



# **Agilent InfiniiVision 6000 Series Oscilloscopes**

## **Programmer's Guide**



**Agilent Technologies**

# Notices

© Agilent Technologies, Inc. 2005-2008

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

## Trademarks

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

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

## Manual Part Number

Version 05.15.0000

## Edition

July 31, 2008

Available in electronic format only

Agilent Technologies, Inc.  
1900 Garden of the Gods Road  
Colorado Springs, CO 80907 USA

## Warranty

**The material contained in this document is provided “as is,” and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.**

## Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

## Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as “Commercial computer software” as defined in DFAR 252.227-7014 (June 1995), or as a “commercial item” as defined in FAR 2.101(a) or as “Restricted computer software” as defined in FAR 52.227-19 (June 1987) or any equivalent

agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies’ standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

## Safety Notices

### CAUTION

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

### WARNING

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

## In This Book

This book is your guide to programming the 6000 Series oscilloscopes:

**Table 1** InfiniiVision 6000 Series Oscilloscope Models

Channels	Input Bandwidth			
	1 GHz	500 MHz	300 MHz	100 MHz
4 analog + 16 digital (mixed-signal)	MSO6104A/L	MSO6054A/L	MSO6034A	MSO6014A/L
2 analog + 16 digital (mixed-signal)	MSO6102A	MSO6052A	MSO6032A	MSO6012A
4 analog	DSO6104A/L	DSO6054A/L	DSO6034A	DSO6014A/L
2 analog	DSO6102A	DSO6052A	DSO6032A	DSO6012A

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

- Chapter 1, ["What's New"](#) on page 19, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, ["Setting Up"](#) on page 37, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, ["Getting Started"](#) on page 47, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, ["Commands Quick Reference"](#) on page 61, is a brief listing of the 6000 Series oscilloscope commands and syntax.

The next chapters provide reference information:

- Chapter 5, ["Commands by Subsystem"](#) on page 107, describes the set of commands that belong to an individual subsystem and explains the function of each command. Command arguments and syntax are described. Some command descriptions have example code.
- Chapter 6, ["Commands A-Z"](#) on page 559, contains an alphabetical listing of all command elements.
- Chapter 7, ["Obsolete and Discontinued Commands"](#) on page 587, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 8, ["Error Messages"](#) on page 635, lists the instrument error messages that can occur while programming the oscilloscope.

The command descriptions in this reference show upper and lowercase characters. For example, :AUToscale indicates that the entire command

name is :AUTOSCALE. The short form, :AUT, is also accepted by the oscilloscope.

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

- Chapter 9, "[Status Reporting](#)" on page 643, describes the oscilloscope's status registers and how to check the status of the instrument.
- Chapter 10, "[Synchronizing Acquisitions](#)" on page 665, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- Chapter 11, "[More About Oscilloscope Commands](#)" on page 675, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 12, "[Programming Examples](#)" on page 699.

#### **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/6000manual>"

# Contents

In This Book 3

## 1 What's New

What's New in Version 5.15	20
What's New in Version 5.10	22
What's New in Version 5.00	23
What's New in Version 4.10	25
What's New in Version 4.00	27
What's New in Version 3.50	29
What's New in Version 3.00	31
Command Differences From 54620/54640 Series Oscilloscopes	33

## 2 Setting Up

Step 1. Install Agilent IO Libraries Suite software	38
Step 2. Connect and set up the oscilloscope	39
Using the USB (Device) Interface	39
Using the LAN Interface	39
Using the GPIB Interface	40
Step 3. Verify the oscilloscope connection	41

## 3 Getting Started

Basic Oscilloscope Program Structure	48
Initializing	48
Capturing Data	48
Analyzing Captured Data	49

Programming the Oscilloscope	50
Referencing the IO Library	50
Opening the Oscilloscope Connection via the IO Library	51
Initializing the Interface and the Oscilloscope	51
Using :AUToscale to Automate Oscilloscope Setup	52
Using Other Oscilloscope Setup Commands	52
Capturing Data with the :DIGitize Command	53
Reading Query Responses from the Oscilloscope	55
Reading Query Results into String Variables	56
Reading Query Results into Numeric Variables	56
Reading Definite-Length Block Query Response Data	56
Sending Multiple Queries and Reading Results	57
Checking Instrument Status	58
Other Ways of Sending Commands	59
Telnet Sockets	59
Sending SCPI Commands Using Browser Web Control	59

## 4 Commands Quick Reference

Command Summary	62
Syntax Elements	104
Number Format	104
<NL> (Line Terminator)	104
[ ] (Optional Syntax Terms)	104
{ } (Braces)	104
::= (Defined As)	104
< > (Angle Brackets)	105
... (Ellipsis)	105
n,...,p (Value Ranges)	105
d (Digits)	105
Quoted ASCII String	105
Definite-Length Block Response Data	105

## 5 Commands by Subsystem

Common (*) Commands	109
*CLS (Clear Status)	113
*ESE (Standard Event Status Enable)	114
*ESR (Standard Event Status Register)	116
*IDN (Identification Number)	118
*LRN (Learn Device Setup)	119
*OPC (Operation Complete)	120

*OPT (Option Identification)	121
*RCL (Recall)	122
*RST (Reset)	123
*SAV (Save)	126
*SRE (Service Request Enable)	127
*STB (Read Status Byte)	129
*TRG (Trigger)	131
*TST (Self Test)	132
*WAI (Wait To Continue)	133
Root (:) Commands	134
:ACTivity	137
:AER (Arm Event Register)	138
:AUToscale	139
:AUToscale:AMODE	141
:AUToscale:CHANnels	142
:BLANK	143
:CDISplay	144
:DIGitize	145
:HWEenable (Hardware Event Enable Register)	147
:HWERegister:CONDition (Hardware Event Condition Register)	149
:HWERegister[:EVENT] (Hardware Event Event Register)	151
:MERGe	153
:OPEE (Operation Status Enable Register)	154
:OPERRegister:CONDition (Operation Status Condition Register)	156
:OPERRegister[:EVENT] (Operation Status Event Register)	158
:OVLenable (Overload Event Enable Register)	160
:OVLRegister (Overload Event Register)	162
:PRINt	164
:RUN	165
:SERial	166
:SINGle	167
:STATus	168
:STOP	169
:TER (Trigger Event Register)	170
:VIEW	171
:ACQuire Commands	172
:ACQuire:AALias	174
:ACQuire:COMPlEte	175
:ACQuire:COUNt	176
:ACQuire:DAALias	177
:ACQuire:MODE	178

:ACQuire:POINts	179
:ACQuire:RSIGnal	180
:ACQuire:SEGmented:COUNT	181
:ACQuire:SEGmented:INDEX	182
:ACQuire:SRATe	184
:ACQuire:TYPE	185
:BUS<n> Commands	187
:BUS<n>:BIT<m>	189
:BUS<n>:BITS	190
:BUS<n>:CLEar	192
:BUS<n>:DISPlay	193
:BUS<n>:LABel	194
:BUS<n>:MASK	195
:CALibrate Commands	196
:CALibrate:DATE	197
:CALibrate:LABel	198
:CALibrate:STARt	199
:CALibrate:STATus	200
:CALibrate:SWITCh	201
:CALibrate:TEMPerature	202
:CALibrate:TIME	203
:CHANnel<n> Commands	204
:CHANnel<n>:BWLimit	207
:CHANnel<n>:COUPling	208
:CHANnel<n>:DISPlay	209
:CHANnel<n>:IMPedance	210
:CHANnel<n>:INVert	211
:CHANnel<n>:LABel	212
:CHANnel<n>:OFFSet	213
:CHANnel<n>:PROBe	214
:CHANnel<n>:PROBe:ID	215
:CHANnel<n>:PROBe:SKEW	216
:CHANnel<n>:PROBe:STYPe	217
:CHANnel<n>:PROTection	218
:CHANnel<n>:RANGe	219
:CHANnel<n>:SCALe	220
:CHANnel<n>:UNITs	221
:CHANnel<n>:VERNier	222
:DIGital<n> Commands	223
:DIGital<n>:DISPlay	225



:DiGital<n>:LABel	226
:DiGital<n>:POSition	227
:DiGital<n>:SIZE	228
:DiGital<n>:THReshold	229
:DISPlay Commands	230
:DISPlay:CLEar	232
:DISPlay:DATA	233
:DISPlay:LABel	235
:DISPlay:LABList	236
:DISPlay:PERsistence	237
:DISPlay:SOURce	238
:DISPlay:VECTors	239
:EXTeRnal Trigger Commands	240
:EXTeRnal:BWLimit	242
:EXTeRnal:IMPedance	243
:EXTeRnal:PROBe	244
:EXTeRnal:PROBe:ID	245
:EXTeRnal:PROBe:STYPe	246
:EXTeRnal:PROTection	247
:EXTeRnal:RANGe	248
:EXTeRnal:UNITs	249
:FUNCTion Commands	250
:FUNCTion:CENTer	253
:FUNCTion:DISPlay	254
:FUNCTion:GOFT:OPERation	255
:FUNCTion:GOFT:SOURce1	256
:FUNCTion:GOFT:SOURce2	257
:FUNCTion:OFFSet	258
:FUNCTion:OPERation	259
:FUNCTion:RANGe	260
:FUNCTion:REFerence	261
:FUNCTion:SCALE	262
:FUNCTion:SOURce1	263
:FUNCTion:SOURce2	264
:FUNCTion:SPAN	265
:FUNCTion:WINDow	266
:HARDcopy Commands	267
:HARDcopy:AREA	269
:HARDcopy:APRinter	270
:HARDcopy:FACTors	271

:HARDcopy:FFeEd	272
:HARDcopy:INKSaver	273
:HARDcopy:PALette	274
:HARDcopy:PRinter:LIST	275
:HARDcopy:STARt	276
:MARKer Commands	277
:MARKer:MODE	279
:MARKer:X1Position	280
:MARKer:X1Y1source	281
:MARKer:X2Position	282
:MARKer:X2Y2source	283
:MARKer:XDELta	284
:MARKer:Y1Position	285
:MARKer:Y2Position	286
:MARKer:YDELta	287
:MEASure Commands	288
:MEASure:CLEar	295
:MEASure:COUNter	296
:MEASure:DEFine	297
:MEASure:DELay	300
:MEASure:DUTYcycle	302
:MEASure:FALLtime	303
:MEASure:FREQuency	304
:MEASure:NWIDth	305
:MEASure:OVERshoot	306
:MEASure:PERiod	308
:MEASure:PHASe	309
:MEASure:PREShoot	310
:MEASure:PWIDth	311
:MEASure:RISetime	312
:MEASure:SDEViation	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:VRATio	327
:MEASure:VRMS	328
:MEASure:VTIMe	329
:MEASure:VTOP	330
:MEASure:XMAX	331
:MEASure:XMIN	332
:POD Commands	333
:POD<n>:DISPlay	334
:POD<n>:SIZE	335
:POD<n>:THReshold	336
:RECall Commands	338
:RECall:FiLename	339
:RECall:IMAGe[:STARt]	340
:RECall:PWD	341
:RECall:SETup[:STARt]	342
:SAVE Commands	343
:SAVE:FiLename	345
:SAVE:IMAGe[:STARt]	346
:SAVE:IMAGe:AREA	347
:SAVE:IMAGe:FACTors	348
:SAVE:IMAGe:FORMat	349
:SAVE:IMAGe:INKSaver	350
:SAVE:IMAGe:PALette	351
:SAVE:PWD	352
:SAVE:SETup[:STARt]	353
:SAVE:WAVEform[:STARt]	354
:SAVE:WAVEform:FORMat	355
:SAVE:WAVEform:LENGth	356
:SBUS Commands	357
:SBUS:BUSDoctor:ADDReSS	360
:SBUS:BUSDoctor:BAUDRate	361
:SBUS:BUSDoctor:CHANnel	362
:SBUS:BUSDoctor:MODE	363
:SBUS:CAN:COUNt:ERRor	364
:SBUS:CAN:COUNt:OVERload	365
:SBUS:CAN:COUNt:RESet	366
:SBUS:CAN:COUNt:TOTal	367
:SBUS:CAN:COUNt:UTILization	368
:SBUS:DISPlay	369
:SBUS:FLEXray:COUNt:NULL	370

:SBUS:FLEXray:COUNT:RESet	371
:SBUS:FLEXray:COUNT:SYNC	372
:SBUS:FLEXray:COUNT:TOTal	373
:SBUS:IIC:ASIZe	374
:SBUS:LIN:PARity	375
:SBUS:MODE	376
:SBUS:SPI:WIDTh	377
:SBUS:UART:BASE	378
:SBUS:UART:COUNT:ERRor	379
:SBUS:UART:COUNT:RESet	380
:SBUS:UART:COUNT:RXFRames	381
:SBUS:UART:COUNT:TXFRames	382
:SBUS:UART:FRAMing	383
:SYSTem Commands	384
:SYSTem:DATE	385
:SYSTem:DSP	386
:SYSTem:ERRor	387
:SYSTem:LOCK	388
:SYSTem:PROTection:LOCK	389
:SYSTem:SETup	390
:SYSTem:TIME	392
:TIMebase Commands	393
:TIMebase:MODE	395
:TIMebase:POSition	396
:TIMebase:RANGe	397
:TIMebase:REFClock	398
:TIMebase:REFerence	399
:TIMebase:SCALE	400
:TIMebase:VERNier	401
:TIMebase:WINDow:POSition	402
:TIMebase:WINDow:RANGe	403
:TIMebase:WINDow:SCALE	404
:TRIGger Commands	405
General :TRIGger Commands	408
:TRIGger:HFReject	409
:TRIGger:HOLDoff	410
:TRIGger:MODE	411
:TRIGger:NREJect	412
:TRIGger:PATTern	413
:TRIGger:SWEep	415

- :TRIGger:CAN Commands 416
- :TRIGger:CAN:PATtern:DATA 418
- :TRIGger:CAN:PATtern:DATA:LENGth 419
- :TRIGger:CAN:PATtern:ID 420
- :TRIGger:CAN:PATtern:ID:MODE 421
- :TRIGger:CAN:SAMPlEpoint 422
- :TRIGger:CAN:SIGNal:BAUDrate 423
- :TRIGger:CAN:SOURce 424
- :TRIGger:CAN:TRIGger 425
- :TRIGger:DURation Commands 427
- :TRIGger:DURation:GREaterthan 428
- :TRIGger:DURation:LESSthan 429
- :TRIGger:DURation:PATtern 430
- :TRIGger:DURation:QUALifier 431
- :TRIGger:DURation:RANGe 432
- :TRIGger:EBURst Commands 433
- :TRIGger:EBURst:COUNt 434
- :TRIGger:EBURst:IDLE 435
- :TRIGger:EBURst:SLOPe 436
- :TRIGger[:EDGE] Commands 437
- :TRIGger[:EDGE]:COUPling 438
- :TRIGger[:EDGE]:LEVeL 439
- :TRIGger[:EDGE]:REJect 440
- :TRIGger[:EDGE]:SLOPe 441
- :TRIGger[:EDGE]:SOURce 442
- :TRIGger:FLEXray Commands 443
- :TRIGger:FLEXray:ERRor:TYPE 444
- :TRIGger:FLEXray:FRAMe:CCBase 446
- :TRIGger:FLEXray:FRAMe:CCRepetition 447
- :TRIGger:FLEXray:FRAMe:ID 448
- :TRIGger:FLEXray:FRAMe:TYPE 449
- :TRIGger:FLEXray:TIME:CBASe 450
- :TRIGger:FLEXray:TIME:CREPetition 451
- :TRIGger:FLEXray:TIME:SEGment 452
- :TRIGger:FLEXray:TIME:SLOT 453
- :TRIGger:FLEXray:TRIGger 454
- :TRIGger:GLITch Commands 455
- :TRIGger:GLITch:GREaterthan 457
- :TRIGger:GLITch:LESSthan 458
- :TRIGger:GLITch:LEVeL 459
- :TRIGger:GLITch:POLarity 460
- :TRIGger:GLITch:QUALifier 461

:TRIGger:GLITch:RANGe	462
:TRIGger:GLITch:SOURce	463
:TRIGger:IIC Commands	464
:TRIGger:IIC:PATtern:ADDReSS	465
:TRIGger:IIC:PATtern:DATA	466
:TRIGger:IIC:PATtern:DATA2	467
:TRIGger:IIC:SOURce:CLOCK	468
:TRIGger:IIC:SOURce:DATA	469
:TRIGger:IIC:TRIGger:QUALifier	470
:TRIGger:IIC:TRIGger[:TYPE]	471
:TRIGger:LIN Commands	473
:TRIGger:LIN:ID	474
:TRIGger:LIN:SAMPlepoint	475
:TRIGger:LIN:SIGNal:BAUDrate	476
:TRIGger:LIN:SOURce	477
:TRIGger:LIN:STANdard	478
:TRIGger:LIN:SYNCbreak	479
:TRIGger:LIN:TRIGger	480
:TRIGger:SEQuence Commands	481
:TRIGger:SEQuence:COUNt	482
:TRIGger:SEQuence:EDGE	483
:TRIGger:SEQuence:FIND	484
:TRIGger:SEQuence:PATtern	485
:TRIGger:SEQuence:RESet	486
:TRIGger:SEQuence:TIMer	487
:TRIGger:SEQuence:TRIGger	488
:TRIGger:SPI Commands	489
:TRIGger:SPI:CLOCK:SLOPe	490
:TRIGger:SPI:CLOCK:TIMEout	491
:TRIGger:SPI:FRAMing	492
:TRIGger:SPI:PATtern:DATA	493
:TRIGger:SPI:PATtern:WIDTh	494
:TRIGger:SPI:SOURce:CLOCK	495
:TRIGger:SPI:SOURce:DATA	496
:TRIGger:SPI:SOURce:FRAMe	497
:TRIGger:TV Commands	498
:TRIGger:TV:LINE	499
:TRIGger:TV:MODE	500
:TRIGger:TV:POLarity	501
:TRIGger:TV:SOURce	502
:TRIGger:TV:STANdard	503
:TRIGger:UART Commands	504

