependencies field.
  - b Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
  - c Click **OK** to close the Property Pages dialog.
- 8 Add the include files and library files search paths:
  - a Choose **Tools > Options....**
  - b In the Options dialog, select **VC++ Directories** under Projects and Solutions.
  - c Show directories for **Include files**, and add the include directory (for example, Program Files\Agilent\IO Libraries Suite\include).
  - d Show directories for **Library files**, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
  - e Click **OK** to close the Options dialog.
- 9 Build and run the program.

```

/*
 * Agilent SICL Example in C
 * -----
 * This program illustrates a few commonly-used programming
 * features of your Agilent oscilloscope.
 */

```

```

#include <stdio.h>           /* For printf(). */
#include <string.h>         /* For strcpy(), strcat(). */
#include <time.h>           /* For clock(). */
#include <sicl.h>           /* Agilent SICL routines. */

#define SICL_ADDRESS       "usb0[2391::6054::US50210029::0]"
#define TIMEOUT            5000
#define IEEEBLOCK_SPACE   100000

/* Function prototypes */
void initialize(void);      /* Initialize to known state. */
void capture(void);        /* Capture the waveform. */
void analyze(void);        /* Analyze the captured waveform. */

void do_command(char *command); /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query); /* Query for string. */
void do_query_number(char *query); /* Query for number. */
void do_query_numbers(char *query); /* Query for numbers. */
int do_query_ieeeblock(char *query); /* Query for IEEE block. */
void check_instrument_errors(); /* Check for inst errors. */

/* Global variables */
INST id; /* Device session ID. */
char str_result[256] = {0}; /* Result from do_query_string(). */
double num_result; /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE]; /* Result from
do_query_ieeeblock(). */
double dbl_results[10]; /* Result from do_query_numbers(). */

/* Main Program
 * ----- */
void main(void)
{
    /* Install a default SICL error handler that logs an error message
     * and exits. On Windows 98SE or Windows Me, view messages with
     * the SICL Message Viewer. For Windows 2000 or XP, use the Event
     * Viewer.
     */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the SICL_ADDRESS */
    id = iopen(SICL_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf ("Oscilloscope session opened!\n");
    }

    /* Initialize - start from a known state. */
    initialize();
}

```

```

    /* Capture data. */
    capture();

    /* Analyze the captured waveform. */
    analyze();

    /* Close the device session to the instrument. */
    iclose(id);
    printf ("Program execution is complete...\n");

    /* For WIN16 programs, call _siclcleanup before exiting to release
     * resources allocated by SICL for this application. This call is
     * a no-op for WIN32 programs.
     */
    _siclcleanup();
}

/* Initialize the oscilloscope to a known state.
 * ----- */
void initialize (void)
{
    /* Set the I/O timeout value for this session to 5 seconds. */
    itimeout(id, TIMEOUT);

    /* Clear the interface. */
    iclear(id);

    /* Get and display the device's *IDN? string. */
    do_query_string("*IDN?");
    printf("Oscilloscope *IDN? string: %s\n", str_result);

    /* Clear status and load the default setup. */
    do_command("*CLS");
    do_command("*RST");
}

/* Capture the waveform.
 * ----- */
void capture (void)
{
    int num_bytes;
    FILE *fp;

    /* Use auto-scale to automatically configure oscilloscope.
     * ----- */
    do_command(":AUToscale");

    /* Set trigger mode (EDGE, PULSe, PATtern, etc., and input source. */
    do_command(":TRIGger:MODE EDGE");
    do_query_string(":TRIGger:MODE?");
    printf("Trigger mode: %s\n", str_result);

    /* Set EDGE trigger parameters. */
    do_command(":TRIGger:EDGE:SOURCe CHANnel1");
    do_query_string(":TRIGger:EDGE:SOURce?");
    printf("Trigger edge source: %s\n", str_result);
}

```



```

do_command(":TRIGger:EDGE:LEVel 1.5");
do_query_string(":TRIGger:EDGE:LEVel?");
printf("Trigger edge level: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope configuration.
 * ----- */

/* Read system setup. */
num_bytes = do_query_ieeeblock(":SYSTem:SETUp?");
printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:
 * ----- */

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and position. */
do_command(":TIMEbase:SCALe 0.0002");
do_query_string(":TIMEbase:SCALe?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMEbase:POSition 0.0");
do_query_string(":TIMEbase:POSition?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition type (NORMAL, PEAK, AVERage, or HRESolution). *
/
do_command(":ACQuire:TYPE NORMal");
do_query_string(":ACQuire:TYPE?");
printf("Acquire type: %s\n", str_result);

/* Or, configure by loading a previously saved setup.
 * ----- */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);

```

```

fclose (fp);
printf("Read setup string (%d bytes) from file ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
printf("Restored setup string (%d bytes).\n", num_bytes);

/* Capture an acquisition using :DIGitize.
 * ----- */
do_command(":DIGitize CHANnel1");
}

/* Analyze the captured waveform.
 * ----- */
void analyze (void)
{
    double wav_format;
    double acq_type;
    double wav_points;
    double avg_count;
    double x_increment;
    double x_origin;
    double x_reference;
    double y_increment;
    double y_origin;
    double y_reference;

    FILE *fp;
    int num_bytes; /* Number of bytes returned from instrument. */
    int i;

    /* Make a couple of measurements.
     * ----- */
    do_command(":MEASure:SOURce CHANnel1");
    do_query_string(":MEASure:SOURce?");
    printf("Measure source: %s\n", str_result);

    do_command(":MEASure:FREQuency");
    do_query_number(":MEASure:FREQuency?");
    printf("Frequency: %.4f kHz\n", num_result / 1000);

    do_command(":MEASure:VAMplitude");
    do_query_number(":MEASure:VAMplitude?");
    printf("Vertical amplitude: %.2f V\n", num_result);

    /* Download the screen image.
     * ----- */
    do_command(":HARDcopy:INKSaver OFF");

    /* Read screen image. */
    num_bytes = do_query_ieeeblock(":DISPlay:DATA? PNG, COLor");
    printf("Screen image bytes: %d\n", num_bytes);

    /* Write screen image bytes to file. */
    fp = fopen ("c:\\scope\\data\\screen.png", "wb");
    num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,

```

```

    fp);
fclose (fp);
printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * ----- */

/* Set the waveform points mode. */
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Get the number of waveform points available. */
do_command(":WAVeform:POINts 10240");
do_query_string(":WAVeform:POINts?");
printf("Waveform points available: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
    printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
    printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
    printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
    printf("Acquire type: NORMal\n");
}
else if (acq_type == 1.0)
{
    printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
    printf("Acquire type: AVERage\n");
}

```

```

    }
    else if (acq_type == 3.0)
    {
        printf("Acquire type: HRESolution\n");
    }

    wav_points = dbl_results[2];
    printf("Waveform points: %e\n", wav_points);

    avg_count = dbl_results[3];
    printf("Waveform average count: %e\n", avg_count);

    x_increment = dbl_results[4];
    printf("Waveform X increment: %e\n", x_increment);

    x_origin = dbl_results[5];
    printf("Waveform X origin: %e\n", x_origin);

    x_reference = dbl_results[6];
    printf("Waveform X reference: %e\n", x_reference);

    y_increment = dbl_results[7];
    printf("Waveform Y increment: %e\n", y_increment);

    y_origin = dbl_results[8];
    printf("Waveform Y origin: %e\n", y_origin);

    y_reference = dbl_results[9];
    printf("Waveform Y reference: %e\n", y_reference);

    /* Read waveform data. */
    num_bytes = do_query_ieeeblock(":WAVEform:DATA?");
    printf("Number of data values: %d\n", num_bytes);

    /* Open file for output. */
    fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

    /* Output waveform data in CSV format. */
    for (i = 0; i < num_bytes - 1; i++)
    {
        /* Write time value, voltage value. */
        fprintf(fp, "%9f, %6f\n",
            x_origin + ((float)i * x_increment),
            (((float)ieeeblock_data[i] - y_reference) * y_increment)
            + y_origin);
    }

    /* Close output file. */
    fclose(fp);
    printf("Waveform format BYTE data written to ");
    printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ----- */
void do_command(command)
char *command;

```

```

{
    char message[80];

    strcpy(message, command);
    strcat(message, "\n");
    iprintf(id, message);

    check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ----- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
    char message[80];
    int data_length;

    strcpy(message, command);
    strcat(message, " #8%08d");
    iprintf(id, message, num_bytes);
    ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

    check_instrument_errors();

    return(data_length);
}

/* Query for a string result.
 * ----- */
void do_query_string(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%t\n", str_result);

    check_instrument_errors();
}

/* Query for a number result.
 * ----- */
void do_query_number(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    iprintf(id, message);

    iscanf(id, "%lf", &num_result);
}