:TRIGger:UART:BAUDrate	506
:TRIGger:UART:BITorder	507
:TRIGger:UART:BURSt	508
:TRIGger:UART:DATA	509
:TRIGger:UART:IDLE	510
:TRIGger:UART:PARity	511
:TRIGger:UART:POLarity	512
:TRIGger:UART:QUALifier	513
:TRIGger:UART:SOURce:RX	514
:TRIGger:UART:SOURce:TX	515
:TRIGger:UART:TYPE	516
:TRIGger:UART:WIDTh	517
:TRIGger:USB Commands	518
:TRIGger:USB:SOURce:DMINus	519
:TRIGger:USB:SOURce:DPLus	520
:TRIGger:USB:SPEed	521
:TRIGger:USB:TRIGger	522
:WAVEform Commands	523
:WAVEform:BYTeorder	531
:WAVEform:COUNt	532
:WAVEform:DATA	533
:WAVEform:FORMat	535
:WAVEform:POINts	536
:WAVEform:POINts:MODE	538
:WAVEform:PREamble	540
:WAVEform:SEGmented:COUNt	543
:WAVEform:SEGmented:TTAG	544
:WAVEform:SOURce	545
:WAVEform:SOURce:SUBSource	549
:WAVEform:TYPE	550
:WAVEform:UNSigned	551
:WAVEform:VIEW	552
:WAVEform:XINCrement	553
:WAVEform:XORigin	554
:WAVEform:XREFerence	555
:WAVEform:YINCrement	556
:WAVEform:YORigin	557
:WAVEform:YREFerence	558

## 6 Commands A-Z

### 7 Obsolete and Discontinued Commands

:CHANnel:ACTivity	592
:CHANnel:LABel	593
:CHANnel:THReshold	594
:CHANnel2:SKEW	595
:CHANnel<n>:INPut	596
:CHANnel<n>:PMODE	597
:DISPlay:CONNect	598
:DISPlay:ORDer	599
:ERASe	600
:EXTernal:INPut	601
:EXTernal:PMODE	602
:FUNction:SOURce	603
:FUNction:VIEW	604
:HARDcopy:DESTination	605
:HARDcopy:DEVice	606
:HARDcopy:FILename	607
:HARDcopy:FORMat	608
:HARDcopy:GRAYscale	609
:HARDcopy:IGColors	610
:HARDcopy:PDRiver	611
:MEASure:LOWer	612
:MEASure:SCRatch	613
:MEASure:TDELta	614
:MEASure:THResholds	615
:MEASure:TMAX	616
:MEASure:TMIN	617
:MEASure:TSTart	618
:MEASure:TSTOp	619
:MEASure:TVOLt	620
:MEASure:UPPer	622
:MEASure:VDELta	623
:MEASure:VSTart	624
:MEASure:VSTOp	625
:PRINt?	626
:TIMEbase:DELay	628
:TRIGger:CAN:ACKNowledge	629
:TRIGger:CAN:SIGNAL:DEFinition	630
:TRIGger:LIN:SIGNAL:DEFinition	631



:TRIGger:THReshold	632
:TRIGger:TV:TVMMode	633

## 8 Error Messages

## 9 Status Reporting

Status Reporting Data Structures	645
Status Byte Register (STB)	648
Service Request Enable Register (SRE)	650
Trigger Event Register (TER)	651
Output Queue	652
Message Queue	653
(Standard) Event Status Register (ESR)	654
(Standard) Event Status Enable Register (ESE)	655
Error Queue	656
Operation Status Event Register (:OPERRegister[:EVENTt])	657
Operation Status Condition Register (:OPERRegister:CONDition)	658
Arm Event Register (AER)	659
Hardware Event Event Register (:HWERegister[:EVENTt])	660
Hardware Event Condition Register (:HWERegister:CONDition)	661
Clearing Registers and Queues	662
Status Reporting Decision Chart	663

## 10 Synchronizing Acquisitions

Synchronization in the Programming Flow	666
Set Up the Oscilloscope	666
Acquire a Waveform	666
Retrieve Results	666
Blocking Synchronization	667
Polling Synchronization With Timeout	668
Synchronizing with a Single-Shot Device Under Test (DUT)	670
Synchronization with an Averaging Acquisition	672

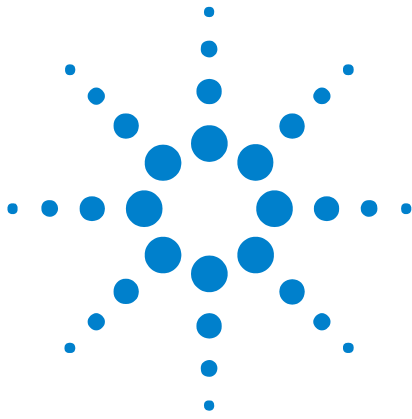
## 11 More About Oscilloscope Commands

Command Classifications	676
Core Commands	676
Non-Core Commands	676
Obsolete Commands	676
Valid Command/Query Strings	677
Program Message Syntax	677
Command Tree	681
Duplicate Mnemonics	693
Tree Traversal Rules and Multiple Commands	693
Query Return Values	696
All Oscilloscope Commands Are Sequential	697

## 12 Programming Examples

SICL Examples	700
SICL Example in C	700
SICL Example in Visual Basic	709
VISA Examples	718
VISA Example in C	718
VISA Example in Visual Basic	727
VISA Example in C#	737
VISA Example in Visual Basic .NET	751
VISA COM Examples	764
VISA COM Example in Visual Basic	764
VISA COM Example in C#	774
VISA COM Example in Visual Basic .NET	785

## Index



# 1

## What's New

What's New in Version 5.15	<a href="#">20</a>
What's New in Version 5.10	<a href="#">22</a>
What's New in Version 5.00	<a href="#">23</a>
What's New in Version 4.10	<a href="#">25</a>
What's New in Version 4.00	<a href="#">27</a>
What's New in Version 3.50	<a href="#">29</a>
What's New in Version 3.00	<a href="#">31</a>
Command Differences From 54620/54640 Series Oscilloscopes	<a href="#">33</a>



## What's New in Version 5.15

New features in version 5.15 of the InfiniiVision 6000 Series oscilloscope software are:

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

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

### New Commands

Command	Description
:FUNction:GOFT:OPERation (see <a href="#">page 255</a> )	Selects the math operation for the internal g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNction:GOFT:SOURce1 (see <a href="#">page 256</a> )	Selects the first input channel for the g(t) source.
:FUNction:GOFT:SOURce2 (see <a href="#">page 257</a> )	Selects the second input channel for the g(t) source.
:FUNction:SOURce1 (see <a href="#">page 263</a> )	Selects the first source for the ADD, SUBTract, and MULTiply arithmetic operations or the single source for the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNction:SOURce2 (see <a href="#">page 264</a> )	Selects the second input channel for the ADD, SUBTract, and MULTiply arithmetic operations.
:MEASure:VRATio (see <a href="#">page 327</a> )	Measures and returns the ratio of AC RMS values of the specified sources expressed in dB.
:SYSTem:PROTection:LOCK (see <a href="#">page 389</a> )	Disables/enables the fifty ohm input impedance setting.

**Changed Commands**

Command	Differences
:ACQuire:COUNT (see <a href="#">page 176</a> )	The :ACQuire:COUNT 1 command has been deprecated. The AVERage acquisition type with a count of 1 is functionally equivalent to the HRESolution acquisition type; however, you should select the high-resolution acquisition mode with the :ACQuire:TYPE HRESolution command instead.
:FUNCTion:OPERation (see <a href="#">page 259</a> )	The ADD parameter is new, and now that waveform math can be performed using channels 3 and 4, this command selects the operation only.
:FUNCTion:WINDow (see <a href="#">page 266</a> )	You can now select the Blackman-Harris FFT window.

**Obsolete Commands**

Obsolete Command	Current Command Equivalent	Behavior Differences
:FUNCTion:SOURce (see <a href="#">page 603</a> )	:FUNCTion:SOURce1 (see <a href="#">page 263</a> )	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.

## What's New in Version 5.10

New features in version 5.10 of the InfiniiVision 6000 Series oscilloscope software are:

- Segmented memory acquisition mode, enabled with Option SGM.

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

### New Commands

Command	Description
:ACQuire:SEGMented:COUNT (see <a href="#">page 181</a> )	Sets the number of memory segments.
:ACQuire:SEGMented:INDEX (see <a href="#">page 182</a> )	Selects the segmented memory index.
:WAVEform:SEGMented:COUNT (see <a href="#">page 543</a> )	Returns the number of segments in the currently acquired waveform data.
:WAVEform:SEGMented:TTAG (see <a href="#">page 544</a> )	Returns the time tag for the selected segmented memory index.

### Changed Commands

Command	Differences
:ACQuire:MODE (see <a href="#">page 178</a> )	You can now select the SEGMented memory mode.

## What's New in Version 5.00

New features in version 5.00 of the InfiniiVision 6000 Series oscilloscope software are:

- The ability to trigger on and decode UART/RS-232 serial bus data with a four-channel oscilloscope that includes the Option 232 license.
- The :SAVE and :RECall command subsystems.
- Changes to the :HARDcopy sommand subsystem to make a clearer distinction between printing and save/recall functionality.

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

### New Commands

Command	Description
:HARDcopy:START (see <a href="#">page 276</a> )	Starts a print job.
:HARDcopy:APRinter (see <a href="#">page 270</a> )	Sets the active printer.
:HARDcopy:AREA (see <a href="#">page 269</a> )	Specifies the area of the display to print (currently SCReen only).
:HARDcopy:INKSaver (see <a href="#">page 273</a> )	Inverts screen colors to save ink when printing.
:HARDcopy:PRinter:LIST (see <a href="#">page 275</a> )	Returns a list of the available printers.
:RECall Commands (see <a href="#">page 338</a> )	Commands for recalling previously saved oscilloscope setups and traces.
:SAVE Commands (see <a href="#">page 343</a> )	Commands for saving oscilloscope setups and traces, screen images, and data.
:SBUS:UART:BASE (see <a href="#">page 378</a> )	Determines the base to use for the UART decode display.
:SBUS:UART:COUNt:ERRor (see <a href="#">page 379</a> )	Returns the UART error frame count.
:SBUS:UART:COUNt:RESet (see <a href="#">page 380</a> )	Resets the UART frame counters.
:SBUS:UART:COUNt:RXFRames (see <a href="#">page 381</a> )	Returns the UART Rx frame count.
:SBUS:UART:COUNt:TXFRames (see <a href="#">page 382</a> )	Returns the UART Tx frame count.
:SBUS:UART:FRAMing (see <a href="#">page 383</a> )	Determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.
:TRIGger:UART Commands (see <a href="#">page 504</a> )	Commands for triggering on UART/RS-232 signals.
:WAVEform:SOURce:SUBSource (see <a href="#">page 549</a> )	Selects subsource when :WAVEform:SOURce is SBUS (serial decode).

## 1 What's New

### Changed Commands

Command	Differences
:SBUS:MODE (see <a href="#">page 376</a> )	You can now select the UART serial bus decode mode.
:TRIGger:MODE (see <a href="#">page 411</a> )	You can now select the UART trigger mode.

### Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:FILENAME (see <a href="#">page 607</a> )	:RECall:FILENAME (see <a href="#">page 339</a> ) :SAVE:FILENAME (see <a href="#">page 339</a> )	
:HARDcopy:FORMat (see <a href="#">page 608</a> )	:HARDcopy:APRinter (see <a href="#">page 270</a> ) :SAVE:IMAGe:FORMat (see <a href="#">page 349</a> ) :SAVE:WAVEform:FORMat (see <a href="#">page 355</a> )	
:HARDcopy:IGColors (see <a href="#">page 610</a> )	:HARDcopy:INKSaver (see <a href="#">page 273</a> )	
:HARDcopy:PDRiver (see <a href="#">page 611</a> )	:HARDcopy:APRinter (see <a href="#">page 270</a> )	



## What's New in Version 4.10

New features in version 4.10 of the InfiniiVision 6000 Series oscilloscope software are:

- The ability to trigger on and decode FlexRay serial bus data using a Decomsys BusDoctor 2 protocol analyzer with a four-channel mixed-signal oscilloscope that includes the Option FRS license.
- The square root waveform math function.
- Several new hardcopy printer drivers.

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

### New Commands

Command	Description
:SBUS:BUSDoctor:ADDReSS (see <a href="#">page 360</a> )	Sets/queries the four fields in the BusDoctor LAN IP Address.
:SBUS:BUSDoctor:BAUDrate (see <a href="#">page 361</a> )	Sets/queries the baud rate for the BusDoctor from 2.5 Mb/s to 10 Mb/s.
:SBUS:BUSDoctor:CHANnel (see <a href="#">page 362</a> )	Sets/queries the FlexRay channel that the BusDoctor analyzes/preprocesses.
:SBUS:BUSDoctor:MODE (see <a href="#">page 363</a> )	Sets/queries the operating mode of the BusDoctor.
:SBUS:FLEXray:COUNt:NULL? (see <a href="#">page 370</a> )	Returns the FlexRay null frame count.
:SBUS:FLEXray:COUNt:RESet (see <a href="#">page 371</a> )	Resets the FlexRay frame counters.
:SBUS:FLEXray:COUNt:SYNC? (see <a href="#">page 372</a> )	Returns the FlexRay sync frame count.
:SBUS:FLEXray:COUNt:TOTal? (see <a href="#">page 373</a> )	Returns the FlexRay total frame count.
:TRIGger:FLEXray:ERRor:TYPE (see <a href="#">page 444</a> )	Sets/queries the FlexRay error type to trigger on.
:TRIGger:FLEXray:FRAMe:CCBase (see <a href="#">page 446</a> )	Sets/queries the base of the FlexRay cycle count (in the frame header) to trigger on.
:TRIGger:FLEXray:FRAMe:CCRepetition (see <a href="#">page 447</a> )	Sets/queries the repetition number of the FlexRay cycle count (in the frame header) to trigger on.
:TRIGger:FLEXray:FRAMe:ID (see <a href="#">page 448</a> )	Sets/queries the FlexRay frame ID to trigger on.
:TRIGger:FLEXray:FRAMe:TYPE (see <a href="#">page 449</a> )	Sets/queries the FlexRay frame type to trigger on.
:TRIGger:FLEXray:TIME:CBASe (see <a href="#">page 450</a> )	Sets/queries the base of the FlexRay cycle to trigger on.

## 1 What's New

Command	Description
:TRIGger:FLEXray:TIME:CREPetition (see <a href="#">page 451</a> )	Sets/queries the repetition number of the FlexRay cycle to trigger on.
:TRIGger:FLEXray:TIME:SEGMENT (see <a href="#">page 452</a> )	Sets/queries the FlexRay segment type.
:TRIGger:FLEXray:TIME:SLOT (see <a href="#">page 453</a> )	Sets/queries the FlexRay slot type and ID.
:TRIGger:FLEXray:TRIGger (see <a href="#">page 454</a> )	Sets/queries the FlexRay trigger mode.

### Changed Commands

Command	Differences
:FUNCTION:OPERation (see <a href="#">page 259</a> )	You can now select the SQRT (square root) waveform math function.
:SBUS:MODE (see <a href="#">page 376</a> )	You can now select the FLEXray serial bus decode mode.
:TRIGger:MODE (see <a href="#">page 411</a> )	You can now select the FLEXray trigger mode.
:HARDcopy:PDRiver (see <a href="#">page 611</a> )	You can now select the new DJPR0kx50, DJ55xx, PS470, and LJFastraster printer drivers.

## What's New in Version 4.00

New features in version 4.00 of the InfiniiVision 6000 Series oscilloscope software are:

- The ability to :AUToscale selected channels only and specify the acquisition type and mode that is set after an :AUToscale.
- The :BUS command subsystem for controlling up to two buses made up of digital channels.
- Additional :CALibrate commands for starting the user calibration procedure, displaying the status of the last user calibration, and displaying the temperature change since the last user calibration.

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

### New Commands

Command	Description
:AUToscale:AMODE (see <a href="#">page 141</a> )	Specifies whether to keep the current acquisition type and mode after subsequent autoscales.
:AUToscale:CHANnels (see <a href="#">page 142</a> )	Specifies whether to autoscale the currently displayed channels or all channels.
:BUS<n>:BIT<m> (see <a href="#">page 189</a> )	Includes or excludes the selected bit in a bus definition.
:BUS<n>:BITS (see <a href="#">page 190</a> )	Includes or excludes a list of bits in a bus definition.
:BUS<n>:CLEar (see <a href="#">page 192</a> )	Excludes all digital channels from a bus definition
:BUS<n>:DISPlay (see <a href="#">page 193</a> )	Displays or hides the bus on the oscilloscope display.
:BUS<n>:LABel (see <a href="#">page 194</a> )	Assigns a label string to a bus.
:BUS<n>:MASK (see <a href="#">page 195</a> )	Includes or excludes bits in a bus definition according to a mask.
:CALibrate:STARt (see <a href="#">page 199</a> )	Starts the user calibration procedure.
:CALibrate:STATus? (see <a href="#">page 200</a> )	Displays the summary results of the last user calibration procedure.
:CALibrate:TEMPerature? (see <a href="#">page 202</a> )	Displays the change in temperature since the last user calibration procedure.

## Changed Commands

Command	Differences
:AUToscale (see <a href="#">page 139</a> )	You can now specify which channels to autoscale.
:BLANK (see <a href="#">page 143</a> )	Now, you can also use this command with digital channel buses.
:DIGitize (see <a href="#">page 145</a> )	Now, you can also use this command with digital channel buses.
:STATus (see <a href="#">page 168</a> )	Now, you can also use this command with digital channel buses.
:VIEW (see <a href="#">page 171</a> )	Now, you can also use this command with digital channel buses.
:WAVEform:SOURce (see <a href="#">page 545</a> )	Now, you can also use this command with digital channel buses.

## What's New in Version 3.50

New features in version 3.50 of the InfiniiVision 6000 Series oscilloscope software are:

- The CAN and LIN options have been added to the :SBUS:MODE (serial decode mode) command.
- The :SBUS:CAN:COUNt commands have been added to count CAN bus frames, count load utilization, and reset the counters.
- The ALLerrors, OVERload, and ACKerror options have been added to the :TRIGger:CAN:TRIGger command.
- The :TRIGger:LIN:ID, :TRIGger:LIN:SAMPlepoint, :TRIGger:LIN:STANdard, and :TRIGger:LIN:SYNCbreak commands have been added.
- The :SBUS:LIN:PARity command has been added.
- The ID (for Frame Id) option has been added to the :TRIGger:LIN:TRIGger command.
- The :HWERegister:CONDition, :HWERegister[:EVENT], and :HWE commands for the hardware event condition, event, and enable registers have been added.

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

### New Commands

Command	Description
:SBUS:CAN:COUNt:ERRor? (see <a href="#">page 364</a> )	Returns the CAN bus error frame count.
:SBUS:CAN:COUNt:OVERload? (see <a href="#">page 364</a> )	Returns the CAN bus overload frame count.
:SBUS:CAN:COUNt:RESet (see <a href="#">page 364</a> )	Resets the CAN bus counters.
:SBUS:CAN:COUNt:TOTal? (see <a href="#">page 364</a> )	Returns the CAN bus total frame count.
:SBUS:CAN:COUNt:UTILization? (see <a href="#">page 364</a> )	Returns a percentage showing CAN bus utilization.
:SBUS:IIC:ASIZe (see <a href="#">page 374</a> )	Determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.
:SBUS:LIN:PARity (see <a href="#">page 375</a> )	Determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.
:TRIGger:LIN:ID (see <a href="#">page 474</a> )	Defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

Command	Description
:TRIGger:LIN:SAMPlpoint (see <a href="#">page 475</a> )	Sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.
:TRIGger:LIN:STANdard (see <a href="#">page 478</a> )	Sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.
:TRIGger:LIN:SYNCbreak (see <a href="#">page 479</a> )	Sets the length of the LIN sync break to be greater than or equal to 11, 12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.
:HWEenable (see <a href="#">page 147</a> )	Sets or reads the hardware event enable mask register.
:HWERegister:CONDition? (see <a href="#">page 149</a> )	Queries the hardware event condition register.
:HWERegister[:EVENT]? (see <a href="#">page 151</a> )	Queries the hardware event event register.

### Changed Commands

Command	Differences
:SBUS:MODE (see <a href="#">page 376</a> )	The CAN and LIN serial bus decode modes have been added.
:TRIGger:CAN:TRIGger (see <a href="#">page 425</a> )	The ALLerrors, OVERload, and ACKerror options have been added.
:TRIGger:LIN:TRIGger (see <a href="#">page 480</a> )	The ID (for Frame Id) option has been added.

### Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:CAN:SIGNAL:DEFinition (see <a href="#">page 630</a> )	none	
:TRIGger:LIN:SIGNAL:DEFinition (see <a href="#">page 631</a> )	none	

## What's New in Version 3.00

New features in version 3.00 of the InfiniiVision 6000 Series oscilloscope software are:

- The :SBUS command subsystem for controlling serial decode bus display, mode, and other options.
- The EBUrst trigger mode and supporting :TRIGger:EBURst commands.
- The :ACQuire:AALias and :ACQuire:DAALias commands.
- The :WAVeform:POINts:MODE command.
- The :MEASure:SDEViation command.
- The :TIMEbase:REFClock command.
- Changes to the :TRIGger:IIC commands.
- Changes to the :TRIGger:SEQuence:TRIGger command.
- Changes to the :ACQuire:TYPE and :WAVeform:TYPE commands to add HRESolution type.
- Changes to the :BLANk, :DIGitize, :STATus, :VIEW, and :WAVeform:SOURce commands to include the serial decode bus.
- Changes to the :HARDcopy:FORMat command to support the PNG, ASCIixy, and BINary format types.
- Changes to the :DISPlay:DATA? query and the :PRINt command to support the PNG format.
- Changes to the :WAVeform:POINts command to set from 2000 to 8,000,000 points (in 1-2-5 sequence) when the waveform points mode is MAXimum or RAW.

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

### New Commands

Command	Description
:ACQuire:AALias? (see <a href="#">page 174</a> )	Returns the current state of the oscilloscope's anti-alias control.
:ACQuire:DAALias (see <a href="#">page 177</a> )	Sets the oscilloscope's disable anti-alias mode.
:MEASure:SDEViation (see <a href="#">page 313</a> )	Measures the std deviation of a waveform.
:SBUS:DISPlay (see <a href="#">page 369</a> )	Controls the decoded serial bus display.
:SBUS:MODE (see <a href="#">page 376</a> )	Determines the decode mode for the serial bus.
:SBUS:SPI:WIDTh (see <a href="#">page 377</a> )	Determines the number of bits in a word of decoded data for SPI.
:TIMEbase:REFClock (see <a href="#">page 398</a> )	Enables or disables the 10 MHz REF BNC input/output.

Command	Description
:TRIGger:EBURst:COUNT (see <a href="#">page 434</a> )	Sets the Nth edge of burst edge counter resource.
:TRIGger:EBURst:IDLE (see <a href="#">page 435</a> )	Sets the Nth edge in a burst idle resource.
:TRIGger:EBURst:SLOPe (see <a href="#">page 434</a> )	Specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.
:TRIGger:IIC:PATtern:DATa2 (see <a href="#">page 467</a> )	Sets IIC data 2.
:WAVeform:POINts:MODE (see <a href="#">page 538</a> )	Sets the waveform points mode.

### Changed Commands

Command	Differences
:ACQuire:TYPE (see <a href="#">page 185</a> )	The HRESolution type has been added for smoothing at slower sweep speeds.
:BLANk (see <a href="#">page 143</a> )	Now, you can also use this command with the serial decode bus.
:DIGitize (see <a href="#">page 145</a> )	Now, you can also use this command with the serial decode bus.
:DISPlay:DATA (see <a href="#">page 233</a> )	Now, the PNG format is supported in the query.
:HARDcopy:FORMat (see <a href="#">page 608</a> )	Now, the PNG, ASCIixy, and BINary formats are also supported.
:PRINt (see <a href="#">page 164</a> )	Now, the PNG option is supported
:STATus (see <a href="#">page 168</a> )	Now, you can also use this command with the serial decode bus.
:TRIGger:IIC:TRIGger[:TYPE] (see <a href="#">page 471</a> )	The ANACknowledge, R7Data2, and W7Data2 types have been added.
:TRIGger:MODE (see <a href="#">page 411</a> )	The EBURst mode has been added.
:TRIGger:SEQuence:TRIGger (see <a href="#">page 488</a> )	The EDGE2,COUNT,NREFind (no re-find) option has been added.
:VIEW (see <a href="#">page 171</a> )	Now, you can now use this command with the serial decode bus.
:WAVeform:POINts (see <a href="#">page 536</a> )	Now, you can set from 2000 to 8,000,000 points (in 1-2-5 sequence) when the waveform points mode is MAXimum or RAW.
:WAVeform:SOURce (see <a href="#">page 545</a> )	Now, you can also use this command with the serial decode bus.
:WAVeform:TYPE (see <a href="#">page 550</a> )	The HRESolution type has been added for smoothing at slower sweep speeds.



## Command Differences From 54620/54640 Series Oscilloscopes

The main differences between the version 1.00 programming command set for the InfiniiVision 6000 Series oscilloscopes and the 54620/54640 Series oscilloscopes are related to:

- :HARDcopy and :DISPlay command subsystem changes for USB printers and the high resolution color display.
- New standards supported by the :TRIGger:TV commands.
- Support for 113xA Series probes.
- New "RAW" :WAVEform:POINts option for retrieving raw acquisition record data.
- Discontinuance of the common commands for macros.

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

### New Commands

Command	Description
:ACQuire:RSIGnal (see <a href="#">page 180</a> )	Selects the 10 MHz reference signal mode.
:CHANnel<n>:PROBe:ID? (see <a href="#">page 215</a> )	Returns the type of probe attached to the specified oscilloscope channel.
:CHANnel<n>:PROBe:STYPe (see <a href="#">page 217</a> )	Sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes, and determines how offset is applied.
:CHANnel<n>:VERNier (see <a href="#">page 222</a> )	Specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).
:DIGital<n>:SIZE (see <a href="#">page 228</a> )	Specifies the size of digital channels on the display.
:EXTernal:PROBe:ID (see <a href="#">page 245</a> )	Returns the type of probe attached to the external trigger input.
:EXTernal:PROBe:STYPe (see <a href="#">page 246</a> )	Sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes, and determines how offset is applied.
:HARDcopy:FILeName (see <a href="#">page 607</a> )	Sets the output filename for print formats whose output is a file. Replaces the 5462x/4x :HARDcopy:DESTination (see <a href="#">page 605</a> ) command.
:HARDcopy:PDRIver (see <a href="#">page 611</a> )	Sets the hardcopy printer driver.
:HARDcopy:IGColors (see <a href="#">page 610</a> )	Specifies whether graticule colors are inverted.

Command	Description
:HARDcopy:PALETTE (see <a href="#">page 274</a> )	Sets the hardcopy palette color. Replaces the 5462x/4x :HARDcopy:GRAYscale (see <a href="#">page 609</a> ) command.
:OPERRegister:CONDition? (see <a href="#">page 156</a> )	Returns the integer value contained in the "Operation Status Condition Register" on page 156 (a new register in addition to the "Operation Status Event Register" on page 158 whose value is returned by the :OPERRegister[:EVENT]? (see <a href="#">page 158</a> ) query).
:POD<n>:SIZE (see <a href="#">page 335</a> )	Specifies the size of digital channels on the display.
:TIMEbase:VERNier (see <a href="#">page 401</a> )	Specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

### Changed Commands

Command	Differences From 5462x/4x Oscilloscopes
:ACquire:COUNt (see <a href="#">page 176</a> )	The count can be set to any value from 1 to 65536 (instead of 16383).
:DISPlay:DATA (see <a href="#">page 233</a> )	The BMP8bit <format> option has been added to the query. There is a new <palette> option which can be MONochrome, GRAYscale, or COLor in the query, or just MONochrome in the command.
:DISPlay:SOURce (see <a href="#">page 238</a> )	The number of pixel memory locations is 10 (instead of 3).
:HARDcopy:FORMat (see <a href="#">page 608</a> )	There is now the BMP8bit format (instead of TIFF) and the PRINter0 or PRINter1 formats (in place of LASerjet, DESKjet, EPSON, or SEIKO). See the new :HARDcopy:PDRiver (see <a href="#">page 611</a> ) command for setting the hardcopy printer driver.
*LRN (see <a href="#">page 119</a> )	The Learn Device Setup query return format matches the IEEE 488.2 specification which says that the query result must contain ":SYST:SET " before the binary block data. (This was not the case in the 5462x/4x oscilloscopes.)
:MERGe (see <a href="#">page 153</a> )	The number of pixel memory locations is 10 (instead of 3).

Command	Differences From 5462x/4x Oscilloscopes
*OPT (see <a href="#">page 121</a> )	The Option Identification query return format now has license information (in addition to the I/O module ID information fields which are now always zero).
:OVLRegister (see <a href="#">page 162</a> )	The Overload Event Register is now a 16-bit register (instead of 8-bit) and it contains bits that identify when faults occur on the oscilloscope channels (in addition to the bits that identify when overloads occur).
:PRINt (see <a href="#">page 164</a> )	The options are now: COLor (instead of HIRes), GRAYscale (instead of LORes), PRINter0 (instead of PARallel), BMP8bit (instead of TIFF). (The PCL option is now invalid.)
*RCL (Recall) (see <a href="#">page 122</a> )	The number of instrument state locations is 10 (instead of 3 for the 54620 Series oscilloscopes or 4 for the 54640 Series oscilloscopes).
*SAV (Save) (see <a href="#">page 126</a> )	The number of instrument state locations is 10 (instead of 3 for the 54620 Series oscilloscopes or 4 for the 54640 Series oscilloscopes).
*TRG (Trigger) (see <a href="#">page 131</a> )	The *TRG has the same effect as the :DIGitize command with no parameters (instead of the :RUN command).
:TRIGger:TV:MODE (see <a href="#">page 500</a> )	The modes have been renamed (however, old forms of the mode names are still accepted).
:TRIGger:TV:STANdard (see <a href="#">page 503</a> )	The P480L60HZ, P720L60HZ, P1080L24HZ, P1080L25HZ, I1080L50HZ, and I1080L60HZ standards are supported (in addition to GENeric, NTSC, PALM, PAL, and SECam).
:VIEW (see <a href="#">page 171</a> )	The number of pixel memory locations is 10 (instead of 3).
:WAVeform:COUNt? (see <a href="#">page 532</a> )	The count can be any value from 1 to 65536 (instead of 16383).
:WAVeform:POINts (see <a href="#">page 536</a> )	There is a new RAW "number of points" option for retrieving the raw acquisition record data. Also the maximum number of points that can be retrieved from the normal measurement record is 1000 (instead of 2000).
:WAVeform:PREamble (see <a href="#">page 540</a> )	The xincrement format is 64-bit floating point NR3 (instead of 32-bit), and the yreference format is 32-bit NR1 (instead of 16-bit).

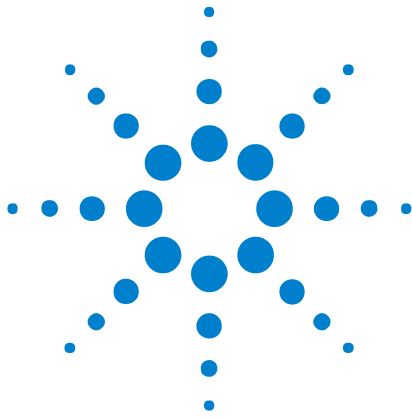
Command	Differences From 5462x/4x Oscilloscopes
:WAVeform:XINCrement (see <a href="#">page 553</a> )	The x-increment value from the preamble is returned in 64-bit (instead of 32-bit) floating point NR3 format.
:WAVeform:YREFerence (see <a href="#">page 558</a> )	The y-reference value from the preamble is returned in 32-bit (instead of 16-bit) NR1 format.

### Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:DESTination (see <a href="#">page 605</a> )	:HARDcopy:FILeName (see <a href="#">page 607</a> )	
:HARDcopy:GRAYscale (see <a href="#">page 609</a> )	:HARDcopy:PALETTE (see <a href="#">page 274</a> )	
:PRINt? (see <a href="#">page 626</a> )	:DISPlay:DATA? (see <a href="#">page 233</a> )	The options are now: COLor (instead of HIRes), GRAYscale (instead of LORes), PRINter0 (instead of PARallel), BMP8bit (instead of TIFF). (The DISK and PCL options are now invalid.)

### Discontinued Commands

Command	Description
*DMC	Define Macro.
*EMC	Enable Macro.
*GMC	Get Macro Contents.
*LMC	Learn Macro.
*PMC	Purge Macro.



## 2 Setting Up

Step 1. Install Agilent IO Libraries Suite software [38](#)

Step 2. Connect and set up the oscilloscope [39](#)

Step 3. Verify the oscilloscope connection [41](#)

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



## **Step 1. Install Agilent IO Libraries Suite software**

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

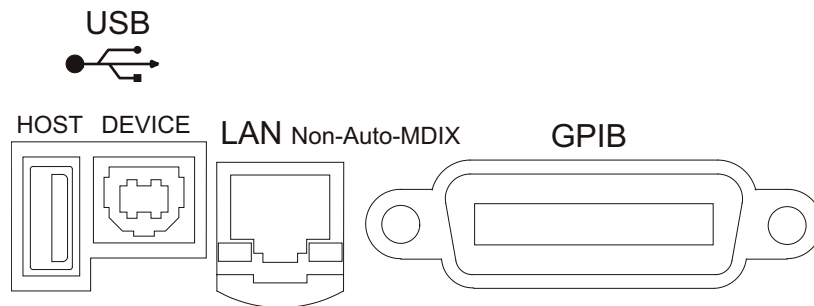
You can also download the Agilent IO Libraries Suite software from the web at:

- ["http://www.agilent.com/find/iolib"](http://www.agilent.com/find/iolib)

## Step 2. Connect and set up the oscilloscope

The 6000 Series oscilloscope has three different interfaces you can use for programming: USB (device), LAN, or GPIB.

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



**Figure 1** Control Connectors on Rear Panel

### Using the USB (Device) Interface

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

This is a USB 2.0 high-speed port.

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

### Using the LAN Interface

- 1 If the controller PC isn't already connected to the local area network (LAN), do that first.
- 2 Get the oscilloscope's network parameters (hostname, domain, IP address, subnet mask, gateway IP, DNS IP, etc.) from your network administrator.
- 3 Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the back of the oscilloscope.