```

```

    check_instrument_errors();
}

/* Query for numbers result.
 * ----- */
void do_query_numbers(query)
char *query;
{
    char message[80];

    strcpy(message, query);
    strcat(message, "\n");
    fprintf(id, message);

    fscanf(id, "%,10lf\n", dbl_results);

    check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ----- */
int do_query_ieeeblock(query)
char *query;
{
    char message[80];
    int data_length;

    strcpy(message, query);
    strcat(message, "\n");
    fprintf(id, message);

    data_length = IEEEBLOCK_SPACE;
    fscanf(id, "%#b", &data_length, ieeeblock_data);

    if (data_length == IEEEBLOCK_SPACE )
    {
        printf("IEEE block buffer full: ");
        printf("May not have received all data.\n");
    }

    check_instrument_errors();

    return(data_length);
}

/* Check for instrument errors.
 * ----- */
void check_instrument_errors()
{
    char str_err_val[256] = {0};
    char str_out[800] = "";

    fprintf(id, ":SYSTEM:ERROR?\n", "%t", str_err_val);
    while(strncmp(str_err_val, "+0,No error", 3) != 0 )
    {
        strcat(str_out, ", ");
    }
}

```

```

    strcat(str_out, str_err_val);
    ipromptf(id, ":SYSTEM:ERROR?\n", "%t", str_err_val);
}

if (strcmp(str_out, "") != 0)
{
    printf("INST Error%s\n", str_out);
    iflush(id, I_BUF_READ | I_BUF_WRITE);
}
}

```

## SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Add the `sicl32.bas` file to your project:
  - a Choose **File>Import File...**
  - b Navigate to the header file, `sicl32.bas` (installed with Agilent IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the SICL address of your oscilloscope, and save the changes.
- 7 Run the program.

```

'
' Agilent SICL Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' -----

Option Explicit

Public id As Integer    ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

```

```

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' -----

Sub Main()

    On Error GoTo ErrorHandler

    ' Open a device session using the SICL_ADDRESS.
    id = iopen("usb0[2391::6054::US50210029::0]")
    Call itimeout(id, 5000)

    ' Initialize - start from a known state.
    Initialize

    ' Capture data.
    Capture

    ' Analyze the captured waveform.
    Analyze

    ' Close the vi session and the resource manager session.
    Call iclose(id)

    Exit Sub

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Sub

'
' Initialize the oscilloscope to a known state.
' -----

Private Sub Initialize()

    On Error GoTo ErrorHandler

    ' Clear the interface.
    Call iclear(id)

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

    Exit Sub

```



```

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Sub

'
' Capture the waveform.
' -----

Private Sub Capture()

    On Error GoTo ErrorHandler

    ' Use auto-scale to automatically configure oscilloscope.
    ' -----
    DoCommand ":AUToscale"

    ' Set trigger mode (EDGE, PULSe, PATtern, etc., and input source.
    DoCommand ":TRIGger:MODE EDGE"
    Debug.Print "Trigger mode: " + _
        DoQueryString(":TRIGger:MODE?")

    ' Set EDGE trigger parameters.
    DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
    Debug.Print "Trigger edge source: " + _
        DoQueryString(":TRIGger:EDGE:SOURce?")

    DoCommand ":TRIGger:EDGE:LEVel 1.5"
    Debug.Print "Trigger edge level: " + _
        DoQueryString(":TRIGger:EDGE:LEVel?")

    DoCommand ":TRIGger:EDGE:SLOPe POSitive"
    Debug.Print "Trigger edge slope: " + _
        DoQueryString(":TRIGger:EDGE:SLOPe?")

    ' Save oscilloscope configuration.
    ' -----
    Dim lngSetupStringSize As Long
    lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
    Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

    ' Output setup string to a file:
    Dim strPath As String
    strPath = "c:\scope\config\setup.dat"
    If Len(Dir(strPath)) Then
        Kill strPath ' Remove file if it exists.
    End If

    ' Open file for output.
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    For lngI = 0 To lngSetupStringSize - 1

```

```

    Put hFile, , byteArray(lngI) ' Write data.
Next lngI
Close hFile ' Close file.

' Change settings with individual commands:
' -----

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALE 0.05"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALE?")

DoCommand ":CHANnel1:OFFSet -1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and position.
DoCommand ":TIMEbase:SCALE 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALE?")

DoCommand ":TIMEbase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSition?")

' Set the acquisition type (NORMAL, PEAK, AVERAGE, or HRESOLUTION).
DoCommand ":ACQuire:TYPE NORMAL"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile) ' Length of file.
Get hFile, , byteArray ' Read data.
Close hFile ' Close file.
' Write setup string back to oscilloscope using ":SYSTEM:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTEM:SETup", lngSetupFileSize)
Debug.Print "Setup bytes restored: " + CStr(lngRestored)

' Capture an acquisition using :DIGitize.
' -----
DoCommand ":DIGitize CHANnel1"

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

```

```

'
' Analyze the captured waveform.
' -----

Private Sub Analyze()

    On Error GoTo ErrorHandler

    ' Make a couple of measurements.
    ' -----
    DoCommand ":MEASure:SOURce CHANnel1"
    Debug.Print "Measure source: " + _
        DoQueryString(":MEASure:SOURce?")

    DoCommand ":MEASure:FREQuency"
    dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
    MsgBox "Frequency:" + vbCrLf + _
        FormatNumber(dblQueryResult / 1000, 4) + " kHz"

    DoCommand ":MEASure:VAMPlitude"
    dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
    MsgBox "Vertial amplitude:" + vbCrLf + _
        FormatNumber(dblQueryResult, 4) + " V"

    ' Download the screen image.
    ' -----
    DoCommand ":HARDcopy:INKSaver OFF"

    ' Get screen image.
    Dim lngBlockSize As Long
    lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, COLor")
    Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

    ' Save screen image to a file:
    Dim strPath As String
    strPath = "c:\scope\data\screen.png"
    If Len(Dir(strPath)) Then
        Kill strPath ' Remove file if it exists.
    End If
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    ' Skip past header.
    For lngI = CInt(Chr(byteArray(1))) + 2 To lngBlockSize - 1
        Put hFile, , byteArray(lngI) ' Write data.
    Next lngI
    Close hFile ' Close file.
    MsgBox "Screen image written to " + strPath

    ' Download waveform data.
    ' -----

    ' Set the waveform points mode.
    DoCommand ":WAVEform:POINts:MODE RAW"
    Debug.Print "Waveform points mode: " + _
        DoQueryString(":WAVEform:POINts:MODE?")

```

```

' Get the number of waveform points available.
DoCommand ":WAVEform:POINTs 10240"
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVEform:POINTs?")

' Set the waveform source.
DoCommand ":WAVEform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVEform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVEform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVEform:FORMat?")

' Display the waveform settings:
Dim Preamble() As Double
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVEform:PREAmble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
    Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
    Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
    Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
    Debug.Print "Acquisition type: NORMal"
ElseIf intType = 1 Then
    Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
    Debug.Print "Acquisition type: AVERage"
ElseIf intType = 3 Then

```

```

    Debug.Print "Acquisition type: HRESolution"
End If

Debug.Print "Waveform points: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVEform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(lngNumBytes - CInt(Chr(byteArray(1))) - 2)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long

' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngNumBytes - 2
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngDataValue - lngYReference) * _
            sngYIncrement) + sngYOrigin)

Next lngI

' Close output file.
Close hFile ' Close file.

```

## 34 Programming Examples

```
MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Sub DoCommand(command As String)

On Error GoTo ErrorHandler

Call ivprintf(id, command + vbLf)

CheckInstrumentErrors

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

On Error GoTo ErrorHandler

' Send command part.
Call ivprintf(id, command + " ")

' Write definite-length block bytes.
Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)

' retCount is now actual number of bytes written.
DoCommandIEEEBlock = retCount

CheckInstrumentErrors

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryString(query As String) As String

Dim actual As Long
```

```

On Error GoTo ErrorHandler

Dim strResult As String * 200

Call ivprintf(id, query + vbLf)
Call ivscanf(id, "%200t", strResult)
DoQueryString = strResult

CheckInstrumentErrors

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryNumber(query As String) As Double

On Error GoTo ErrorHandler

Dim dblResult As Double

Call ivprintf(id, query + vbLf)
Call ivscanf(id, "%lf" + vbLf, dblResult)
DoQueryNumber = dblResult

CheckInstrumentErrors

Exit Function

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryNumbers(query As String) As Double()

On Error GoTo ErrorHandler

Dim dblResults(10) As Double

Call ivprintf(id, query + vbLf)
Call ivscanf(id, "%,10lf" + vbLf, dblResults)
DoQueryNumbers = dblResults

CheckInstrumentErrors

Exit Function

ErrorHandler:

```

```

    MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

    On Error GoTo ErrorHandler

    ' Send query.
    Call ivprintf(id, query + vbCrLf)

    ' Read definite-length block bytes.
    Sleep 2000 ' Delay before reading data.
    Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)

    ' Get number of block length digits.
    Dim intLengthDigits As Integer
    intLengthDigits = CInt(Chr(byteArray(1)))

    ' Get block length from those digits.
    Dim strBlockLength As String
    strBlockLength = ""
    Dim i As Integer
    For i = 2 To intLengthDigits + 1
        strBlockLength = strBlockLength + Chr(byteArray(i))
    Next

    ' Return number of bytes in block plus header.
    DoQueryIEEEBlock_Bytes = CLng(strBlockLength) + intLengthDigits + 2

    CheckInstrumentErrors

    Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Sub CheckInstrumentErrors()

    On Error GoTo ErrorHandler

    Dim strErrVal As String * 200
    Dim strOut As String

    Call ivprintf(id, ":SYSTem:ERRor?" + vbCrLf) ' Query any errors data.
    Call ivscanf(id, "%200t", strErrVal) ' Read: Errnum,"Error String".
    While Val(strErrVal) <> 0 ' End if find: +0,"No Error".
        strOut = strOut + "INST Error: " + strErrVal
        Call ivprintf(id, ":SYSTem:ERRor?" + vbCrLf) ' Request error message
    .
    Call ivscanf(id, "%200t", strErrVal) ' Read error message.
Wend

```



```
If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    Call iflush(id, I_BUF_READ Or I_BUF_WRITE)

End If

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub
```



# Index

## Symbols

+9.9E+37, infinity representation, 615  
+9.9E+37, measurement error, 295

## Numerics

0 (zero) values in waveform data, 469  
1 (one) values in waveform data, 469  
7000B Series oscilloscopes, command differences from, 24  
82350A GPIB interface, 6

## A

AC coupling, trigger edge, 434  
AC input coupling for specified channel, 195  
AC RMS measured on waveform, 327  
accumulate activity, 125  
ACQuire commands, 157  
acquire data, 133, 169  
acquire mode on autoscale, 129  
acquire reset conditions, 109, 403  
acquire sample rate, 168  
ACQuire subsystem, 43  
acquired data points, 162  
acquisition count, 160  
acquisition mode, 157, 161, 486  
acquisition type, 157, 169  
acquisition types, 461  
active edges, 125  
active printer, 260  
activity logic levels, 125  
activity on digital channels, 125  
add function, 481  
add math function, 248  
add math function as g(t) source, 244  
address of network printer, 265  
Addresses softkey, 30  
AER (Arm Event Register), 126, 141, 143, 592  
Agilent Connection Expert, 31  
Agilent Interactive IO application, 35  
Agilent IO Control icon, 31  
Agilent IO Libraries Suite, 5, 27, 40, 42  
Agilent IO Libraries Suite, installing, 28  
ALB waveform data format, 392  
all (snapshot) measurement, 296  
ALL segments waveform save option, 394  
AM demo signal, 212  
amplitude, vertical, 321  
amplitude, waveform generator, 507  
analog channel coupling, 195  
analog channel display, 196

analog channel impedance, 197  
analog channel input, 530  
analog channel inversion, 198  
analog channel labels, 199, 230  
analog channel offset, 200  
analog channel protection lock, 406  
analog channel range, 207  
analog channel scale, 208  
analog channel source for glitch, 447  
analog channel units, 209  
analog channels only oscilloscopes, 5  
analog probe attenuation, 201  
analog probe head type, 202  
analog probe sensing, 531  
analog probe skew, 204, 529  
analyzing captured data, 39  
angle brackets, 92  
annotate channels, 199  
apply network printer connection settings, 266  
area for hardcopy print, 259  
area for saved image, 565  
Arm Event Register (AER), 126, 141, 143, 592  
arrange waveforms, 533  
ASCII format, 471  
ASCII format for data transfer, 465  
ASCII string, quoted, 92  
ASCIixy waveform data format, 392  
assign channel names, 199  
attenuation factor (external trigger) probe, 235  
attenuation for oscilloscope probe, 201  
AUT option for probe sense, 531, 535  
auto trigger sweep mode, 423  
automask create, 337  
automask source, 338  
automask units, 339  
automatic measurements constants, 201  
automatic probe type detection, 531, 535  
autoscale, 127  
autoscale acquire mode, 129  
autoscale channels, 130  
AUToscale command, 42  
average value measurement, 322  
averaging acquisition type, 158, 463  
averaging, synchronizing with, 604

## B

bandwidth filter limits, 234  
bandwidth filter limits to 20 MHz, 194  
base value measurement, 323  
basic instrument functions, 97  
begin acquisition, 133, 150, 152

BHARris window for minimal spectral leakage, 255  
binary block data, 92, 228, 407, 469  
BINary waveform data format, 392  
bind levels for masks, 358  
bit selection command, bus, 173  
bit weights, 102  
bitmap display, 228  
bits in Service Request Enable Register, 114  
bits in Standard Event Status Enable Register, 100  
bits in Status Byte Register, 116  
bits selection command, bus, 174  
blank, 132  
block data, 92, 105, 407  
block response data, 46  
blocking synchronization, 599  
blocking wait, 598  
BMP format screen image data, 228  
braces, 91  
built-in measurements, 39  
burst data demo signal, 212  
bus bit selection command, 173  
bus bits selection commands, 174  
bus clear command, 176  
bus commands, 172  
BUS data format, 466  
bus display, 177  
bus label command, 178  
bus mask command, 179  
BUS<n> commands, 171  
button disable, 402  
button, calibration protect, 186  
byte format for data transfer, 465, 471  
BYTeorder, 467

## C

C, SICL library example, 686  
C, VISA library example, 645  
C#, VISA COM example, 627  
C#, VISA example, 664  
CAL PROTECT button, 186  
CAL PROTECT switch, 181  
calculating preshoot of waveform, 311  
calculating the waveform overshoot, 307  
calibrate, 183, 184, 186, 190  
CALibrate commands, 181  
calibrate date, 183  
calibrate introduction, 181  
calibrate label, 184  
calibrate output, 185  
calibrate start, 187

## Index

calibrate status, 188  
calibrate switch, 186  
calibrate temperature, 189  
calibrate time, 190  
capture data, 133  
capturing data, 38  
center frequency set, 240, 242  
center of screen, 494  
center reference, 416  
center screen, vertical value at, 247, 250  
channel, 156, 199, 526, 528  
channel coupling, 195  
channel display, 196  
channel input impedance, 197  
channel inversion, 198  
channel label, 199, 527  
channel labels, 229, 230  
channel numbers, 533  
channel overload, 206  
channel protection, 206  
channel reset conditions, 109, 403  
channel selected to produce trigger, 447, 457  
channel signal type, 205  
channel skew for oscilloscope probe, 204, 529  
channel status, 153, 533  
channel threshold, 528  
channel vernier, 210  
channel, stop displaying, 132  
CHANnel<n> commands, 191, 193  
channels to autoscale, 130  
channels, how autoscale affects, 127  
characters to display, 400  
classes of input signals, 255  
classifications, command, 608  
clear, 227  
clear bus command, 176  
clear cumulative edge variables, 526  
clear markers, 297, 544  
clear measurement, 297, 544  
clear message queue, 99  
Clear method, 41  
clear reference waveforms, 513  
clear screen, 534  
clear status, 99  
clear waveform area, 225  
clipped high waveform data value, 469  
clipped low waveform data value, 469  
clock with infrequent glitch demo signal, 212  
CLS (Clear Status), 99  
CME (Command Error) status bit, 100, 102  
CMOS threshold voltage for digital channels, 223, 528  
CMOS trigger threshold voltage, 567  
code, :ACQUIRE:COMPLETE, 159  
code, :ACQUIRE:SEGMENTED, 165  
code, :ACQUIRE:TYPE, 170  
code, :AUTOSCALE, 128  
code, :CHANNEL<n>:LABEL, 199  
code, :CHANNEL<n>:PROBE, 201  
code, :CHANNEL<n>:RANGE, 207  
code, :DIGITIZE, 134  
code, :DISPLAY:DATA, 228

code, :DISPLAY:LABEL, 229  
code, :DISPLAY:ORDER, 533  
code, :MEASURE:PERIOD, 316  
code, :MEASURE:TEDGE, 318  
code, :MTEST, 333  
code, :POD<n>:THRESHOLD, 368  
code, :RUN/:STOP, 150  
code, :SYSTEM:SETUP, 407  
code, :TIMEBASE:DELAY, 566  
code, :TIMEBASE:MODE, 413  
code, :TIMEBASE:RANGE, 415  
code, :TIMEBASE:REFERENCE, 416  
code, :TRIGGER:MODE, 430  
code, :TRIGGER:SLOPE, 437  
code, :TRIGGER:SOURCE, 438  
code, :VIEW and :BLANK, 156  
code, :WAVEFORM, 482  
code, :WAVEFORM:DATA, 469  
code, :WAVEFORM:POINTS, 473  
code, :WAVEFORM:PREAMBLE, 477  
code, :WAVEFORM:SEGMENTED, 165  
code, \*RST, 111  
code, SICL library example in C, 686  
code, SICL library example in Visual Basic, 695  
code, VISA COM library example in C#, 627  
code, VISA COM library example in Visual Basic, 618  
code, VISA COM library example in Visual Basic .NET, 636  
code, VISA library example in C, 645  
code, VISA library example in C#, 664  
code, VISA library example in Visual Basic, 654  
code, VISA library example in Visual Basic .NET, 675  
colon, root commands prefixed by, 124  
color palette for hardcopy, 271  
color palette for image, 387  
Comma Separated Values (CSV) waveform data format, 392  
command classifications, 608  
command differences from 7000B Series oscilloscopes, 24  
command errors detected in Standard Event Status, 102  
command header, 609  
command headers, common, 611  
command headers, compound, 611  
command headers, simple, 611  
command strings, valid, 609  
commands quick reference, 51  
commands sent over interface, 97  
commands, more about, 607  
commands, obsolete and discontinued, 521  
common (\*) commands, 3, 95, 97  
common command headers, 611  
completion criteria for an acquisition, 159, 160  
compound command headers, 611  
compound header, 613  
computer control examples, 617  
conditions for external trigger, 233  
conditions, reset, 109, 403  
Config softkey, 30

configurations, oscilloscope, 105, 108, 112, 407  
Configure softkey, 30  
connect oscilloscope, 29  
connect sampled data points, 532  
constants for making automatic measurements, 201  
constants for scaling display factors, 201  
constants for setting trigger levels, 201  
controller initialization, 38  
copy display, 149  
core commands, 608  
count, 468  
count values, 160  
coupling, 434  
coupling for channels, 195  
create automask, 337  
CSV (Comma Separated Values) waveform data format, 392  
cumulative edge activity, 526  
current logic levels on digital channels, 125  
current oscilloscope configuration, 105, 108, 112, 407  
current probe, 209, 237  
CURRENT segment waveform save option, 394  
cursor mode, 277  
cursor position, 278, 280, 282, 283, 285  
cursor readout, 545, 547, 548  
cursor reset conditions, 109, 403  
cursor source, 279, 281  
cursor time, 545, 547, 548  
cursors track measurements, 314  
cursors, how autoscale affects, 127  
cursors, X1, X2, Y1, Y2, 276  
cycle measured, 303  
cycle time, 309

## D

data, 469  
data acquisition types, 461  
data conversion, 463  
data format for transfer, 464  
data output order, 467  
data point index, 491  
data points, 162  
data record, measurement, 474  
data record, raw acquisition, 474  
data required to fill time buckets, 159  
data structures, status reporting, 579  
data, saving and recalling, 225  
date, calibration, 183  
date, system, 399  
dB versus frequency, 240  
DC coupling for edge trigger, 434  
DC input coupling for specified channel, 195  
DC RMS measured on waveform, 327  
DC waveform generator output, 499  
DDE (Device Dependent Error) status bit, 100, 102  
decision chart, status reporting, 596  
default conditions, 109, 403

define channel labels, 199  
 define glitch trigger, 445  
 define logic thresholds, 528  
 define measurement, 299  
 define measurement source, 315  
 define trigger, 446  
 defined as, 91  
 definite-length block query response, 46  
 definite-length block response data, 92  
 delay measured to calculate phase, 310  
 delay measurement, 299  
 delay measurements, 317  
 delay parameters for measurement, 301  
 delay, how autoscale affects, 127  
 delayed time base, 413  
 delayed window horizontal scale, 421  
 delete mask, 347  
 delta time, 545  
 delta voltage measurement, 552  
 delta X cursor, 276  
 delta Y cursor, 276  
 demo, 211  
 DEMO commands, 211  
 demo signal function, 212  
 demo signal phase angle, 214  
 demo signals output control, 215  
 detecting probe types, 531, 535  
 device-defined error queue clear, 99  
 differences from 7000B Series oscilloscope commands, 24  
 differential probe heads, 202  
 differential signal type, 205  
 digital channel commands, 218, 219, 220, 221, 223  
 digital channel data, 466  
 digital channel labels, 230  
 digital channel order, 533  
 digital channel source for glitch trigger, 447  
 digital channels, 5  
 digital channels, activity and logic levels on, 125  
 digital channels, groups of, 365, 366, 368  
 digital pod, stop displaying, 132  
 digital reset conditions, 110, 404  
 DIGital<d> commands, 217  
 digitize channels, 133  
 DIGitize command, 38, 43, 462  
 digits, 92  
 disable front panel, 402  
 disable function, 537  
 disabling calibration, 186  
 disabling channel display, 196  
 disabling status register bits, 100, 113  
 discontinued and obsolete commands, 521  
 display channel labels, 229  
 display clear, 227  
 DISPLAY commands, 225  
 display commands introduction, 225  
 display connect, 532  
 display date, 399  
 display factors scaling, 201  
 display for channels, 196

display frequency span, 254  
 display measurements, 294, 314  
 display order, 533  
 display persistence, 231  
 display reference, 414, 416  
 display reference waveforms, 514  
 display reset conditions, 110, 404  
 display serial number, 151  
 display vectors, 232  
 display wave position, 533  
 display, oscilloscope, 219, 231, 243, 366, 400  
 displaying a baseline, 432  
 displaying unsynchronized signal, 432  
 DNS IP, 30  
 domain, 30  
 domain, network printer, 267  
 driver, printer, 542  
 DSO models, 5  
 duplicate mnemonics, 613  
 duration for glitch trigger, 441, 442, 446  
 duration triggering, 424  
 duty cycle measurement, 39, 294, 303

## E

ECL channel threshold, 528  
 ECL threshold voltage for digital channels, 223  
 ECL trigger threshold voltage, 567  
 edge activity, 526  
 edge coupling, 434  
 edge fall time, 304  
 edge parameter for delay measurement, 301  
 edge preshoot measured, 311  
 edge rise time, 313  
 edge slope, 437  
 edge source, 438  
 EDGE trigger commands, 433  
 edge triggering, 423  
 edges (activity) on digital channels, 125  
 edges in measurement, 299  
 elapsed time in mask test, 344  
 ellipsis, 92  
 enable channel labels, 229  
 enabling calibration, 186  
 enabling channel display, 196  
 enabling status register bits, 100, 113  
 end of string (EOS) terminator, 610  
 end of text (EOT) terminator, 610  
 end or identify (EOI), 610  
 EOI (end or identify), 610  
 EOS (end of string) terminator, 610  
 EOT (end of text) terminator, 610  
 erase data, 227  
 erase measurements, 544  
 erase screen, 534  
 error messages, 401, 569  
 error number, 401  
 error queue, 401, 589  
 error, measurement, 294  
 ESB (Event Status Bit), 114, 116  
 ESE (Standard Event Status Enable Register), 100, 588

ESR (Standard Event Status Register), 102, 587  
 event status conditions occurred, 116  
 Event Status Enable Register (ESE), 100, 588  
 Event Status Register (ESR), 102, 155, 587  
 example code, :ACQUIRE:COMPLETE, 159  
 example code, :ACQUIRE:SEGMENTED, 165  
 example code, :ACQUIRE:TYPE, 170  
 example code, :AUTOSCALE, 128  
 example code, :CHANNEL<n>:LABEL, 199  
 example code, :CHANNEL<n>:PROBE, 201  
 example code, :CHANNEL<n>:RANGE, 207  
 example code, :DIGITIZE, 134  
 example code, :DISPLAY:DATA, 228  
 example code, :DISPLAY:LABEL, 229  
 example code, :DISPLAY:ORDER, 533  
 example code, :MEASURE:PERIOD, 316  
 example code, :MEASURE:TEDGE, 318  
 example code, :MTEST, 333  
 example code, :POD<n>:THRESHOLD, 368  
 example code, :RUN/STOP, 150  
 example code, :SYSTEM:SETUP, 407  
 example code, :TIMEBASE:DELAY, 566  
 example code, :TIMEBASE:MODE, 413  
 example code, :TIMEBASE:RANGE, 415  
 example code, :TIMEBASE:REFERENCE, 416  
 example code, :TRIGGER:MODE, 430  
 example code, :TRIGGER:SLOPE, 437  
 example code, :TRIGGER:SOURCE, 438  
 example code, :VIEW and :BLANK, 156  
 example code, :WAVEFORM, 482  
 example code, :WAVEFORM:DATA, 469  
 example code, :WAVEFORM:POINTS, 473  
 example code, :WAVEFORM:PREAMBLE, 477  
 example code, :WAVEFORM:SEGMENTED, 165  
 example code, \*RST, 111  
 example programs, 5, 617  
 EXE (Execution Error) status bit, 100, 102  
 execution error detected in Standard Event Status, 102  
 exponential notation, 91  
 external glitch trigger source, 447  
 external range, 236  
 external trigger, 233, 235, 438  
 EXTERNAL trigger commands, 233  
 EXTERNAL trigger level, 435  
 external trigger probe attenuation factor, 235  
 external trigger probe sensing, 535  
 EXTERNAL trigger source, 438  
 external trigger units, 237

## F

failed waveforms in mask test, 342  
 failure, self test, 118  
 fall time measurement, 294, 304  
 Fast Fourier Transform (FFT) functions, 240, 242, 253, 254, 255, 536  
 FF values in waveform data, 469  
 FFT (Fast Fourier Transform) functions, 240, 242, 253, 254, 255, 536  
 FFT (Fast Fourier Transform) operation, 248, 481