- 4 On the oscilloscope, verify that the controller interface is enabled:
  - a Press the **Utility** button.
  - b Using the softkeys, press **I/O** and **Control**.
  - c Ensure the box next to **LAN** is selected (■). If not (□), use the Entry knob to select **LAN**; then, press the **Control** softkey again.
- 5 Configure the oscilloscope's LAN interface:
  - a Press the **Configure** softkey until "LAN" is selected.
  - b Press the **LAN Settings** softkey.
  - c Press the **Addresses** softkey. Use the **IP Options** softkey and the Entry knob to select DHCP, AutoIP, or netBIOS. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the IP Address, Subnet Mask, Gateway IP, and DNS IP values. When you are done, press the return (up arrow) softkey.
  - d Press the **Domain** softkey. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the Host name and the Domain name. When you are done, press the return (up arrow) softkey.

### Using the GPIB Interface

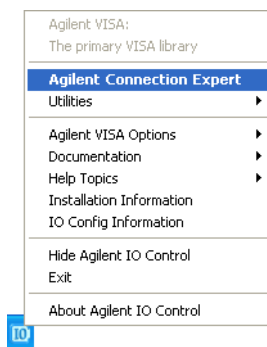
- 1 Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the back of the oscilloscope.
- 2 On the oscilloscope, verify that the controller interface is enabled:
  - a Press the **Utility** button.
  - b Using the softkeys, press **I/O** and **Control**.
  - c Use the Entry knob to select "GPIB"; then, press the **Control** softkey again.

Ensure the box next to **GPIB** is selected (■). If not (□), use the Entry knob to select **GPIB**; then, press the **Control** softkey again.
- 3 Configure the oscilloscope's GPIB interface:
  - a Press the **Configure** softkey until "GPIB" is selected.
  - b Use the Entry knob to select the **Address** value.

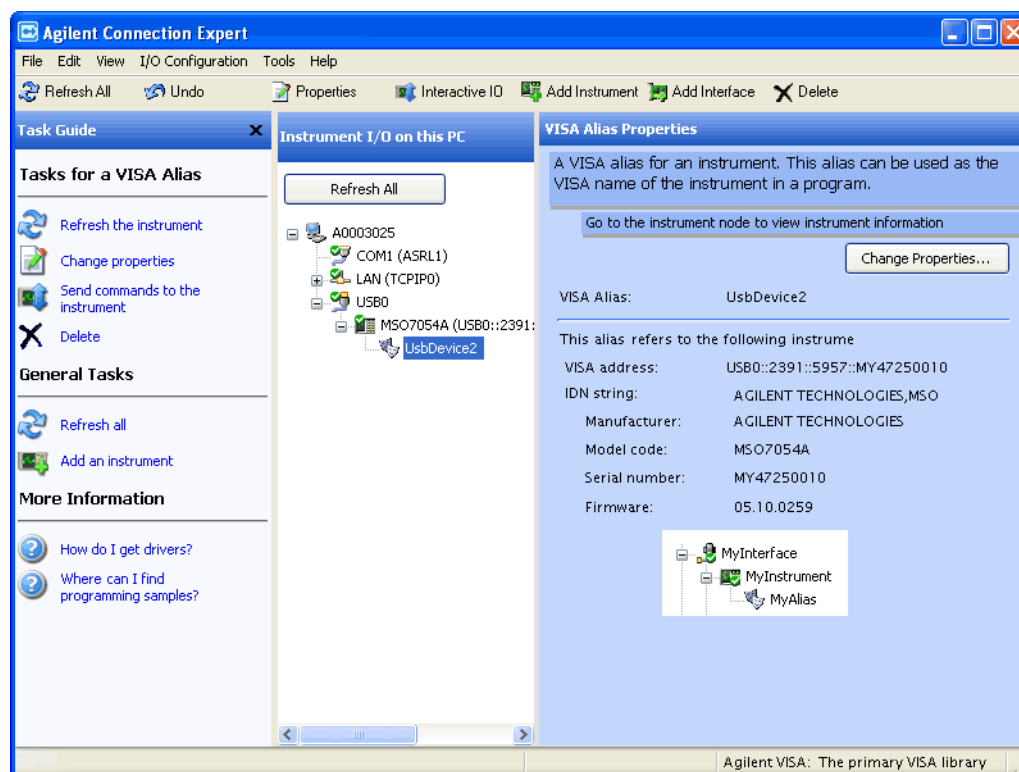


### Step 3. Verify the oscilloscope connection

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



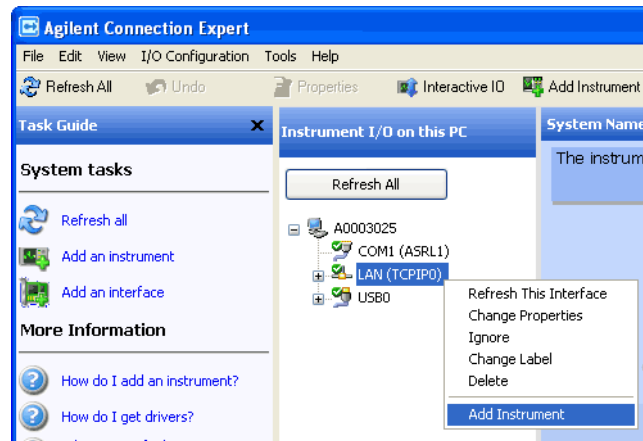
- 2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



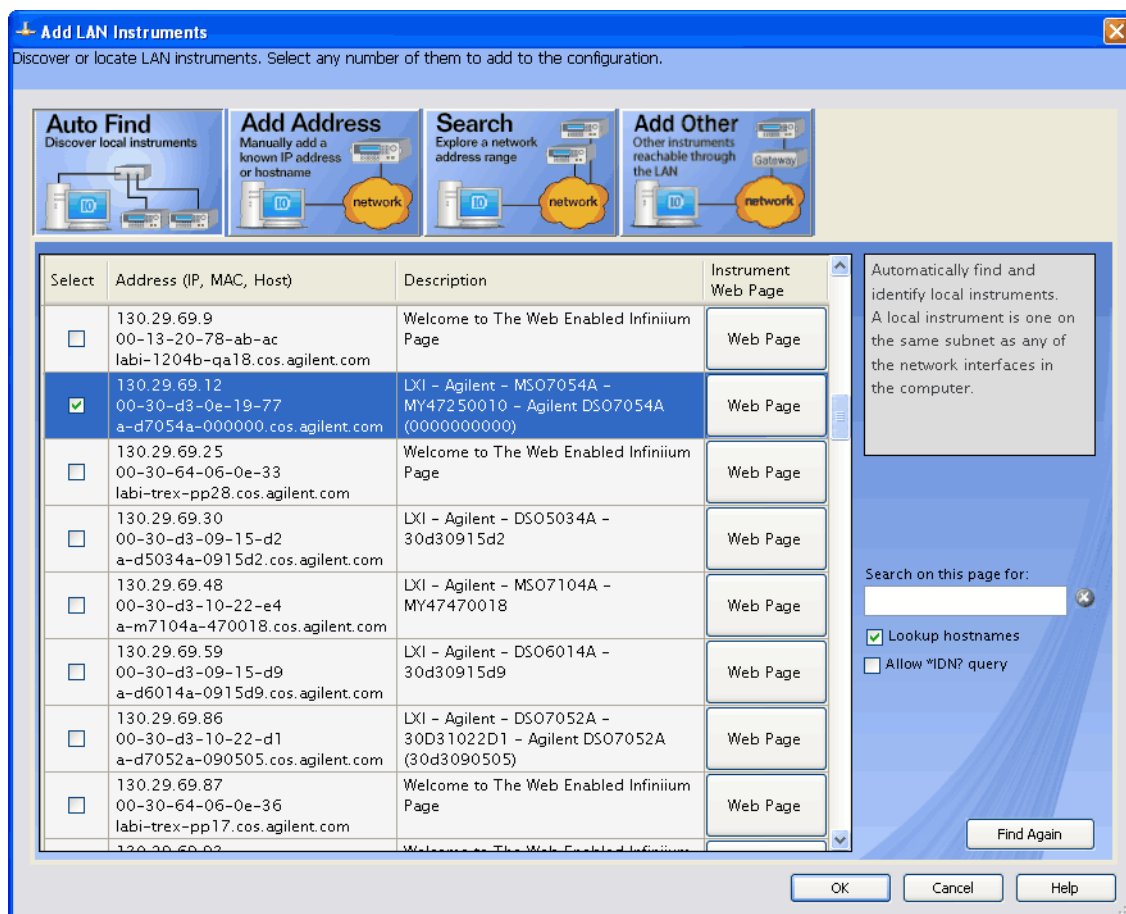
## 2 Setting Up

You must manually add instruments on LAN interfaces:

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



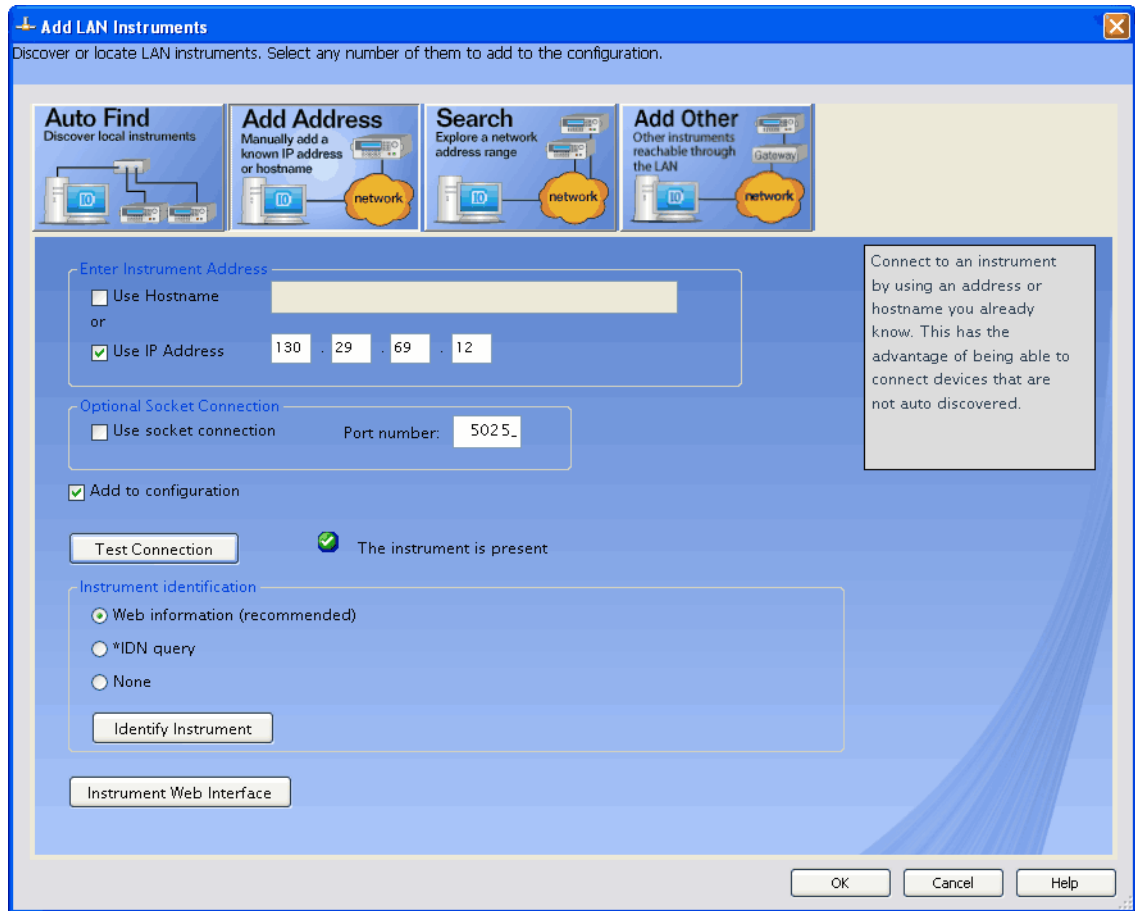
- b If the oscilloscope is on the same subnet, select it, and click **OK**.



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

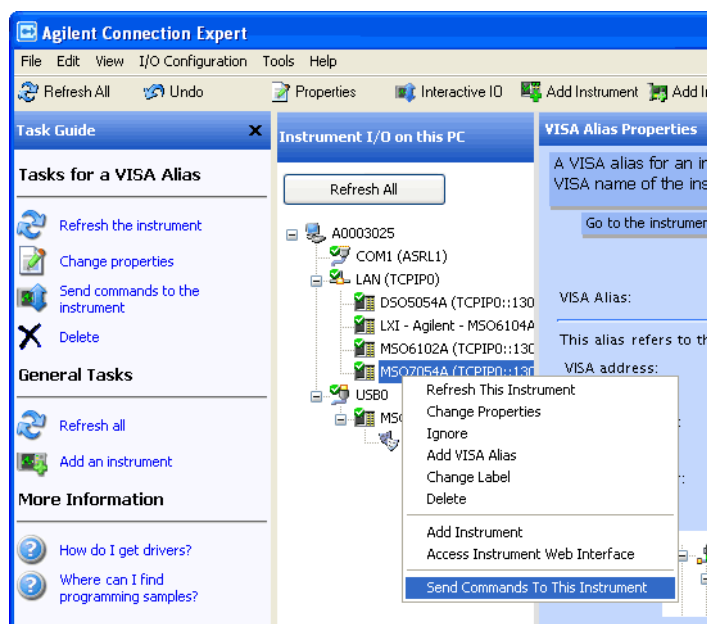
- i In the next dialog, select either **Hostname** or **IP address**, and enter the oscilloscope's hostname or IP address.
- ii Click **Test Connection**.

## 2 Setting Up

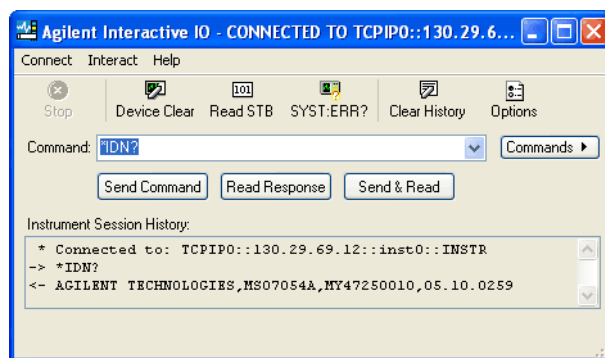


- iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

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

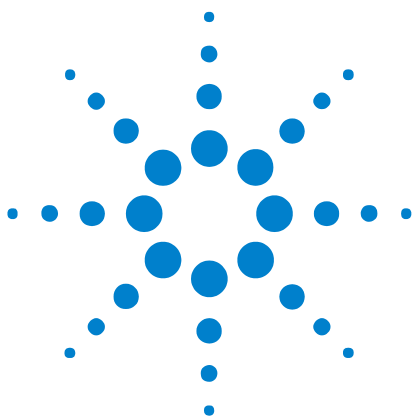


- b In the Agilent Interactive IO application, enter commands in the **Command** field and press **Send Command**, **Read Response**, or **Send&Read**.



- c Choose **Connect>Exit** from the menu to exit the Agilent Interactive IO application.
- 4 In the Agilent Connection Expert application, choose **File>Exit** from the menu to exit the application.





## 3 Getting Started

Basic Oscilloscope Program Structure 48

Programming the Oscilloscope 50

Other Ways of Sending Commands 59

This chapter gives you an overview of programming the 6000 Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

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

### NOTE

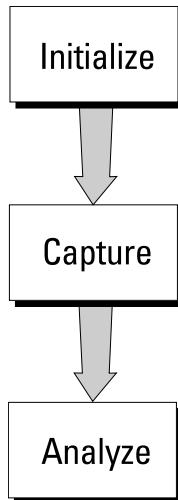
#### Language for Program Examples

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



## Basic Oscilloscope Program Structure

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



### Initializing

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

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

### Capturing Data

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



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

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

## Analyzing Captured Data

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

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

## Programming the Oscilloscope

- "Referencing the IO Library" on page 50
- "Opening the Oscilloscope Connection via the IO Library" on page 51
- "Using :AUToscale to Automate Oscilloscope Setup" on page 52
- "Using Other Oscilloscope Setup Commands" on page 52
- "Capturing Data with the :DIGitize Command" on page 53
- "Reading Query Responses from the Oscilloscope" on page 55
- "Reading Query Results into String Variables" on page 56
- "Reading Query Results into Numeric Variables" on page 56
- "Reading Definite-Length Block Query Response Data" on page 56
- "Sending Multiple Queries and Reading Results" on page 57
- "Checking Instrument Status" on page 58

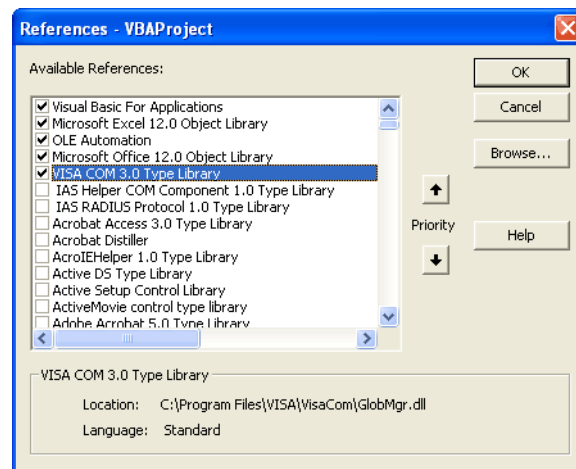
## Referencing the IO Library

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 677.