## Index

fifty ohm impedance, disable setting, 406  
filename for hardcopy, 539  
filename for recall, 373, 497  
filename for save, 382  
filter for frequency reject, 436  
filter for high frequency reject, 426  
filter for noise reject, 431  
filter used to limit bandwidth, 194, 234  
filters to Fast Fourier Transforms, 255  
fine horizontal adjustment (vernier), 418  
fine vertical adjustment (vernier), 210  
finish pending device operations, 106  
first point displayed, 491  
FLATop window for amplitude measurements, 255  
format, 471, 476  
format for block data, 105  
format for hardcopy, 538  
format for image, 385  
format for waveform data, 392  
FormattedIO488 object, 41  
formfeed for hardcopy, 258, 262  
formulas for data conversion, 463  
frequency measurement, 39, 294, 305  
frequency resolution, 255  
frequency span of display, 254  
frequency versus dB, 240  
front panel mode, 432  
front panel Single key, 152  
front panel Stop key, 154  
front-panel lock, 402  
full-scale horizontal time, 415, 420  
full-scale vertical axis defined, 249  
function, 156, 242, 243, 247, 248, 249, 250, 251, 254, 255, 536, 537  
FUNCTION commands, 239  
function memory, 153  
function turned on or off, 537  
function, demo signal, 212  
function, waveform generator, 498  
functions, 481

## G

g(t) source, first input channel, 245  
g(t) source, math operation, 244  
g(t) source, second input channel, 246  
gateway IP, 30  
general trigger commands, 425  
glitch demo signal, 212  
glitch duration, 446  
glitch qualifier, 445  
glitch source, 447  
GLITCh trigger commands, 439  
glitch trigger duration, 441  
glitch trigger polarity, 444  
glitch trigger source, 441  
GPIB interface, 29, 30  
graticule area for hardcopy print, 259  
graticule colors, invert for hardcopy, 263, 541  
graticule colors, invert for image, 386  
grayscale palette for hardcopy, 271

grayscale palette for image, 387  
grayscale on hardcopy, 540  
greater than qualifier, 445  
greater than time, 441, 446  
groups of digital channels, 365, 366, 368, 528

## H

HANNing window for frequency resolution, 255  
hardcopy, 149, 258  
HARDcopy commands, 257  
hardcopy factors, 261, 384  
hardcopy filename, 539  
hardcopy format, 538  
hardcopy formfeed, 262  
hardcopy grayscale, 540  
hardcopy invert graticule colors, 263, 541  
hardcopy layout, 264  
hardcopy palette, 271  
hardcopy print, area, 259  
hardcopy printer driver, 542  
head type, probe, 202  
header, 609  
high resolution acquisition type, 463  
high trigger level, 428  
high-frequency reject filter, 426, 436  
high-level voltage, waveform generator, 508  
high-resolution acquisition type, 158  
hold until operation complete, 106  
holdoff time, 427  
holes in waveform data, 469  
horizontal adjustment, fine (vernier), 418  
horizontal position, 419  
horizontal scale, 417, 421  
horizontal scaling, 476  
horizontal time, 415, 420, 545  
Host name softkey, 30  
hostname, 30

## I

identification number, 104  
identification of options, 107  
idle until operation complete, 106  
IDN (Identification Number), 104  
IEEE 488.2 standard, 97  
image format, 385  
image invert graticule colors, 386  
image memory, 153  
image palette, 387  
image, save, 383  
image, save with inksaver, 386  
impedance, 197  
infinity representation, 615  
initialization, 38, 41  
initialize, 109, 403  
initialize label list, 230  
initiate acquisition, 133  
inksaver, save image with, 386  
input coupling for channels, 195

input impedance for channels, 197, 530  
input inversion for specified channel, 198  
insert label, 199  
installed options identified, 107  
instruction header, 609  
instrument number, 104  
instrument options identified, 107  
instrument requests service, 116  
instrument serial number, 151  
instrument settings, 258  
instrument status, 48  
instrument type, 104  
internal low-pass filter, 194, 234  
introduction to :ACquire commands, 157  
introduction to :BUS<n> commands, 172  
introduction to :CALibrate commands, 181  
introduction to :CHANnel<n> commands, 193  
introduction to :DEMO commands, 211  
introduction to :DIGital<d> commands, 218  
introduction to :DISPlay commands, 225  
introduction to :EXternal commands, 233  
introduction to :FUNctIon commands, 240  
introduction to :HARDcopy commands, 258  
introduction to :MARKer commands, 276  
introduction to :MEASure commands, 294  
introduction to :POD<n> commands, 365  
introduction to :RECall commands, 371  
introduction to :SAVE commands, 380  
introduction to :SYSTem commands, 398  
introduction to :TIMEbase commands, 412  
introduction to :TRIGger commands, 423  
introduction to :WAVEform commands, 461  
introduction to :WGENemory, 496  
introduction to :WMEMory<r> commands, 511  
introduction to common (\*) commands, 97  
introduction to root (: ) commands, 124  
invert graticule colors for hardcopy, 263, 541  
invert graticule colors for image, 386  
inverted masks, bind levels, 358  
inverting input for channels, 198  
IO library, referencing, 40  
IP address, 30

## K

key disable, 402  
key press detected in Standard Event Status Register, 102  
knob disable, 402  
known state, 109, 403

## L

label, 527  
label command, bus, 178  
label list, 199, 230  
label reference waveforms, 515  
label, digital channel, 220  
labels, 199, 229, 230  
labels to store calibration information, 184

labels, specifying, 225  
 LAN interface, 29, 32  
 LAN Settings softkey, 30  
 landscape layout for hardcopy, 264  
 language for program examples, 37  
 layout for hardcopy, 264  
 leakage into peak spectrum, 255  
 learn string, 105, 407  
 least significant byte first, 467  
 left reference, 416  
 legal values for channel offset, 200  
 legal values for frequency span, 254  
 legal values for offset, 247, 250  
 length for waveform data, 393  
 less than qualifier, 445  
 less than time, 442, 446  
 level for trigger voltage, 435, 443  
 LF coupling, 434  
 license information, 107  
 limits for line number, 454  
 line glitch trigger source, 447  
 line number for TV trigger, 454  
 line terminator, 91  
 LINE trigger level, 435  
 LINE trigger source, 438  
 list of channel labels, 230  
 local lockout, 402  
 lock, 402  
 lock mask to signal, 349  
 lock, analog channel protection, 406  
 lockout message, 402  
 logic level activity, 526  
 long form, 610  
 low frequency sine with glitch demo signal, 213  
 low trigger level, 429  
 lower threshold, 309  
 lower threshold voltage for measurement, 543  
 lowercase characters in commands, 609  
 low-frequency reject filter, 436  
 low-level voltage, waveform generator, 509  
 low-pass filter used to limit bandwidth, 194, 234  
 LRN (Learn Device Setup), 105  
 lsbfirst, 467

## M

magnitude of occurrence, 319  
 main sweep range, 419  
 main time base, 566  
 main time base mode, 413  
 making measurements, 294  
 MAN option for probe sense, 531, 535  
 manual cursor mode, 277  
 MARKer commands, 275  
 marker mode, 283  
 marker position, 284  
 marker readout, 547, 548  
 marker set for voltage measurement, 553, 554  
 marker sets start time, 546  
 marker time, 545  
 markers for delta voltage measurement, 552  
 markers track measurements, 314  
 markers, command overview, 276  
 markers, mode, 277  
 markers, time at start, 548  
 markers, time at stop, 547  
 markers, X delta, 282  
 markers, X1 position, 278  
 markers, X1Y1 source, 279  
 markers, X2 position, 280  
 markers, X2Y2 source, 281  
 markers, Y delta, 285  
 markers, Y1 position, 283  
 markers, Y2 position, 284  
 mask, 100, 113  
 mask command, bus, 179  
 mask statistics, reset, 343  
 mask test commands, 331  
 Mask Test Event Enable Register (MTEenable), 135  
 mask test event event register, 137  
 Mask Test Event Event Register (:MTERegister[:EVENT]), 137, 594  
 mask test run mode, 350  
 mask test termination conditions, 350  
 mask test, all channels, 336  
 mask test, enable/disable, 348  
 mask, delete, 347  
 mask, get as binary block data, 346  
 mask, load from binary block data, 346  
 mask, lock to signal, 349  
 mask, recall, 374  
 mask, save, 388  
 masks, bind levels, 358  
 master summary status bit, 116  
 math function, stop displaying, 132  
 math operations, 240  
 MAV (Message Available), 99, 114, 116  
 maximum duration, 442  
 maximum position, 414  
 maximum range for zoomed window, 420  
 maximum scale for zoomed window, 421  
 maximum vertical value measurement, 324  
 MEASure commands, 287  
 measure mask test failures, 351  
 measure overshoot, 307  
 measure period, 309  
 measure phase between channels, 310  
 measure preshoot, 311  
 measure start voltage, 553  
 measure stop voltage, 554  
 measure value at a specified time, 328  
 measure value at top of waveform, 329  
 measurement error, 294  
 measurement record, 474  
 measurement setup, 294, 315  
 measurement source, 315  
 measurement window, 330  
 measurements, AC RMS, 327  
 measurements, average value, 322  
 measurements, base value, 323  
 measurements, built-in, 39  
 measurements, clear, 297, 544  
 measurements, command overview, 294  
 measurements, DC RMS, 327  
 measurements, definition setup, 299  
 measurements, delay, 301  
 measurements, duty cycle, 303  
 measurements, fall time, 304  
 measurements, frequency, 305  
 measurements, how autoscale affects, 127  
 measurements, lower threshold level, 543  
 measurements, maximum vertical value, 324  
 measurements, minimum vertical value, 325  
 measurements, overshoot, 307  
 measurements, period, 309  
 measurements, phase, 310  
 measurements, preshoot, 311  
 measurements, pulse width, negative, 306  
 measurements, pulse width, positive, 312  
 measurements, rise time, 313  
 measurements, show, 314  
 measurements, snapshot all, 296  
 measurements, source channel, 315  
 measurements, start marker time, 547  
 measurements, stop marker time, 548  
 measurements, thresholds, 546  
 measurements, time between start and stop markers, 545  
 measurements, time between trigger and edge, 317  
 measurements, time between trigger and vertical value, 319  
 measurements, time between trigger and voltage level, 549  
 measurements, upper threshold value, 551  
 measurements, vertical amplitude, 321  
 measurements, vertical peak-to-peak, 326  
 measurements, voltage difference, 552  
 memory setup, 112, 407  
 message available bit, 116  
 message available bit clear, 99  
 message displayed, 116  
 message error, 569  
 message queue, 586  
 messages ready, 116  
 midpoint of thresholds, 309  
 minimum duration, 441  
 minimum vertical value measurement, 325  
 mixed-signal demo signals, 213  
 mixed-signal oscilloscopes, 5  
 mnemonics, duplicate, 613  
 mode, 277, 413  
 model number, 104  
 models, oscilloscope, 3  
 modes for triggering, 430  
 Modify softkey, 30  
 most significant byte first, 467  
 move cursors, 547, 548  
 msbfirst, 467  
 MSG (Message), 114, 116  
 MSO models, 5  
 MSS (Master Summary Status), 116  
 MTEenable (Mask Test Event Enable Register), 135