## Initializing the Interface and the Oscilloscope

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

```

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

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

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

' Clear the interface buffer.
myScope.IO.Clear

```

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

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

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

#### NOTE

#### Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in "[Common \(\\*\) Commands](#)" on page 109.

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 NORMal"
myScope.WriteString ":TIMEbase:RANGe 1E-3"
myScope.WriteString ":TIMEbase:DELay 100E-6"

```

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

### Example Oscilloscope Setup Code

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

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear

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

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

' Set the vertical range to 1.6 volts full scale with center screen
' at -0.4 volts with 10:1 probe attenuation and DC coupling.
myScope.WriteString ":CHANnel1:PROBe 10" ' Probe attenuation
' to 10:1.
myScope.WriteString ":CHANnel1:RANGe 1.6" ' Vertical range
' 1.6 V full scale.
myScope.WriteString ":CHANnel1:OFFSet -.4" ' Offset to -0.4.
myScope.WriteString ":CHANnel1:COUPling DC" ' Coupling to DC.

' Configure the instrument to trigger at -0.4 volts with normal
' triggering.
myScope.WriteString ":TRIGger:SWEep NORMal" ' Normal triggering.
myScope.WriteString ":TRIGger:LEVel -.4" ' Trigger level to -0.4.
myScope.WriteString ":TRIGger:SLOPe POSitive" ' Trigger on pos. slope.

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

### Capturing Data with the :DIGitize Command

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

**NOTE****Ensure New Data is Collected**

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

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

**NOTE****Set :TIMEbase:MODE to NORMal when using :DIGitize**

:TIMEbase:MODE must be set to NORMal 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 DELayed. Sending the \*RST (reset) command will also set the time base mode to normal.

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

The following program example shows a typical setup:

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

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

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

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

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

For more information, see the waveform subsystem commands and corresponding program code examples in [":WAVEform Commands"](#) on page 523.

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

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

**Reading Query Responses from the Oscilloscope**

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

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

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

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

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

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

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

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

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

## Reading Query Results into String Variables

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific command descriptions in ["Commands by Subsystem"](#) on page 107 for the formats and types of data returned from queries.

### NOTE

#### Express String Variables Using Exact Syntax

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

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

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

After running this program, the controller displays:

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

## Reading Query Results into Numeric Variables

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

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

After running this program, the controller displays:

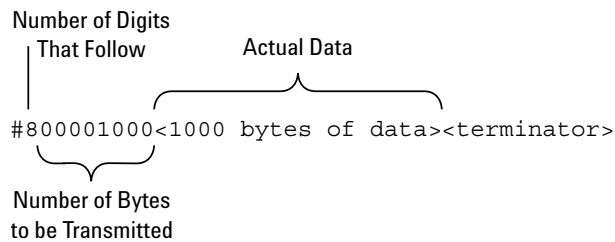
**Range (variant): 40**

## Reading Definite-Length Block Query Response Data

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

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





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

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

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

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

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

## Sending Multiple Queries and Reading Results

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

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

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

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

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

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

```
myScope.WriteString ":TIMEbase:RANGe?;DELaY?"
Dim strResults() As String
```

```
strResults() = myScope.ReadList(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 ["Status Reporting"](#) on page 643 which explains how to check the status of the instrument.

## Other Ways of Sending Commands

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

### Telnet Sockets

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

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

```
telnet <hostname> 5024
```

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

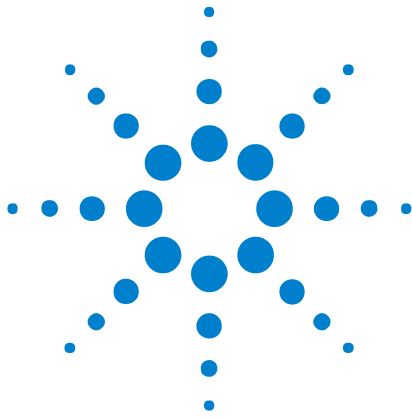
For a command line without a prompt, use port 5025. For example:

```
telnet <hostname> 5025
```

### Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the *6000 Series Oscilloscopes User's Guide*. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the **Browser Web Control** tab, then select the **Remote Programming** link.





## 4 Commands Quick Reference

Command Summary [62](#)

Syntax Elements [104](#)



## Command Summary

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

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 113</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 114</a> )	*ESE? (see <a href="#">page 115</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 116</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 116</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 119</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 120</a> )	*OPC? (see <a href="#">page 120</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 121</a> )	<p>&lt;return_value&gt; ::= 0,0,&lt;license info&gt;          &lt;license info&gt; ::= &lt;All field&gt;,&lt;reserved&gt;,&lt;Factory MSO&gt;,&lt;Upgraded MSO&gt;,&lt;Xilinx FPGA Probe&gt;,&lt;Memory&gt;,&lt;Low Speed Serial&gt;,&lt;Automotive Serial&gt;,&lt;reserved&gt;,&lt;Secure&gt;,&lt;Battery&gt;,&lt;Altera FPGA Probe&gt;,&lt;FlexRay Serial&gt;,&lt;reserved&gt;,&lt;RS-232/UART Serial&gt;,&lt;reserved&gt;          &lt;All field&gt; ::= {0   All}          &lt;reserved&gt; ::= 0          &lt;Factory MSO&gt; ::= {0   MSO}          &lt;Upgraded MSO&gt; ::= {0   MSO}          &lt;Xilinx FPGA Probe&gt; ::= {0   FPG}          &lt;Memory&gt; ::= {0   mem2M   mem8M}          &lt;Low Speed Serial&gt; ::= {0   LSS}          &lt;Automotive Serial&gt; ::= {0   AMS}          &lt;Secure&gt; ::= {0   SEC}          &lt;Battery&gt; ::= {0   BAT}          &lt;Altera FPGA Probe&gt; ::= {0   ALT}          &lt;FlexRay Serial&gt; ::= {0   FRS}          &lt;RS-232/UART Serial&gt; ::= {0   232}</p>
*RCL <value> (see <a href="#">page 122</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*RST (see <a href="#">page 123</a> )	n/a	See *RST (Reset) (see <a href="#">page 123</a> )
*SAV <value> (see <a href="#">page 126</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*SRE <mask> (see <a href="#">page 127</a> )	*SRE? (see <a href="#">page 128</a> )	<p>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= following values:</p> <p>Bit Weight Name Enables</p> <pre> ----- 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 129</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 131</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 132</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 133</a> )	n/a	n/a

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

Command	Query	Options and Query Returns
:ACTivity (see <a href="#">page 137</a> )	:ACTivity? (see <a href="#">page 137</a> )	<p>&lt;return value&gt; ::=</p> <p>&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 138</a> )	{0   1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see <a href="#">page 139</a> )	n/a	<p>&lt;source&gt; ::= CHANnel&lt;n&gt; for DSO models</p> <p>&lt;source&gt; ::= {CHANnel&lt;n&gt;   DIGital0,...,DIGital15   POD1   POD2} for MSO models</p> <p>&lt;source&gt; can be repeated up to 5 times</p> <p>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</p>



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

Command	Query	Options and Query Returns
:AUToscale:AMODE <value> (see <a href="#">page 141</a> )	:AUToscale:AMODE? (see <a href="#">page 141</a> )	<value> ::= {NORMAL   CURRENT}}
:AUToscale:CHANnels <value> (see <a href="#">page 142</a> )	:AUToscale:CHANnels? (see <a href="#">page 142</a> )	<value> ::= {ALL   DISPLAYed}}
:BLANK [<source>] (see <a href="#">page 143</a> )	n/a	<source> ::= {CHANnel<n>   FUNCTION   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see <a href="#">page 144</a> )	n/a	n/a
:DIGitize [<source>[,...,<source>]] (see <a href="#">page 145</a> )	n/a	<source> ::= {CHANnel<n>   FUNCTION   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n> (see <a href="#">page 147</a> )	:HWEenable? (see <a href="#">page 147</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister:CONDiti on? (see <a href="#">page 149</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENT]? (see <a href="#">page 151</a> )	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see <a href="#">page 153</a> )	n/a	<pixel memory> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:OPEE <n> (see <a href="#">page 154</a> )	:OPEE? (see <a href="#">page 155</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister:CONDiti on? (see <a href="#">page 156</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister[:EVENT]? (see <a href="#">page 158</a> )	<n> ::= 16-bit integer in NR1 format

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

Command	Query	Options and Query Returns
:OVLenable <mask> (see <a href="#">page 160</a> )	:OVLenable? (see <a href="#">page 161</a> )	<p>&lt;mask&gt; ::= 16-bit integer in NR1 format as shown:</p> <p>Bit Weight Input</p> <pre> ---  ----- 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 </pre>
n/a	:OVLRegister? (see <a href="#">page 162</a> )	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>] (see <a href="#">page 164</a> )	n/a	<p>&lt;options&gt; ::= [&lt;print option&gt;][,...,&lt;print option&gt;]</p> <p>&lt;print option&gt; ::= {COLor   GRAYscale   PRINter0   BMP8bit   BMP   PNG   NOFactors   FACTors}</p> <p>&lt;print option&gt; can be repeated up to 5 times.</p>
:RUN (see <a href="#">page 165</a> )	n/a	n/a
n/a	:SERial (see <a href="#">page 166</a> )	<return value> ::= unquoted string containing serial number
:SINGle (see <a href="#">page 167</a> )	n/a	n/a
n/a	:STATus? <display> (see <a href="#">page 168</a> )	<p>{0   1}</p> <p>&lt;display&gt; ::= {CHANnel&lt;n&gt;   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS}</p> <p>&lt;n&gt; ::= 1-2 or 1-4 in NR1 format</p>
:STOP (see <a href="#">page 169</a> )	n/a	n/a

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

Command	Query	Options and Query Returns
n/a	:TER? (see <a href="#">page 170</a> )	{0   1}
:VIEW <source> (see <a href="#">page 171</a> )	n/a	<source> ::= {CHANnel<n>   PMEMory{0   1   2   3   4   5   6   7   8   9}   FUNction   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   PMEMory{0   1   2   3   4   5   6   7   8   9}   POD{1   2}   BUS{1   2}   FUNction   MATH   SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

**Table 4** :ACQUIRE Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQUIRE:AALias? (see <a href="#">page 174</a> )	{1   0}
:ACQUIRE:COMPLete <complete> (see <a href="#">page 175</a> )	:ACQUIRE:COMPLete? (see <a href="#">page 175</a> )	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNT <count> (see <a href="#">page 176</a> )	:ACQUIRE:COUNT? (see <a href="#">page 176</a> )	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQUIRE:DAALias <mode> (see <a href="#">page 177</a> )	:ACQUIRE:DAALias? (see <a href="#">page 177</a> )	<mode> ::= {DISable   AUTO}
:ACQUIRE:MODE <mode> (see <a href="#">page 178</a> )	:ACQUIRE:MODE? (see <a href="#">page 178</a> )	<mode> ::= {RTIME   ETIME   SEGmented}
n/a	:ACQUIRE:POINTs? (see <a href="#">page 179</a> )	<# points> ::= an integer in NR1 format
:ACQUIRE:RSIGNAL <ref_signal_mode> (see <a href="#">page 180</a> )	:ACQUIRE:RSIGNAL? (see <a href="#">page 180</a> )	<ref_signal_mode> ::= {OFF   OUT   IN}
:ACQUIRE:SEGmented:CO UNT <count> (see <a href="#">page 181</a> )	:ACQUIRE:SEGmented:CO UNT? (see <a href="#">page 181</a> )	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
:ACQUIRE:SEGmented:IN Dex <index> (see <a href="#">page 182</a> )	:ACQUIRE:SEGmented:IN Dex? (see <a href="#">page 182</a> )	<index> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)

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

Command	Query	Options and Query Returns
n/a	:ACQuire:SRATe? (see <a href="#">page 184</a> )	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see <a href="#">page 185</a> )	:ACQuire:TYPE? (see <a href="#">page 185</a> )	<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 <a href="#">page 189</a> )	:BUS<n>:BIT<m>? (see <a href="#">page 189</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0   OFF}   {1   ON}} (see <a href="#">page 190</a> )	:BUS<n>:BITS? (see <a href="#">page 190</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> ::= 0-15; an integer in NR1 format
:BUS<n>:CLEAr (see <a href="#">page 192</a> )	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 193</a> )	:BUS<n>:DISPlay? (see <a href="#">page 193</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format

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

Command	Query	Options and Query Returns
:BUS<n>:LABel <string> (see <a href="#">page 194</a> )	:BUS<n>:LABel? (see <a href="#">page 194</a> )	<string> ::= quoted ASCII string up to 16 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see <a href="#">page 195</a> )	:BUS<n>:MASK? (see <a href="#">page 195</a> )	<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 <a href="#">page 197</a> )	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string> (see <a href="#">page 198</a> )	:CALibrate:LABel? (see <a href="#">page 198</a> )	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:STARt (see <a href="#">page 199</a> )	n/a	n/a
n/a	:CALibrate:STATus? (see <a href="#">page 200</a> )	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITCh? (see <a href="#">page 201</a> )	{PROtected   UNPRotected}

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

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPerature? (see <a href="#">page 202</a> )	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see <a href="#">page 203</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 207</a> )	:CHANnel<n>:BWLimit? (see <a href="#">page 207</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUpling <coupling> (see <a href="#">page 208</a> )	:CHANnel<n>:COUpling? (see <a href="#">page 208</a> )	<coupling> ::= {AC   DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay { {0   OFF}   {1   ON} } (see <a href="#">page 209</a> )	:CHANnel<n>:DISPlay? (see <a href="#">page 209</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see <a href="#">page 210</a> )	:CHANnel<n>:IMPedance? (see <a href="#">page 210</a> )	<impedance> ::= {ONEMeg   FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert { {0   OFF}   {1   ON} } (see <a href="#">page 211</a> )	:CHANnel<n>:INVert? (see <a href="#">page 211</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see <a href="#">page 212</a> )	:CHANnel<n>:LABel? (see <a href="#">page 212</a> )	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see <a href="#">page 213</a> )	:CHANnel<n>:OFFSet? (see <a href="#">page 213</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 214</a> )	:CHANnel<n>:PROBe? (see <a href="#">page 214</a> )	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see <a href="#">page 215</a> )	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format

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

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:SKEW <skew_value> (see <a href="#">page 216</a> )	:CHANnel<n>:PROBe:SKEW? (see <a href="#">page 216</a> )	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see <a href="#">page 217</a> )	:CHANnel<n>:PROBe:STYPe? (see <a href="#">page 217</a> )	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see <a href="#">page 218</a> )	:CHANnel<n>:PROTection? (see <a href="#">page 218</a> )	{NORM   TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see <a href="#">page 219</a> )	:CHANnel<n>:RANGe? (see <a href="#">page 219</a> )	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see <a href="#">page 220</a> )	:CHANnel<n>:SCALe? (see <a href="#">page 220</a> )	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see <a href="#">page 221</a> )	:CHANnel<n>:UNITs? (see <a href="#">page 221</a> )	<units> ::= {VOLT   AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 222</a> )	:CHANnel<n>:VERNier? (see <a href="#">page 222</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format

**Table 8** :DIGital<n> Commands Summary

Command	Query	Options and Query Returns
:DIGital<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 225</a> )	:DIGital<n>:DISPlay? (see <a href="#">page 225</a> )	{0   1} <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:LABel <string> (see <a href="#">page 226</a> )	:DIGital<n>:LABel? (see <a href="#">page 226</a> )	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:POSition <position> (see <a href="#">page 227</a> )	:DIGital<n>:POSition? (see <a href="#">page 227</a> )	<n> ::= 0-15; an integer in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small

**Table 8** :DIGital<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGital<n>:SIZE <value> (see page 228)	:DIGital<n>:SIZE? (see page 228)	<value> ::= {SMALl   MEDium   LARGE}
:DIGital<n>:THReshold <value>[suffix] (see page 229)	:DIGital<n>:THReshold ? (see page 229)	<n> ::= 0-15; an integer in NR1 format <value> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format from -8.00 to +8.00 [suffix] ::= {V   mV   uV}

**Table 9** :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLear (see page 232)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>] [,][<palette>]<displa y data> (see page 233)	:DISPlay:DATA? [<format>][,][<area>] [,][<palette>] (see page 233)	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF   BMP   BMP8bit   PNG} (query) <area> ::= {GRATicule   SCReen} (query) <palette> ::= {MONochrome   GRAYscale   COLor} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0   OFF}   {1   ON}} (see page 235)	:DISPlay:LABel? (see page 235)	{0   1}
:DISPlay:LABList <binary block> (see page 236)	:DISPlay:LABList? (see page 236)	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see page 237)	:DISPlay:PERsistence? (see page 237)	<value> ::= {MINimum   INFinite}}



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

Command	Query	Options and Query Returns
:DISPlay:SOURce <value> (see page 238)	:DISPlay:SOURce? (see page 238)	<value> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:DISPlay:VECTors {{1   ON}   {0   OFF}} (see page 239)	:DISPlay:VECTors? (see page 239)	{1   0}

**Table 10** :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see page 242)	:EXternal:BWLimit? (see page 242)	<bwlimit> ::= {0   OFF}
:EXternal:IMPedance <value> (see page 243)	:EXternal:IMPedance? (see page 243)	<impedance> ::= {ONEMeg   FIFTy}
:EXternal:PROBe <attenuation> (see page 244)	:EXternal:PROBe? (see page 244)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXternal:PROBe:ID? (see page 245)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXternal:PROBe:SType <signal type> (see page 246)	:EXternal:PROBe:SType ? (see page 246)	<signal type> ::= {DIFFerential   SINGle}
:EXternal:PROTection[ :CLEar] (see page 247)	:EXternal:PROTection? (see page 247)	{NORM   TRIP}
:EXternal:RANGe <range>[<suffix>] (see page 248)	:EXternal:RANGe? (see page 248)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXternal:UNITs <units> (see page 249)	:EXternal:UNITs? (see page 249)	<units> ::= {VOLT   AMPere}

**Table 11** :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTer <frequency> (see page 253)	:FUNCTION:CENTer? (see page 253)	<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 254)	:FUNCTION:DISPlay? (see page 254)	{0   1}
:FUNCTION:GOFT:OPERat ion <operation> (see page 255)	:FUNCTION:GOFT:OPERat ion? (see page 255)	<operation> ::= {ADD   SUBTract   MULTiply}
:FUNCTION:GOFT:SOURce 1 <source> (see page 256)	:FUNCTION:GOFT:SOURce 1? (see page 256)	<source> ::= CHANnel<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURce 2 <source> (see page 257)	:FUNCTION:GOFT:SOURce 2? (see page 257)	<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 258)	:FUNCTION:OFFSet? (see page 258)	<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 259)	:FUNCTION:OPERation? (see page 259)	<operation> ::= {ADD   SUBTract   MULTiply   INTegrate   DIFFerentiate   FFT   SQRT}
:FUNCTION:RANGe <range> (see page 260)	:FUNCTION:RANGe? (see page 260)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.