## Index

MTERegister[:EVENT] (Mask Test Event Event Register), 137, 594  
MTESt commands, 331  
multiple commands, 613  
multiple queries, 47  
multiply math function, 240, 248, 481  
multiply math function as g(t) source, 244

## N

name channels, 199  
name list, 230  
negative glitch trigger polarity, 444  
negative pulse width, 306  
negative pulse width measurement, 39  
negative slope, 437  
negative TV trigger polarity, 456  
network domain password, 268  
network domain user name, 270  
network printer address, 265  
network printer domain, 267  
network printer slot, 269  
network printer, apply connection settings, 266  
new line (NL) terminator, 91, 610  
NL (new line) terminator, 91, 610  
noise reject filter, 431  
noise waveform generator output, 499  
noisy sine waveform demo signal, 212  
non-core commands, 608  
non-volatile memory, label list, 178, 220, 230  
normal acquisition type, 157, 462  
normal trigger sweep mode, 423  
notices, 2  
NR1 number format, 91  
NR3 number format, 91  
NTSC, 454, 458  
NULL string, 400  
number format, 91  
number of points, 162, 472, 474  
number of time buckets, 472, 474  
numeric variables, 46  
numeric variables, reading query results into multiple, 48  
nwidth, 306

## O

obsolete and discontinued commands, 521  
obsolete commands, 608  
occurrence reported by magnitude, 549  
offset value for channel voltage, 200  
offset value for selected function, 247, 250  
offset, waveform generator, 510  
one values in waveform data, 469  
OPC (Operation Complete) command, 106  
OPC (Operation Complete) status bit, 100, 102  
OPEE (Operation Status Enable Register), 139  
Open method, 41  
operating configuration, 105, 407  
operating state, 112  
operation complete, 106

operation status condition register, 141  
Operation Status Condition Register (:OPERRegister:CONDition), 141, 591  
operation status conditions occurred, 116  
Operation Status Enable Register (OPEE), 139  
operation status event register, 143  
Operation Status Event Register (:OPERRegister[:EVENT]), 143, 590  
operation, math, 240  
operations for function, 248  
OPERRegister:CONDition (Operation Status Condition Register), 141, 591  
OPERRegister[:EVENT] (Operation Status Event Register), 143, 590  
OPT (Option Identification), 107  
optional syntax terms, 91  
options, 107  
order of digital channels on display, 533  
order of output, 467  
oscilloscope connection, opening, 41  
oscilloscope connection, verifying, 31  
oscilloscope external trigger, 233  
oscilloscope models, 3  
oscilloscope rate, 168  
oscilloscope, connecting, 29  
oscilloscope, initialization, 38  
oscilloscope, operation, 6  
oscilloscope, program structure, 38  
oscilloscope, setting up, 29  
oscilloscope, setup, 42  
output control, demo signals, 215  
output control, waveform generator, 503  
output load impedance, waveform generator, 504  
output messages ready, 116  
output queue, 106, 585  
output queue clear, 99  
output sequence, 467  
overlapped commands, 616  
overload, 206  
Overload Event Enable Register (OVL), 145  
Overload Event Register (:OVLRegister), 593  
Overload Event Register (OVLRL), 147  
overload protection, 145, 147  
overshoot of waveform, 307  
overvoltage, 206  
OVL (Overload Event Enable Register), 145  
OVLRL (Overload Event Register), 147  
OVLRL bit, 141, 143  
OVLRegister (Overload Event Register), 593

## P

PAL, 454, 458  
palette for hardcopy, 271  
palette for image, 387  
PAL-M, 454, 458  
parameters for delay measurement, 301  
parametric measurements, 294  
parser, 124, 613  
pass, self test, 118  
password, network domain, 268  
path information, recall, 375  
path information, save, 389  
pattern duration, 441, 442  
pattern for pattern trigger, 449  
PATtern trigger commands, 448  
pattern trigger format, 451  
pattern trigger qualifier, 452  
pattern triggering, 424  
peak data, 463  
peak detect, 169  
peak detect acquisition type, 158, 463  
peak-to-peak vertical value measurement, 326  
pending operations, 106  
percent of waveform overshoot, 307  
percent thresholds, 299  
period measured to calculate phase, 310  
period measurement, 39, 294, 309  
period, waveform generator, 505  
persistence, waveform, 225, 231  
phase angle, demo signals, 214  
phase measured between channels, 310  
phase measurements, 317  
phase shifted demo signals, 212  
PNG format screen image data, 228  
pod, 365, 366, 367, 368, 481, 528  
POD commands, 365  
POD data format, 466  
pod, stop displaying, 132  
points, 162, 472, 474  
points in waveform data, 462  
polarity, 456  
polarity for glitch trigger, 444  
polling synchronization with timeout, 600  
polling wait, 598  
PON (Power On) status bit, 100, 102  
portrait layout for hardcopy, 264  
position, 221, 280, 414, 419  
position cursors, 547, 548  
position in zoomed view, 419  
position waveforms, 533  
positive glitch trigger polarity, 444  
positive pulse width, 312  
positive pulse width measurement, 39  
positive slope, 437  
positive TV trigger polarity, 456  
positive width, 312  
preamble data, 476  
preamble metadata, 461  
predefined logic threshold, 528  
predefined threshold voltages, 567  
present working directory, recall operations, 375  
present working directory, save operations, 389  
preset conditions, 403  
preshoot measured on waveform, 311  
previously stored configuration, 108  
print command, 149  
print job, start, 273  
print mask test failures, 352  
print query, 563  
printer driver for hardcopy, 542  
printer, active, 260



printing, 258  
 printing in grayscale, 540  
 probe, 435  
 probe attenuation affects channel voltage range, 207  
 probe attenuation factor (external trigger), 235  
 probe attenuation factor for selected channel, 201  
 probe head type, 202  
 probe ID, 203  
 probe sense for oscilloscope, 531, 535  
 probe skew value, 204, 529  
 process sigma, mask test run, 355  
 program data, 610  
 program data syntax rules, 612  
 program initialization, 38  
 program message, 41, 97  
 program message syntax, 609  
 program message terminator, 610  
 program structure, 38  
 programming examples, 5, 617  
 protecting against calibration, 186  
 protection, 145, 147, 206  
 protection lock, 406  
 pulse waveform generator output, 499  
 pulse width, 306, 312  
 pulse width duration trigger, 441, 442, 446  
 pulse width measurement, 39, 294  
 pulse width trigger, 431  
 pulse width trigger level, 443  
 pulse width triggering, 423  
 pulse width, waveform generator, 500  
 pwidth, 312

## Q

qualifier, 446  
 qualifier, trigger pattern, 452  
 queries, multiple, 47  
 query error detected in Standard Event Status, 102  
 query responses, block data, 46  
 query responses, reading, 45  
 query results, reading into numeric variables, 46  
 query results, reading into string variables, 46  
 query return values, 615  
 query setup, 258, 276, 294, 407  
 query subsystem, 172, 218  
 querying setup, 193  
 querying the subsystem, 424  
 queues, clearing, 595  
 quick reference, commands, 51  
 quoted ASCII string, 92  
 QYE (Query Error) status bit, 100, 102