**Table 11** :FUNCTION Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCTION:REFERENCE <level> (see <a href="#">page 261</a> )	:FUNCTION:REFERENCE? (see <a href="#">page 261</a> )	<level> ::= the current reference level in NR3 format. The range of legal values is from 400.0 dBV to +400.0 dBV (depending on current range value).
:FUNCTION:SCALE <scale value>[<suffix>] (see <a href="#">page 262</a> )	:FUNCTION:SCALE? (see <a href="#">page 262</a> )	<scale value> ::= integer in NR1 format <suffix> ::= {V   dB}
:FUNCTION:SOURCE1 <source> (see <a href="#">page 263</a> )	:FUNCTION:SOURCE1? (see <a href="#">page 263</a> )	<source> ::= {CHANNEL<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models GOFT is only for FFT, INTEGRATE, DIFFERENTIATE, and SQRT operations.
:FUNCTION:SOURCE2 <source> (see <a href="#">page 264</a> )	:FUNCTION:SOURCE2? (see <a href="#">page 264</a> )	<source> ::= {CHANNEL<n>   NONE} <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURCE1 selection <n> ::= {1   2} for 2ch models
:FUNCTION:SPAN <span> (see <a href="#">page 265</a> )	:FUNCTION:SPAN? (see <a href="#">page 265</a> )	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCTION:WINDOW <window> (see <a href="#">page 266</a> )	:FUNCTION:WINDOW? (see <a href="#">page 266</a> )	<window> ::= {RECTANGULAR   HANNING   FLATTOP   BHARRIS}

**Table 12** :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see <a href="#">page 269</a> )	:HARDcopy:AREA? (see <a href="#">page 269</a> )	<area> ::= SCREEN
:HARDcopy:APRINTER <active_printer> (see <a href="#">page 270</a> )	:HARDcopy:APRINTER? (see <a href="#">page 270</a> )	<active_printer> ::= {<index>   <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list

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

Command	Query	Options and Query Returns
:HARDcopy:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 271</a> )	:HARDcopy:FACTors? (see <a href="#">page 271</a> )	{0   1}
:HARDcopy:FFEd {{0   OFF}   {1   ON}} (see <a href="#">page 272</a> )	:HARDcopy:FFEd? (see <a href="#">page 272</a> )	{0   1}
:HARDcopy:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 273</a> )	:HARDcopy:INKSaver? (see <a href="#">page 273</a> )	{0   1}
:HARDcopy:PALette <palette> (see <a href="#">page 274</a> )	:HARDcopy:PALette? (see <a href="#">page 274</a> )	<palette> ::= {COLor   GRAYscale   NONE}
n/a	:HARDcopy:PRinter:LIS T? (see <a href="#">page 275</a> )	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y   N} <name> ::= name of printer
:HARDcopy:START (see <a href="#">page 276</a> )	n/a	n/a

**Table 13** :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see <a href="#">page 279</a> )	:MARKer:MODE? (see <a href="#">page 279</a> )	<mode> ::= {OFF   MEASurement   MANUAL}
:MARKer:X1Position <position>[suffix] (see <a href="#">page 280</a> )	:MARKer:X1Position? (see <a href="#">page 280</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 281</a> )	:MARKer:X1Y1source? (see <a href="#">page 281</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>

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

Command	Query	Options and Query Returns
:MARKer:X2Position <position>[suffix] (see <a href="#">page 282</a> )	:MARKer:X2Position? (see <a href="#">page 282</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 283</a> )	:MARKer:X2Y2source? (see <a href="#">page 283</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see <a href="#">page 284</a> )	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see <a href="#">page 285</a> )	:MARKer:Y1Position? (see <a href="#">page 285</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 286</a> )	:MARKer:Y2Position? (see <a href="#">page 286</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 287</a> )	<return_value> ::= Y cursors delta value in NR3 format

**Table 14** :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see <a href="#">page 295</a> )	n/a	n/a
:MEASure:COUNter [<source>] (see <a href="#">page 296</a> )	:MEASure:COUNter? [<source>] (see <a href="#">page 296</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXTERNAL} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format

**Table 14** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DEFine DElay, <delay spec> (see <a href="#">page 297</a> )	:MEASure:DEFine? DElay (see <a href="#">page 298</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 297</a> )	:MEASure:DEFine? THResholds (see <a href="#">page 298</a> )	<threshold spec> ::= {STANdard}   {<threshold mode>,<upper>, <middle>,<lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DElay [<source1>] [,<source2>] (see <a href="#">page 300</a> )	:MEASure:DElay? [<source1>] [,<source2>] (see <a href="#">page 300</a> )	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see <a href="#">page 302</a> )	:MEASure:DUTYcycle? [<source>] (see <a href="#">page 302</a> )	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see <a href="#">page 303</a> )	:MEASure:FALLtime? [<source>] (see <a href="#">page 303</a> )	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format

**Table 14** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FREQuency [<source>] (see page 304)	:MEASure:FREQuency? [<source>] (see page 304)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 305)	:MEASure:NWIDth? [<source>] (see page 305)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 306)	:MEASure:OVERshoot? [<source>] (see page 306)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 308)	:MEASure:PERiod? [<source>] (see page 308)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 309)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 309)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 310)	:MEASure:PREShoot? [<source>] (see page 310)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format

**Table 14** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PWIDth [<source>] (see page 311)	:MEASure:PWIDth? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISEtime [<source>] (see page 312)	:MEASure:RISEtime? [<source>] (see page 312)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEviation [<source>] (see page 313)	:MEASure:SDEviation? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1   ON} (see page 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   EXtErnal} for DSO models <source1,2> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH   EXtErnal} for MSO models <n> ::= 1-2 or 1-4 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} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition



**Table 14** :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. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPlitude [<source>] (see page 321)	:MEASure:VAMPlitude? [<source>] (see page 321)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<source>] (see page 322)	:MEASure:VAverage? [<source>] (see page 322)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 323)	:MEASure:VBASe? [<source>] (see page 323)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 324)	:MEASure:VMAX? [<source>] (see page 324)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 325)	:MEASure:VMIN? [<source>] (see page 325)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format

**Table 14** :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} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 309)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 327)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 328)	:MEASure:VRMS? [<source>] (see page 328)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTime? <vtime>[,<source>] (see page 329)	<vtime> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see page 330)	:MEASure:VTOP? [<source>] (see page 330)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:XMAX [<source>] (see page 331)	:MEASure:XMAX? [<source>] (see page 331)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 332)	:MEASure:XMIN? [<source>] (see page 332)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

**Table 15** :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 334</a> )	:POD<n>:DISPlay? (see <a href="#">page 334</a> )	{0   1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see <a href="#">page 335</a> )	:POD<n>:SIZE? (see <a href="#">page 335</a> )	<value> ::= {SMALl   MEDium   LARGe}
:POD<n>:THReshold <type>[suffix] (see <a href="#">page 336</a> )	:POD<n>:THReshold? (see <a href="#">page 336</a> )	<n> ::= 1-2 in NR1 format <type> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V   mV   uV }

**Table 16** :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FIleName <base_name> (see <a href="#">page 339</a> )	:RECall:FIleName? (see <a href="#">page 339</a> )	<base_name> ::= quoted ASCII string
:RECall:IMAGe[:START] [<file_spec>] (see <a href="#">page 340</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
n/a	:RECall:PWD? (see <a href="#">page 341</a> )	<path_info> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see <a href="#">page 342</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string

**Table 17** :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILEname <base_name> (see <a href="#">page 345</a> )	:SAVE:FILEname? (see <a href="#">page 345</a> )	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:START] [<file_spec>] (see <a href="#">page 346</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:IMAGe:AREA <area> (see <a href="#">page 347</a> )	:SAVE:IMAGe:AREA? (see <a href="#">page 347</a> )	<area> ::= {GRATicule   SCReen}
:SAVE:IMAGe:FACTors {{0   OFF}   {1   ON}} (see <a href="#">page 348</a> )	:SAVE:IMAGe:FACTors? (see <a href="#">page 348</a> )	{0   1}
:SAVE:IMAGe:FORMat <format> (see <a href="#">page 349</a> )	:SAVE:IMAGe:FORMat? (see <a href="#">page 349</a> )	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGe:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 350</a> )	:SAVE:IMAGe:INKSaver? (see <a href="#">page 350</a> )	{0   1}
:SAVE:IMAGe:PALette <palette> (see <a href="#">page 351</a> )	:SAVE:IMAGe:PALette? (see <a href="#">page 351</a> )	<palette> ::= {COLor   GRAYscale   MONochrome}
n/a	:SAVE:PWD? (see <a href="#">page 352</a> )	<path_info> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see <a href="#">page 353</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVEform[:START] [<file_name>] (see <a href="#">page 354</a> )	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVEform:FORMat <format> (see <a href="#">page 355</a> )	:SAVE:WAVEform:FORMat? (see <a href="#">page 355</a> )	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVEform:LENGth <length> (see <a href="#">page 356</a> )	:SAVE:WAVEform:LENGth? (see <a href="#">page 356</a> )	<length> ::= 100 to max. length; an integer in NR1 format

**Table 18** :SBUS Commands Summary

Command	Query	Options and Query Returns
:SBUS:BUSDoctor:ADDRe ss <value> (see page 360)	:SBUS:BUSDoctor:ADDRe ss? (see page 360)	<value> ::= <field value>, <field value>, <field value>, <field value> <field value> ::= integer from 0-255 in NR1 format
:SBUS:BUSDoctor:BAUDr ate <baudrate> (see page 361)	:SBUS:BUSDoctor:BAUDr ate? (see page 361)	<baudrate> ::= {2500000   5000000   10000000}
:SBUS:BUSDoctor:CHANn el <channel> (see page 362)	:SBUS:BUSDoctor:CHANn el? (see page 362)	<channel> ::= {A   B}
:SBUS:BUSDoctor:MODE <mode> (see page 363)	:SBUS:BUSDoctor:MODE? (see page 363)	<mode> ::= {ASYNchronous   SYNchronous   PC}
n/a	:SBUS:CAN:COUNT:ERRor ? (see page 364)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:OVERl oad? (see page 365)	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see page 366)	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal ? (see page 367)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILi zation? (see page 368)	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0   OFF}   {1   ON}} (see page 369)	:SBUS:DISPlay? (see page 369)	{0   1}
n/a	:SBUS:FLEXray:COUNT:N ULL? (see page 370)	<frame_count> ::= integer in NR1 format
:SBUS:FLEXray:COUNT:R ESet (see page 371)	n/a	n/a
n/a	:SBUS:FLEXray:COUNT:S YNC? (see page 372)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:FLEXray:COUNT:T OTal? (see page 373)	<frame_count> ::= integer in NR1 format
:SBUS:IIC:ASize <size> (see page 374)	:SBUS:IIC:ASize? (see page 374)	<size> ::= {BIT7   BIT8}

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

Command	Query	Options and Query Returns
:SBUS:LIN:PARity {{0   OFF}   {1   ON}} (see <a href="#">page 375</a> )	:SBUS:LIN:PARity? (see <a href="#">page 375</a> )	{0   1}
:SBUS:MODE <mode> (see <a href="#">page 376</a> )	:SBUS:MODE? (see <a href="#">page 376</a> )	<mode> ::= {IIC   SPI   CAN   LIN   FLEXray   UART}
:SBUS:SPI:WIDTh <word_width> (see <a href="#">page 377</a> )	:SBUS:SPI:WIDTh? (see <a href="#">page 377</a> )	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see <a href="#">page 378</a> )	:SBUS:UART:BASE? (see <a href="#">page 378</a> )	<base> ::= {ASCIi   BINary   HEX}
n/a	:SBUS:UART:COUNT:ERROr? (see <a href="#">page 379</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESeT (see <a href="#">page 380</a> )	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFRames? (see <a href="#">page 381</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:UART:COUNT:TXFRames? (see <a href="#">page 382</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see <a href="#">page 383</a> )	:SBUS:UART:FRAMing? (see <a href="#">page 383</a> )	<value> ::= {OFF   <decimal>   <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary

**Table 19** :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see <a href="#">page 385</a> )	:SYSTem:DATE? (see <a href="#">page 385</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 386</a> )	n/a	<string> ::= up to 254 characters as a quoted ASCII string

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

Command	Query	Options and Query Returns
n/a	:SYSTem:ERRor? (see <a href="#">page 387</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 635</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 388</a> )	:SYSTem:LOCK? (see <a href="#">page 388</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PROTection:LOCK <value> (see <a href="#">page 389</a> )	:SYSTem:PROTection:LOCK? (see <a href="#">page 389</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 390</a> )	:SYSTem:SETup? (see <a href="#">page 390</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 392</a> )	:SYSTem:TIME? (see <a href="#">page 392</a> )	<time> ::= hours, minutes, seconds in NR1 format

**Table 20** :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see <a href="#">page 395</a> )	:TIMebase:MODE? (see <a href="#">page 395</a> )	<value> ::= {MAIN   WINDOW   XY   ROLL}
:TIMebase:POSition <pos> (see <a href="#">page 396</a> )	:TIMebase:POSition? (see <a href="#">page 396</a> )	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see <a href="#">page 397</a> )	:TIMebase:RANGe? (see <a href="#">page 397</a> )	<range_value> ::= 5 ns through 500 s in NR3 format
:TIMebase:REFClock {{0   OFF}   {1   ON}} (see <a href="#">page 398</a> )	:TIMebase:REFClock? (see <a href="#">page 398</a> )	{0   1}
:TIMebase:REFErence {LEFT   CENTER   RIGHT} (see <a href="#">page 399</a> )	:TIMebase:REFErence? (see <a href="#">page 399</a> )	<return_value> ::= {LEFT   CENTER   RIGHT}
:TIMebase:SCALE <scale_value> (see <a href="#">page 400</a> )	:TIMebase:SCALE? (see <a href="#">page 400</a> )	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 401</a> )	:TIMebase:VERNier? (see <a href="#">page 401</a> )	{0   1}

**Table 20** :TIMEbase Commands Summary (continued)

Command	Query	Options and Query Returns
:TIMEbase:WINDow:POSi tion <pos> (see page 402)	:TIMEbase:WINDow:POSi tion? (see page 402)	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMEbase:WINDow:RANG e <range_value> (see page 403)	:TIMEbase:WINDow:RANG e? (see page 403)	<range_value> ::= range value in seconds in NR3 format for the delayed window
:TIMEbase:WINDow:SCAL e <scale_value> (see page 404)	:TIMEbase:WINDow:SCAL e? (see page 404)	<scale_value> ::= scale value in seconds in NR3 format for the delayed window

**Table 21** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}}   {{1   ON}} (see page 409)	:TRIGger:HFReject? (see page 409)	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see page 410)	:TRIGger:HOLDoff? (see page 410)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:MODE <mode> (see page 411)	:TRIGger:MODE? (see page 411)	<mode> ::= {EDGE   GLITCh   PATtern   CAN   DURation   IIC   EBURst   LIN   SEquence   SPI   TV   USB   FLEXray} <return_value> ::= {<mode>   <none>} <none> ::= query returns "NONE" if the :TIMEbase:MODE is ROLL or XY
:TRIGger:NREJect {{0   OFF}}   {{1   ON}} (see page 412)	:TRIGger:NREJect? (see page 412)	{0   1}



**Table 21** General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:PATtern <value>, <mask> [,<edge source>,<edge>] (see page 413)	:TRIGger:PATtern? (see page 414)	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xn timer"; n ::= {0,...,9   A,...,F} (# bits = # channels) <edge source> ::= {CHANnel<n>   EXTERNAL   NONE} for DSO models <edge source> ::= {CHANnel<n>   DIGital0,...,DIGital15   NONE} for MSO models <edge> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see page 415)	:TRIGger:SWEep? (see page 415)	<sweep> ::= {AUTO   NORMal}

**Table 22** :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATtern: DATA <value>, <mask> (see page 418)	:TRIGger:CAN:PATtern: DATA? (see page 418)	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:CAN:PATtern: DATA:LENGth <length> (see page 419)	:TRIGger:CAN:PATtern: DATA:LENGth? (see page 419)	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)

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

Command	Query	Options and Query Returns
:TRIGger:CAN:PATtern: ID <value>, <mask> (see <a href="#">page 420</a> )	:TRIGger:CAN:PATtern: ID? (see <a href="#">page 420</a> )	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:CAN:PATtern: ID:MODE <value> (see <a href="#">page 421</a> )	:TRIGger:CAN:PATtern: ID:MODE? (see <a href="#">page 421</a> )	<value> ::= {STANDARD   EXTENDED} (with Option AMS)
:TRIGger:CAN:SAMPlepo int <value> (see <a href="#">page 422</a> )	:TRIGger:CAN:SAMPlepo int? (see <a href="#">page 422</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:CAN:SIGNAL:B AUDrate <baudrate> (see <a href="#">page 423</a> )	:TRIGger:CAN:SIGNAL:B AUDrate? (see <a href="#">page 423</a> )	<baudrate> ::= {10000   20000   33300   50000   62500   83300   100000   125000   250000   500000   800000   1000000}
:TRIGger:CAN:SOURce <source> (see <a href="#">page 424</a> )	:TRIGger:CAN:SOURce? (see <a href="#">page 424</a> )	<source> ::= {CHANNEL<n>   EXTERNAL} for DSO models <source> ::= {CHANNEL<n>   DIGital0,...,DIGital15  } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see <a href="#">page 425</a> )	:TRIGger:CAN:TRIGger? (see <a href="#">page 426</a> )	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF   DATA   ERROR   IDData   IDEither   IDRemote   ALLerrors   OVERload   ACKerror} (with Option AMS)

**Table 23** :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see <a href="#">page 428</a> )	:TRIGger:DURation:GREaterthan? (see <a href="#">page 428</a> )	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see <a href="#">page 429</a> )	:TRIGger:DURation:LESSthan? (see <a href="#">page 429</a> )	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:Pattern <value>, <mask> (see <a href="#">page 430</a> )	:TRIGger:DURation:Pattern? (see <a href="#">page 430</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xxxxxxxx" n ::= {0,...,9   A,...,F}
:TRIGger:DURation:Qualifier <qualifier> (see <a href="#">page 431</a> )	:TRIGger:DURation:Qualifier? (see <a href="#">page 431</a> )	<qualifier> ::= {GREaterthan   LESSthan   INRange   OUTRange   TIMEOUT}
:TRIGger:DURation:Range <greater than time>[suffix], <less than time>[suffix] (see <a href="#">page 432</a> )	:TRIGger:DURation:Range? (see <a href="#">page 432</a> )	<greater than time> ::= min duration from 10 ns to 9.99 seconds in NR3 format <less than time> ::= max duration from 15 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}

**Table 24** :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNT <count> (see <a href="#">page 434</a> )	:TRIGger:EBURst:COUNT? (see <a href="#">page 434</a> )	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see <a href="#">page 435</a> )	:TRIGger:EBURst:IDLE? (see <a href="#">page 435</a> )	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see <a href="#">page 436</a> )	:TRIGger:EBURst:SLOPe? (see <a href="#">page 436</a> )	<slope> ::= {NEGative   POSitive}

**Table 25** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LF} (see <a href="#">page 438</a> )	:TRIGger[:EDGE]:COUPling? (see <a href="#">page 438</a> )	{AC   DC   LF}
:TRIGger[:EDGE]:LEVel <level> [, <source>] (see <a href="#">page 439</a> )	:TRIGger[:EDGE]:LEVel? [<source>] (see <a href="#">page 439</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. For digital channels (MSO models), <level> ::= 8 V. <source> ::= {CHANnel<n>   EXTErnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXTErnal} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LF   HF} (see <a href="#">page 440</a> )	:TRIGger[:EDGE]:REJect? (see <a href="#">page 440</a> )	{OFF   LF   HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see <a href="#">page 441</a> )	:TRIGger[:EDGE]:SLOPe? (see <a href="#">page 441</a> )	<polarity> ::= {POSitive   NEGative   EITHer   ALTErnate}
:TRIGger[:EDGE]:SOURCe <source> (see <a href="#">page 442</a> )	:TRIGger[:EDGE]:SOURCe? (see <a href="#">page 442</a> )	<source> ::= {CHANnel<n>   EXTErnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXTErnal} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

**Table 26** :TRIGger:FLEXray Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FLEXray:ERRor:TYPE <error_type> (see <a href="#">page 444</a> )	:TRIGger:FLEXray:ERRor:TYPE? (see <a href="#">page 444</a> )	<error_type> ::= {ALL   CODE   TSS   HCRC   FCRC   END   BOUNdary   IDLE   SYMbOl   SLOt   NULL   SOS   FID   CCounT   PLENgth}
:TRIGger:FLEXray:FRAMe:CCBase <cycle_count_base> (see <a href="#">page 446</a> )	:TRIGger:FLEXray:FRAMe:CCBase? (see <a href="#">page 446</a> )	<cycle_count_base> ::= integer from 0-63

**Table 26** :TRIGger:FLEXray Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:FLEXray:FRAME:CCRepetition <cycle_count_repetition> (see <a href="#">page 447</a> )	:TRIGger:FLEXray:FRAME:CCRepetition? (see <a href="#">page 447</a> )	<cycle_count_repetition> ::= {ALL   <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:FRAME:ID <frame_id> (see <a href="#">page 448</a> )	:TRIGger:FLEXray:FRAME:ID? (see <a href="#">page 448</a> )	<frame_id> ::= {ALL   <frame #>} <frame #> ::= integer from 1-2047
:TRIGger:FLEXray:FRAME:TYPE <frame_type> (see <a href="#">page 449</a> )	:TRIGger:FLEXray:FRAME:TYPE? (see <a href="#">page 449</a> )	<frame_type> ::= {NORMAL   STARTup   NULL   SYNC   NSTArtup   NNULl   NSYNc   ALL}
:TRIGger:FLEXray:TIME:CBASe <cycle_base> (see <a href="#">page 450</a> )	:TRIGger:FLEXray:TIME:CBASe? (see <a href="#">page 450</a> )	<cycle_base> ::= integer from 0-63
:TRIGger:FLEXray:TIME:CREPetition <cycle_repetition> (see <a href="#">page 451</a> )	:TRIGger:FLEXray:TIME:CREPetition? (see <a href="#">page 451</a> )	<cycle_repetition> ::= {ALL   <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:TIME:SEGMENT <segment_type> (see <a href="#">page 452</a> )	:TRIGger:FLEXray:TIME:SEGMENT? (see <a href="#">page 452</a> )	<segment_type> ::= {STATIC   DYNAMIC   SYMBOL   IDLE}
:TRIGger:FLEXray:TIME:SLOT <slot_type>, <slot_id> (see <a href="#">page 453</a> )	:TRIGger:FLEXray:TIME:SLOT? (see <a href="#">page 453</a> )	<slot_type> ::= {ALL   EMPTY} <slot_id> ::= {ALL   <slot #>} <slot #> ::= integer from 1-2047
:TRIGger:FLEXray:TRIGger <condition> (see <a href="#">page 454</a> )	:TRIGger:FLEXray:TRIGger? (see <a href="#">page 454</a> )	<condition> ::= {FRAME   TIME   ERROR}

**Table 27** :TRIGger:GLITCh Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITCh:GREAt erthan <greater than time>[suffix] (see <a href="#">page 457</a> )	:TRIGger:GLITCh:GREAt erthan? (see <a href="#">page 457</a> )	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITCh:LESSt han <less than time>[suffix] (see <a href="#">page 458</a> )	:TRIGger:GLITCh:LESSt han? (see <a href="#">page 458</a> )	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}

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

Command	Query	Options and Query Returns
:TRIGger:GLITch:LEVel <level> [<source>] (see <a href="#">page 459</a> )	:TRIGger:GLITch:LEVel ? (see <a href="#">page 459</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. For digital channels (MSO models), <level> ::= 6 V. <source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see <a href="#">page 460</a> )	:TRIGger:GLITch:POLarity? (see <a href="#">page 460</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see <a href="#">page 461</a> )	:TRIGger:GLITch:QUALifier? (see <a href="#">page 461</a> )	<qualifier> ::= {GREATERthan   LESSthan   RANGE}
:TRIGger:GLITch:RANGE <greater than time>[suffix], <less than time>[suffix] (see <a href="#">page 462</a> )	:TRIGger:GLITch:RANGE ? (see <a href="#">page 462</a> )	<greater than time> ::= start time from 10 ns to 9.99 seconds in NR3 format <less than time> ::= stop time from 15 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:SOURce <source> (see <a href="#">page 463</a> )	:TRIGger:GLITch:SOURce? (see <a href="#">page 463</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format

**Table 28** :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATtern:ADDRESS <value> (see <a href="#">page 465</a> )	:TRIGger:IIC:PATtern:ADDRESS? (see <a href="#">page 465</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATtern:DATA <value> (see <a href="#">page 466</a> )	:TRIGger:IIC:PATtern:DATA? (see <a href="#">page 466</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATtern:DATA2 <value> (see <a href="#">page 467</a> )	:TRIGger:IIC:PATtern:DATA2? (see <a href="#">page 467</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC[:SOURce]:CLOCK <source> (see <a href="#">page 468</a> )	:TRIGger:IIC[:SOURce]:CLOCK? (see <a href="#">page 468</a> )	<source> ::= {CHANnel<n>   EXTeRnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see <a href="#">page 469</a> )	:TRIGger:IIC[:SOURce]:DATA? (see <a href="#">page 469</a> )	<source> ::= {CHANnel<n>   EXTeRnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see <a href="#">page 470</a> )	:TRIGger:IIC:TRIGger:QUALifier? (see <a href="#">page 470</a> )	<value> ::= {EQUal   NOTequal   LESSthan   GREATERthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see <a href="#">page 471</a> )	:TRIGger:IIC:TRIGger[:TYPE]? (see <a href="#">page 471</a> )	<type> ::= {START   STOP   READ7   READEprom   WRITe7   WRITe10   NACKnowledge   ANACKnowledge   R7Data2   W7Data2   REStart}

**Table 29** :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see <a href="#">page 474</a> )	:TRIGger:LIN:ID? (see <a href="#">page 474</a> )	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:LIN:SAMplepo int <value> (see <a href="#">page 475</a> )	:TRIGger:LIN:SAMplepo int? (see <a href="#">page 475</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:LIN:SIGNAL:B AUDrate <baudrate> (see <a href="#">page 476</a> )	:TRIGger:LIN:SIGNAL:B AUDrate? (see <a href="#">page 476</a> )	<baudrate> ::= {2400   9600   19200}
:TRIGger:LIN:SOURce <source> (see <a href="#">page 477</a> )	:TRIGger:LIN:SOURce? (see <a href="#">page 477</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std> (see <a href="#">page 478</a> )	:TRIGger:LIN:STANdard ? (see <a href="#">page 478</a> )	<std> ::= {LIN13   LIN20}
:TRIGger:LIN:SYNCbrea k <value> (see <a href="#">page 479</a> )	:TRIGger:LIN:SYNCbrea k? (see <a href="#">page 479</a> )	<value> ::= integer = {11   12   13}
:TRIGger:LIN:TRIGger <condition> (see <a href="#">page 480</a> )	:TRIGger:LIN:TRIGger? (see <a href="#">page 480</a> )	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak   ID} (with Option AMS)



**Table 30** :TRIGger:SEquence Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SEquence:COU Nt <count> (see <a href="#">page 482</a> )	:TRIGger:SEquence:COU Nt? (see <a href="#">page 482</a> )	<count> ::= integer in NR1 format
:TRIGger:SEquence:EDG E{1 2} <source>, <slope> (see <a href="#">page 483</a> )	:TRIGger:SEquence:EDG E{1 2}? (see <a href="#">page 483</a> )	<source> ::= {CHANnel<n>   EXTErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <slope> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= query returns "NONE" if edge source is disabled
:TRIGger:SEquence:FIN D <value> (see <a href="#">page 484</a> )	:TRIGger:SEquence:FIN D? (see <a href="#">page 484</a> )	<value> ::= {PATTErn1,ENTERed   PATTErn1,EXITed   EDGE1   PATTErn1,AND,EDGE1}
:TRIGger:SEquence:PAT Tern{1 2} <value>, <mask> (see <a href="#">page 485</a> )	:TRIGger:SEquence:PAT Tern{1 2}? (see <a href="#">page 485</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn timer" n ::= {0,...,9   A,...,F}
:TRIGger:SEquence:RES et <value> (see <a href="#">page 486</a> )	:TRIGger:SEquence:RES et? (see <a href="#">page 486</a> )	<value> ::= {NONE   PATTErn1,ENTERed   PATTErn1,EXITed   EDGE1   PATTErn1,AND,EDGE1   PATTErn2,ENTERed   PATTErn2,EXITed   EDGE2   TIMer} Values used in find and trigger stages not available. EDGE2 not available if EDGE2,COUNT used in trigger stage.
:TRIGger:SEquence:TIM er <time_value> (see <a href="#">page 487</a> )	:TRIGger:SEquence:TIM er? (see <a href="#">page 487</a> )	<time_value> ::= time from 100 ns to 10 seconds in NR3 format
:TRIGger:SEquence:TRI Gger <value> (see <a href="#">page 488</a> )	:TRIGger:SEquence:TRI Gger? (see <a href="#">page 488</a> )	<value> ::= {PATTErn2,ENTERed   PATTErn2,EXITed   EDGE2   PATTErn2,AND,EDGE2   EDGE2,COUNT   EDGE2,COUNT,NREFind}

**Table 31** :TRIGger:SPI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCK:SL OPe <slope> (see page 490)	:TRIGger:SPI:CLOCK:SL OPe? (see page 490)	<slope> ::= {NEGative   POSitive}
:TRIGger:SPI:CLOCK:TI Meout <time_value> (see page 491)	:TRIGger:SPI:CLOCK:TI Meout? (see page 491)	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 492)	:TRIGger:SPI:FRAMing? (see page 492)	<value> ::= {CHIPselect   NOTChipselect   TIMEout}
:TRIGger:SPI:PATtern: DATA <value>, <mask> (see page 493)	:TRIGger:SPI:PATtern: DATA? (see page 493)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn timer" where n ::= {0,...,9   A,...,F}
:TRIGger:SPI:PATtern: WIDTh <width> (see page 494)	:TRIGger:SPI:PATtern: WIDTh? (see page 494)	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:C LOCK <source> (see page 495)	:TRIGger:SPI:SOURce:C LOCK? (see page 495)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:D ATA <source> (see page 496)	:TRIGger:SPI:SOURce:D ATA? (see page 496)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:F RAME <source> (see page 497)	:TRIGger:SPI:SOURce:F RAME? (see page 497)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format

**Table 32** :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see <a href="#">page 499</a> )	:TRIGger:TV:LINE? (see <a href="#">page 499</a> )	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see <a href="#">page 500</a> )	:TRIGger:TV:MODE? (see <a href="#">page 500</a> )	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LINE   VERTICAL   LFIELD1   LFIELD2   LALTERNATE   LVERTICAL}
:TRIGger:TV:POLarity <polarity> (see <a href="#">page 501</a> )	:TRIGger:TV:POLarity? (see <a href="#">page 501</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:TV:SOURce <source> (see <a href="#">page 502</a> )	:TRIGger:TV:SOURce? (see <a href="#">page 502</a> )	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard> (see <a href="#">page 503</a> )	:TRIGger:TV:STANdard? (see <a href="#">page 503</a> )	<standard> ::= {GENeric   NTSC   PALM   PAL   SECam   {P480L60HZ   P480}   {P720L60HZ   P720}   {P1080L24HZ   P1080}   P1080L25HZ   {I1080L50HZ   I1080}   I1080L60HZ}

**Table 33** :TRIGger:UART Commands Summary

Command	Query	Options and Query Returns
:TRIGger:UART:BAUDrate <baudrate> (see <a href="#">page 506</a> )	:TRIGger:UART:BAUDrate? (see <a href="#">page 506</a> )	<baudrate> ::= {1200   1800   2000   2400   3600   4800   7200   9600   14400   15200   19200   28800   38400   56000   57600   76800   115200   128000   230400   460800   921600   1382400   1843200   2764800}
:TRIGger:UART:BITorder <bitorder> (see <a href="#">page 507</a> )	:TRIGger:UART:BITorder? (see <a href="#">page 507</a> )	<bitorder> ::= {LSBFirst   MSBFirst}
:TRIGger:UART:BURSt <value> (see <a href="#">page 508</a> )	:TRIGger:UART:BURSt? (see <a href="#">page 508</a> )	<value> ::= {OFF   1 to 4096 in NR1 format}

**Table 33** :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:DATA <value> (see page 509)	:TRIGger:UART:DATA? (see page 509)	<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary
:TRIGger:UART:IDLE <time_value> (see page 510)	:TRIGger:UART:IDLE? (see page 510)	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 511)	:TRIGger:UART:PARity? (see page 511)	<parity> ::= {EVEN   ODD   NONE}
:TRIGger:UART:POLarity <polarity> (see page 512)	:TRIGger:UART:POLarity? (see page 512)	<polarity> ::= {HIGH   LOW}
:TRIGger:UART:QUALifier <value> (see page 513)	:TRIGger:UART:QUALifier? (see page 513)	<value> ::= {EQUAL   NOTequal   GREaterthan   LESSthan}
:TRIGger:UART:SOURce: RX <source> (see page 514)	:TRIGger:UART:SOURce: RX? (see page 514)	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:SOURce: TX <source> (see page 515)	:TRIGger:UART:SOURce: TX? (see page 515)	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see page 516)	:TRIGger:UART:TYPE? (see page 516)	<value> ::= {RSTArt   RSTOp   RDATA   RD1   RD0   RDX   PARityerror   TSTArt   TSTOp   TDATA   TD1   TD0   TDX}
:TRIGger:UART:WIDTH <width> (see page 517)	:TRIGger:UART:WIDTH? (see page 517)	<width> ::= {5   6   7   8   9}