## R

ramp symmetry, waveform generator, 501  
 ramp waveform generator output, 498  
 range, 420

range for channels, 207  
 range for external trigger, 236  
 range for full-scale vertical axis, 249  
 range for glitch trigger, 446  
 range for time base, 415  
 range of offset values, 200  
 range qualifier, 445  
 ranges, value, 92  
 rate, 168  
 raw acquisition record, 474  
 RCL (Recall), 108  
 read configuration, 105  
 ReadIEEEBlock method, 41, 45, 47  
 ReadList method, 41, 45  
 ReadNumber method, 41, 45  
 readout, 545  
 ReadString method, 41, 45  
 real-time acquisition mode, 161  
 recall, 108, 371, 407  
 RECall commands, 371  
 recall filename, 373, 497  
 recall mask, 374  
 recall path information, 375  
 recall reference waveform, 377  
 recall setup, 376  
 recalling and saving data, 225  
 RECTangular window for transient signals, 255  
 reference, 416  
 reference for time base, 566  
 reference waveform save source, 395  
 reference waveform, recall, 377  
 reference waveform, save, 396  
 reference waveforms, clear, 513  
 reference waveforms, display, 514  
 reference waveforms, label, 515  
 reference waveforms, save to, 516  
 reference waveforms, skew, 517  
 reference waveforms, Y offset, 518  
 reference waveforms, Y range, 519  
 reference waveforms, Y scale, 520  
 registers, 102, 108, 112, 126, 135, 137, 139, 141, 143, 145, 147  
 registers, clearing, 595  
 reject filter, 436  
 reject high frequency, 426  
 reject noise, 431  
 remote control examples, 617  
 remove cursor information, 277  
 remove labels, 229  
 remove message from display, 400  
 reorder channels, 127  
 repetitive acquisitions, 150  
 report errors, 401  
 report transition, 317, 319  
 reporting status, 577  
 reporting the setup, 424  
 request service, 116  
 Request-for-OPC flag clear, 99  
 reset, 109  
 reset conditions, 109  
 reset defaults, waveform generator, 506  
 reset mask statistics, 343

reset measurements, 227  
 resolution of printed copy, 540  
 resource session object, 41  
 ResourceManager object, 41  
 restore configurations, 105, 108, 112, 407  
 restore labels, 229  
 restore setup, 108  
 return values, query, 615  
 returning acquisition type, 169  
 returning number of data points, 162  
 RF burst demo signal, 213  
 right reference, 416  
 ringing pulse demo signal, 212  
 rise time measurement, 294  
 rise time of positive edge, 313  
 RMS value measurement, 327  
 roll time base mode, 613  
 root (:) commands, 121, 124  
 root level commands, 3  
 RQL (Request Control) status bit, 100, 102  
 RQS (Request Service), 116  
 RST (Reset), 109  
 rules, tree traversal, 613  
 rules, truncation, 610  
 run, 117, 150  
 Run bit, 141, 143  
 run mode, mask test, 350  
 running configuration, 112, 407

## S

sample rate, 168  
 sampled data, 532  
 sampled data points, 469  
 SAV (Save), 112  
 save, 112, 380  
 SAVE commands, 379  
 save filename, 382  
 save image, 383  
 save image with inksaver, 386  
 save mask, 388  
 save mask test failures, 353  
 save path information, 389  
 save reference waveform, 396  
 save setup, 390  
 save to reference waveform location, 516  
 save waveform data, 391  
 saved image, area, 565  
 saving and recalling data, 225  
 scale, 251, 417, 421  
 scale factors output on hardcopy, 261, 384  
 scale for channels, 208  
 scale units for channels, 209  
 scale units for external trigger, 237  
 scaling display factors, 201  
 SCPI commands, 49  
 scratch measurements, 544  
 screen area for hardcopy print, 259  
 screen area for saved image, 565  
 screen image data, 228  
 SECAM, 454, 458  
 seconds per division, 417

## Index

segmented waveform save option, 394  
segments, analyze, 163  
segments, count of waveform, 479  
segments, setting number of memory, 164  
segments, setting the index, 165  
segments, time tag, 480  
select measurement channel, 315  
self-test, 118  
sensing a channel probe, 531  
sensing a external trigger probe, 535  
sensitivity of oscilloscope input, 201  
sequential commands, 616  
serial number, 151  
service request, 116  
Service Request Enable Register (SRE), 114, 583  
set center frequency, 242  
set cursors, 547, 548  
set date, 399  
set time, 409  
set up oscilloscope, 29  
setting digital display, 219  
setting digital label, 178, 220  
setting digital position, 221  
setting digital threshold, 223  
setting display, 243  
setting external trigger level, 233  
setting impedance for channels, 197  
setting inversion for channels, 198  
setting pod display, 366  
setting pod size, 367  
setting pod threshold, 368  
settings, 108, 112  
settings, instrument, 258  
setup, 158, 172, 193, 218, 225, 258, 407  
setup configuration, 108, 112, 407  
setup defaults, 109, 403  
setup memory, 108  
setup reported, 424  
setup, recall, 376  
setup, save, 390  
short form, 5, 610  
show channel labels, 229  
show measurements, 294, 314  
SICL example in C, 686  
SICL example in Visual Basic, 695  
SICL examples, 686  
sigma, mask test run, 355  
signal type, 205  
signed data, 465  
simple command headers, 611  
sine waveform demo signal, 212  
sine waveform generator output, 498  
single acquisition, 152  
single-ended probe heads, 202  
single-ended signal type, 205  
single-shot demo signal, 212  
single-shot DUT, synchronizing with, 602  
size, 367  
size, digital channels, 222  
skew, 204, 529  
skew reference waveform, 517

slope, 437  
slope (direction) of waveform, 549  
slope not valid in TV trigger mode, 437  
slope parameter for delay measurement, 301  
slot, network printer, 269  
smoothing acquisition type, 463  
snapshot all measurement, 296  
software version, 104  
source, 315  
source for function, 252, 253, 536  
source for trigger, 438  
source for TV trigger, 457  
source, automask, 338  
source, mask test, 363  
source, save reference waveform, 395  
source, waveform, 481  
span, 240  
span of frequency on display, 254  
specify measurement, 315  
square wave duty cycle, waveform generator, 502  
square waveform generator output, 498  
SRE (Service Request Enable Register), 114, 583  
SRQ (Service Request interrupt), 135, 139  
Standard Event Status Enable Register (ESE), 100, 588  
Standard Event Status Register (ESR), 102, 587  
standard for video, 458  
start acquisition, 117, 133, 150, 152  
start and stop edges, 299  
start cursor, 547  
start measurement, 294  
start print job, 273  
start time, 446, 547  
start time marker, 546  
state memory, 112  
state of instrument, 105, 407  
status, 115, 153, 155  
Status Byte Register (STB), 113, 115, 116, 581  
status data structure clear, 99  
status registers, 48  
status reporting, 577  
STB (Status Byte Register), 113, 115, 116, 581  
step size for frequency span, 254  
stop, 133, 154  
stop acquisition, 154  
stop cursor, 548  
stop displaying channel, 132  
stop displaying math function, 132  
stop displaying pod, 132  
stop on mask test failure, 354  
stop time, 446, 548  
storage, 112  
store instrument setup, 105, 112  
store setup, 112  
storing calibration information, 184  
string variables, 46  
string variables, reading multiple query results into, 47  
string variables, reading query results into multiple, 47

string, quoted ASCII, 92  
subnet mask, 30  
subsource, waveform source, 485  
subsystem commands, 3, 613  
subtract math function, 240, 248, 481  
subtract math function as g(t) source, 244  
sweep mode, trigger, 423, 432  
sweep speed set to fast to measure fall time, 304  
sweep speed set to fast to measure rise time, 313  
switch disable, 402  
syntax elements, 91  
syntax rules, program data, 612  
syntax, optional terms, 91  
syntax, program message, 609  
SYSTem commands, 397  
system commands, 399, 400, 401, 402, 407, 409  
system commands introduction, 398

## T

tdelta, 545  
tedge, 317  
telnet ports 5024 and 5025, 469  
Telnet sockets, 49  
temporary message, 400  
TER (Trigger Event Register), 155, 584  
termination conditions, mask test, 350  
test sigma, mask test run, 355  
test, self, 118  
text, writing to display, 400  
threshold, 223, 368, 528, 567  
threshold voltage (lower) for measurement, 543  
threshold voltage (upper) for measurement, 551  
thresholds, 299, 546  
thresholds used to measure period, 309  
thresholds, how autoscale affects, 127  
time base, 413, 414, 415, 416, 417, 566  
time base commands introduction, 412  
time base reset conditions, 110, 404  
time base window, 419, 420, 421  
time between points, 545  
time buckets, 159, 160  
time delay, 566  
time delta, 545  
time difference between data points, 489  
time duration, 446  
time holdoff for trigger, 427  
time interval, 317, 319, 545  
time interval between trigger and occurrence, 549  
time marker sets start time, 546  
time per division, 415  
time record, 255  
time specified, 328  
time, calibration, 190  
time, mask test run, 356  
time, start marker, 547

time, stop marker, 548  
time, system, 409  
time/div, how autoscale affects, 127  
TIMEbase commands, 411  
timebase vernier, 418  
TIMEbase:MODE, 44  
time-ordered label list, 230  
timing measurement, 294  
title channels, 199  
title, mask test, 364  
tolerance, automask, 340, 341  
top of waveform value measured, 329  
total waveforms in mask test, 345  
trace memory, 153  
track measurements, 314  
trademarks, 2  
transfer instrument state, 105, 407  
tree traversal rules, 613  
TRG (Trigger), 114, 116, 117  
TRIG OUT BNC, 185  
trigger armed event register, 141, 143  
trigger channel source, 447, 457  
TRIGger commands, 423  
TRIGger commands, general, 425  
TRIGger EDGE commands, 433  
trigger edge coupling, 434  
trigger edge slope, 437  
trigger event bit, 155  
Trigger Event Register (TER), 584  
TRIGger GLITCh commands, 439  
trigger holdoff, 427  
trigger level constants, 201  
trigger level voltage, 435  
trigger level, high, 428  
trigger level, low, 429  
trigger occurred, 116  
TRIGger PATtern commands, 448  
trigger pattern qualifier, 452  
trigger reset conditions, 110, 404  
trigger status bit, 155  
trigger sweep mode, 423  
TRIGger TV commands, 453  
trigger, edge coupling, 434  
trigger, edge level, 435  
trigger, edge reject, 436  
trigger, edge slope, 437  
trigger, edge source, 438  
trigger, glitch greater than, 441  
trigger, glitch less than, 442  
trigger, glitch level, 443  
trigger, glitch polarity, 444  
trigger, glitch qualifier, 445  
trigger, glitch range, 446  
trigger, glitch source, 447  
trigger, high frequency reject filter, 426  
trigger, holdoff, 427  
trigger, mode, 430  
trigger, noise reject filter, 431  
trigger, sweep, 432  
trigger, threshold, 567  
trigger, TV line, 454  
trigger, TV mode, 455, 568

trigger, TV polarity, 456  
trigger, TV source, 457  
trigger, TV standard, 458  
truncation rules, 610  
TST (Self Test), 118  
tstart, 547  
tstop, 548  
TTL threshold voltage for digital channels, 223, 528  
TTL trigger threshold voltage, 567  
turn function on or off, 537  
turn off channel, 132  
turn off channel labels, 229  
turn off digital pod, 132  
turn off math function, 132  
turn on channel labels, 229  
turn on channel number display, 533  
turning channel display on and off, 196  
turning off/on function calculation, 243  
turning vectors on or off, 532  
TV mode, 455, 568  
TV trigger commands, 453  
TV trigger line number setting, 454  
TV trigger mode, 457  
TV trigger polarity, 456  
TV trigger standard setting, 458  
TV triggering, 424  
tvmode, 568  
type, 486

## U

units per division, 208, 209, 237, 417  
units per division (vertical) for function, 208, 251  
units, automask, 339  
unsigned data, 465  
unsigned mode, 487  
upper threshold, 309  
upper threshold voltage for measurement, 551  
uppercase characters in commands, 609  
URQ (User Request) status bit, 100, 102  
USB (Device) interface, 29  
user defined channel labels, 199  
user defined threshold, 528  
user event conditions occurred, 116  
user name, network domain, 270  
User's Guide, 6  
user-defined threshold voltage for digital channels, 223  
user-defined trigger threshold, 567  
USR (User Event bit), 114, 116

## V

valid command strings, 609  
value, 319  
value measured at base of waveform, 323  
value measured at specified time, 328  
value measured at top of waveform, 329  
value ranges, 92

values required to fill time buckets, 160  
VBA, 40, 618  
vectors turned on or off, 532  
vectors, display, 232  
vectors, turning on or off, 225  
vernier, channel, 210  
vernier, horizontal, 418  
vertical adjustment, fine (vernier), 210  
vertical amplitude measurement, 321  
vertical axis defined by RANGe, 249  
vertical axis range for channels, 207  
vertical offset for channels, 200  
vertical peak-to-peak measured on waveform, 326  
vertical scale, 208, 251  
vertical scaling, 476  
vertical threshold, 528  
vertical value at center screen, 247, 250  
vertical value maximum measured on waveform, 324  
vertical value measurements to calculate overshoot, 307  
vertical value minimum measured on waveform, 325  
video line to trigger on, 454  
video standard selection, 458  
view, 156, 240, 488, 533  
view turns function on or off, 537  
VISA COM example in C#, 627  
VISA COM example in Visual Basic, 618  
VISA COM example in Visual Basic .NET, 636  
VISA example in C, 645  
VISA example in C#, 664  
VISA example in Visual Basic, 654  
VISA example in Visual Basic .NET, 675  
VISA examples, 618, 645  
Visual Basic .NET, VISA COM example, 636  
Visual Basic .NET, VISA example, 675  
Visual Basic 6.0, 41  
Visual Basic for Applications, 40, 618  
Visual Basic, SICL library example, 695  
Visual Basic, VISA COM example, 618  
Visual Basic, VISA example, 654  
voltage crossing reported or not found, 549  
voltage difference between data points, 492  
voltage difference measured, 552  
voltage level for active trigger, 435  
voltage marker used to measure waveform, 553, 554  
voltage offset value for channels, 200  
voltage probe, 209, 237  
voltage ranges for channels, 207  
voltage ranges for external trigger, 236  
voltage threshold, 299

## W

WAI (Wait To Continue), 119  
wait, 119  
wait for operation complete, 106  
Wait Trig bit, 141, 143  
waveform base value measured, 323

WAVeform command, 39  
 WAVeform commands, 459  
 waveform data, 461  
 waveform data format, 392  
 waveform data length, 393  
 waveform data, save, 391  
 waveform generator, 496  
 waveform generator amplitude, 507  
 waveform generator function, 498  
 waveform generator high-level voltage, 508  
 waveform generator low-level voltage, 509  
 waveform generator offset, 510  
 waveform generator output control, 503  
 waveform generator output load  
   impedance, 504  
 waveform generator period, 505  
 waveform generator pulse width, 500  
 waveform generator ramp symmetry, 501  
 waveform generator reset defaults, 506  
 waveform generator square wave duty  
   cycle, 502  
 waveform introduction, 461  
 waveform maximum vertical value  
   measured, 324  
 waveform minimum vertical value  
   measured, 325  
 waveform must cross voltage level to be an  
   occurrence, 549  
 WAVeform parameters, 44  
 waveform peak-to-peak vertical value  
   measured, 326  
 waveform period, 309  
 waveform persistence, 225  
 waveform RMS value measured, 327  
 waveform save option for segments, 394  
 waveform source, 481  
 waveform source subsource, 485  
 waveform vertical amplitude, 321  
 waveform voltage measured at marker, 553,  
   554  
 waveform, byte order, 467  
 waveform, count, 468  
 waveform, data, 469  
 waveform, format, 471  
 waveform, points, 472, 474  
 waveform, preamble, 476  
 waveform, type, 486  
 waveform, unsigned, 487  
 waveform, view, 488  
 waveform, X increment, 489  
 waveform, X origin, 490  
 waveform, X reference, 491  
 waveform, Y increment, 492  
 waveform, Y origin, 493  
 waveform, Y reference, 494  
 WAVeform:FORMat, 44  
 waveforms, mask test run, 357  
 Web control, 49  
 WGEN commands, 495  
 WGEN trigger source, 438  
 what's new, 21  
 width, 446

window, 419, 420, 421  
 window time, 415  
 window time base mode, 413  
 window, measurement, 330  
 windows, 255  
 windows as filters to Fast Fourier  
   Transforms, 255  
 windows for Fast Fourier Transform  
   functions, 255  
 WMEMory commands, 511  
 word format, 471  
 word format for data transfer, 465  
 write text to display, 400  
 WriteEEEBlock method, 41, 47  
 WriteList method, 41  
 WriteNumber method, 41  
 WriteString method, 41

## X

X axis markers, 276  
 X delta, 282  
 X delta, mask scaling, 360  
 X1 and X2 cursor value difference, 282  
 X1 cursor, 276, 278, 279  
 X1, mask scaling, 359  
 X2 cursor, 276, 280, 281  
 X-axis functions, 412  
 X-increment, 489  
 X-origin, 490  
 X-reference, 491  
 X-Y mode, 412, 413

## Y

Y axis markers, 276  
 Y offset, reference waveform, 518  
 Y range, reference waveform, 519  
 Y scale, reference waveform, 520  
 Y1 and Y2 cursor value difference, 285  
 Y1 cursor, 276, 279, 283, 285  
 Y1, mask scaling, 361  
 Y2 cursor, 276, 281, 284, 285  
 Y2, mask scaling, 362  
 Y-axis value, 493  
 Y-increment, 492  
 Y-origin, 493, 494  
 Y-reference, 494

## Z

zero values in waveform data, 469  
 zoomed time base, 413  
 zoomed time base measurement window, 330  
 zoomed time base mode, how autoscale  
   affects, 127  
 zoomed window horizontal scale, 421