**Table 34** :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:D MINus <source> (see page 519)	:TRIGger:USB:SOURce:D MINus? (see page 519)	<source> ::= {CHANnel<n>   EXTErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SOURce:D PLus <source> (see page 520)	:TRIGger:USB:SOURce:D PLus? (see page 520)	<source> ::= {CHANnel<n>   EXTErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SPEEd <value> (see page 521)	:TRIGger:USB:SPEEd? (see page 521)	<value> ::= {LOW   FULL}
:TRIGger:USB:TRIGger <value> (see page 522)	:TRIGger:USB:TRIGger? (see page 522)	<value> ::= {SOP   EOP   ENTersuspend   EXITsuspend   RESet}

**Table 35** :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see page 531)	:WAVEform:BYTeorder? (see page 531)	<value> ::= {LSBFirst   MSBFirst}
n/a	:WAVEform:COUNT? (see page 532)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see page 533)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVEform:FORMat <value> (see page 535)	:WAVEform:FORMat? (see page 535)	<value> ::= {WORD   BYTE   ASCII}

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

Command	Query	Options and Query Returns
:WAVeform:POINTs <# points> (see <a href="#">page 536</a> )	:WAVeform:POINTs? (see <a href="#">page 536</a> )	<# points> ::= {100   250   500   1000   <points_mode>} if waveform points mode is NORMAl <# points> ::= {100   250   500   1000   2000 ... 8000000 in 1-2-5 sequence   <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl   MAXimum   RAW}
:WAVeform:POINTs:MODE <points_mode> (see <a href="#">page 538</a> )	:WAVeform:POINTs:MODE ? (see <a href="#">page 539</a> )	<points_mode> ::= {NORMAl   MAXimum   RAW}
n/a	:WAVeform:PREamble? (see <a href="#">page 540</a> )	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: <ul style="list-style-type: none"> <li>• 0 for BYTE format</li> <li>• 1 for WORD format</li> <li>• 2 for ASCii format</li> </ul> <type> ::= an integer in NR1 format: <ul style="list-style-type: none"> <li>• 0 for NORMAl type</li> <li>• 1 for PEAK detect type</li> <li>• 2 for AVERAge type</li> <li>• 3 for HRESolution type</li> </ul> <count> ::= Average count, or 1 if PEAK detect type or NORMAl; an integer in NR1 format
n/a	:WAVeform:SEGmented:COUNT? (see <a href="#">page 543</a> )	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGmented:TAG? (see <a href="#">page 544</a> )	<time_tag> ::= in NR3 format (with Option SGM)

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

Command	Query	Options and Query Returns
:WAVeform:SOURce <source> (see <a href="#">page 545</a> )	:WAVeform:SOURce? (see <a href="#">page 545</a> )	<source> ::= {CHANnel<n>   FUNction   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   POD{1   2}   BUS{1   2}   FUNction   MATH   SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:WAVeform:SOURce:SUBS ource <subsource> (see <a href="#">page 549</a> )	:WAVeform:SOURce:SUBS ource? (see <a href="#">page 549</a> )	<subsource> ::= {{NONE   RX}   TX}
n/a	:WAVeform:TYPE? (see <a href="#">page 550</a> )	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVeform:UNSigned {{0   OFF}   {1   ON}} (see <a href="#">page 551</a> )	:WAVeform:UNSigned? (see <a href="#">page 551</a> )	{0   1}
:WAVeform:VIEW <view> (see <a href="#">page 552</a> )	:WAVeform:VIEW? (see <a href="#">page 552</a> )	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see <a href="#">page 553</a> )	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see <a href="#">page 554</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see <a href="#">page 555</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see <a href="#">page 556</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see <a href="#">page 557</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see <a href="#">page 558</a> )	<return_value> ::= y-reference value in the current preamble in NR1 format

## Syntax Elements

- "Number Format" on page 104
- "<NL> (Line Terminator)" on page 104
- "[ ] (Optional Syntax Terms)" on page 104
- "{ } (Braces)" on page 104
- "::<= (Defined As)" on page 104
- "< > (Angle Brackets)" on page 105
- "... (Ellipsis)" on page 105
- "n,...,p (Value Ranges)" on page 105
- "d (Digits)" on page 105
- "Quoted ASCII String" on page 105
- "Definite-Length Block Response Data" on page 105

### Number Format

NR1 specifies integer data.

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

### <NL> (Line Terminator)

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

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

### [ ] (Optional Syntax Terms)

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

### { } (Braces)

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

### ::= (Defined As)

::= means "defined as".



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

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

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

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

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

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

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

### **d (Digits)**

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

### **Quoted ASCII String**

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

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

has a quoted ASCII string of:

```
'One'
```

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

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

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

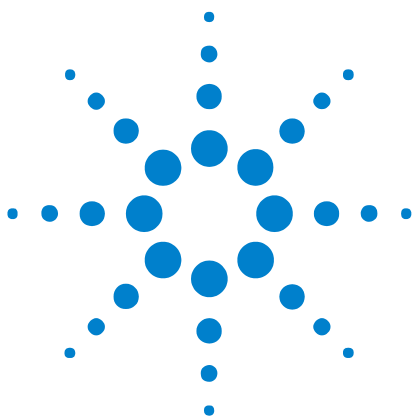
For example, for transmitting 1000 bytes of data, the syntax would be

#800001000<1000 bytes of data> <NL>

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

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

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



## 5 Commands by Subsystem

Subsystem	Description
"Common (*) Commands" on page 109	Commands defined by IEEE 488.2 standard that are common to all instruments.
"Root (:) Commands" on page 134	Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.
":ACQuire Commands" on page 172	Set the parameters for acquiring and storing data.
":BUS<n> Commands" on page 187	Control all oscilloscope functions associated with the digital channels bus display.
":CALibrate Commands" on page 196	Utility commands for determining the state of the calibration factor protection switch.
":CHANnel<n> Commands" on page 204	Control all oscilloscope functions associated with individual analog channels or groups of channels.
":DIGital<n> Commands" on page 223	Control all oscilloscope functions associated with individual digital channels.
":DISPlay Commands" on page 230	Control how waveforms, graticule, and text are displayed and written on the screen.
":EXTeRnal Trigger Commands" on page 240	Control the input characteristics of the external trigger input.
":FUNcTION Commands" on page 250	Control functions in the measurement/storage module.
":HARDcopy Commands" on page 267	Set and query the selection of hardcopy device and formatting options.
":MARKeR Commands" on page 277	Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
":MEASure Commands" on page 288	Select automatic measurements to be made and control time markers.



Subsystem	Description
<a href="#">":POD Commands" on page 333</a>	Control all oscilloscope functions associated with groups of digital channels.
<a href="#">":RECall Commands" on page 338</a>	Recall previously saved oscilloscope setups and traces.
<a href="#">":SAVE Commands" on page 343</a>	Save oscilloscope setups and traces, screen images, and data.
<a href="#">":SBUS Commands" on page 357</a>	Control oscilloscope functions associated with the serial decode bus.
<a href="#">":SYSTem Commands" on page 384</a>	Control basic system functions of the oscilloscope.
<a href="#">":TIMebase Commands" on page 393</a>	Control all horizontal sweep functions.
<a href="#">":TRIGger Commands" on page 405</a>	Control the trigger modes and parameters for each trigger type.
<a href="#">":WAVEform Commands" on page 523</a>	Provide access to waveform data.

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

- **Common (\*) Commands** – See ["Introduction to Common \(\\*\) Commands" on page 111](#) for more information.
- **Root Level (:) Commands** – See ["Introduction to Root \(:\) Commands" on page 136](#) for more information.
- **Subsystem Commands** – Subsystem commands are grouped together under a common node of the ["Command Tree" on page 681](#), such as the :TIMebase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree; therefore, no subsystem is selected.

## Common (\*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See ["Introduction to Common \(\\*\) Commands"](#) on page 111.

**Table 36** Common (\*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see <a href="#">page 113</a> )	n/a	n/a
*ESE <mask> (see <a href="#">page 114</a> )	*ESE? (see <a href="#">page 115</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 116</a> )	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see <a href="#">page 116</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 119</a> )	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see <a href="#">page 120</a> )	*OPC? (see <a href="#">page 120</a> )	ASCII "1" is placed in the output queue when all pending device operations have completed.

**Table 36** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see <a href="#">page 121</a> )	<p>&lt;return_value&gt; ::= 0,0,&lt;license info&gt;            &lt;license info&gt; ::= &lt;All field&gt;,&lt;reserved&gt;,&lt;Factory MSO&gt;,&lt;Upgraded MSO&gt;,&lt;Xilinx FPGA Probe&gt;,&lt;Memory&gt;,&lt;Low Speed Serial&gt;,&lt;Automotive Serial&gt;,&lt;reserved&gt;,&lt;Secure&gt;,&lt;Battery&gt;,&lt;Altera FPGA Probe&gt;,&lt;FlexRay Serial&gt;,&lt;reserved&gt;,&lt;RS-232/UART Serial&gt;,&lt;reserved&gt;            &lt;All field&gt; ::= {0   All}            &lt;reserved&gt; ::= 0            &lt;Factory MSO&gt; ::= {0   MSO}            &lt;Upgraded MSO&gt; ::= {0   MSO}            &lt;Xilinx FPGA Probe&gt; ::= {0   FPG}            &lt;Memory&gt; ::= {0   mem2M   mem8M}            &lt;Low Speed Serial&gt; ::= {0   LSS}            &lt;Automotive Serial&gt; ::= {0   AMS}            &lt;Secure&gt; ::= {0   SEC}            &lt;Battery&gt; ::= {0   BAT}            &lt;Altera FPGA Probe&gt; ::= {0   ALT}            &lt;FlexRay Serial&gt; ::= {0   FRS}            &lt;RS-232/UART Serial&gt; ::= {0   232}</p>
*RCL <value> (see <a href="#">page 122</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*RST (see <a href="#">page 123</a> )	n/a	See *RST (Reset) (see <a href="#">page 123</a> )
*SAV <value> (see <a href="#">page 126</a> )	n/a	<value> ::= {0   1   2   3   4   5   6   7   8   9}
*SRE <mask> (see <a href="#">page 127</a> )	*SRE? (see <a href="#">page 128</a> )	<p>&lt;mask&gt; ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. &lt;mask&gt; ::= following values:</p> <p>Bit Weight Name Enables</p> <pre> 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 36** Common (\*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see <a href="#">page 129</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 131</a> )	n/a	n/a
n/a	*TST? (see <a href="#">page 132</a> )	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see <a href="#">page 133</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.

### 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 676](#))

### 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 111
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 114
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 116
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127
  - [":SYSTem:ERRor"](#) on page 387

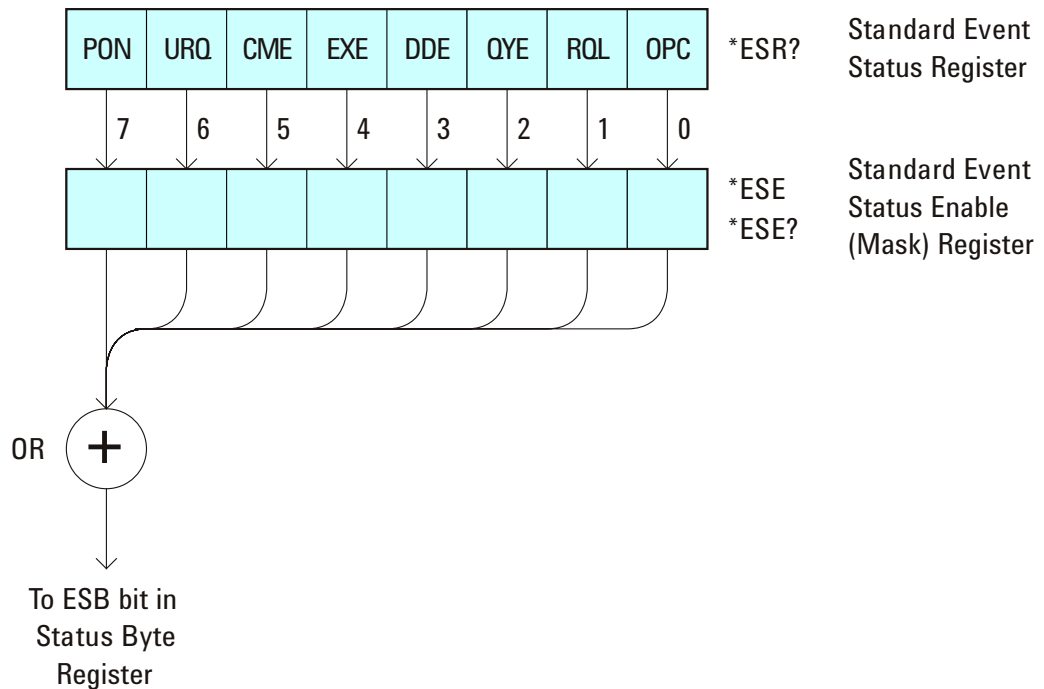
**\*ESE (Standard Event Status Enable)**

**C** (see [page 676](#))

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

Bit	Name	Description	When Set (1 = High = True), Enables:
7	PON	Power On	Event when an OFF to ON transition occurs.
6	URQ	User Request	Event when a front-panel key is pressed.
5	CME	Command Error	Event when a command error is detected.
4	EXE	Execution Error	Event when an execution error is detected.
3	DDE	Device Dependent Error	Event when a device-dependent error is detected.
2	QYE	Query Error	Event when a query error is detected.

**Table 37** Standard Event Status Enable (ESE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
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 111
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 116
  - ["\\*OPC \(Operation Complete\)"](#) on page 120
  - ["\\*CLS \(Clear Status\)"](#) on page 113

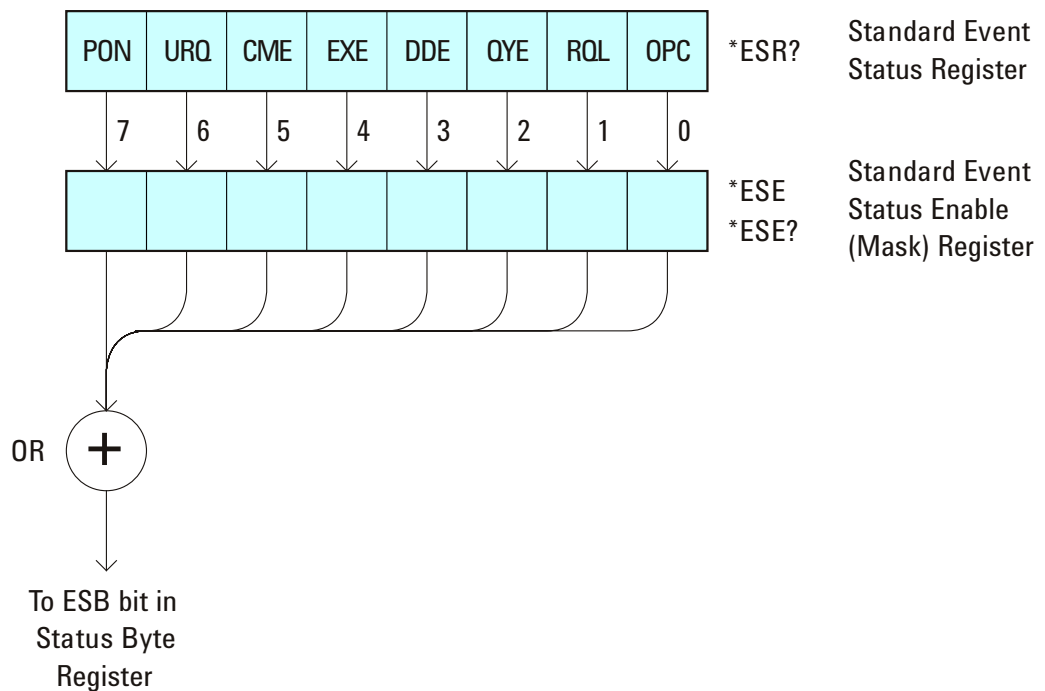
**\*ESR (Standard Event Status Register)**

**C** (see [page 676](#))

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

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	PON	Power On	An OFF to ON transition has occurred.
6	URQ	User Request	A front-panel key has been pressed.
5	CME	Command Error	A command error has been detected.
4	EXE	Execution Error	An execution error has been detected.
3	DDE	Device Dependent Error	A device-dependent error has been detected.
2	QYE	Query Error	A query error has been detected.

**Table 38** Standard Event Status Register (ESR) (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
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 111
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 114
  - ["\\*OPC \(Operation Complete\)"](#) on page 120
  - ["\\*CLS \(Clear Status\)"](#) on page 113
  - [":SYSTem:ERRor"](#) on page 387

## \*IDN (Identification Number)

**C** (see [page 676](#))

**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 111
  - ["\\*OPT \(Option Identification\)"](#) on page 121

## \*LRN (Learn Device Setup)

**C** (see [page 676](#))

### Query Syntax

\*LRN?

The \*LRN? query result contains the current state of the instrument. This query is similar to the :SYSTEM:SETup? (see [page 390](#)) 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 111
  - ["\\*RCL \(Recall\)"](#) on page 122
  - ["\\*SAV \(Save\)"](#) on page 126
  - [":SYSTEM:SETup"](#) on page 390

### \*OPC (Operation Complete)

**C** (see [page 676](#))

**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 111
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 114
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 116
  - ["\\*CLS \(Clear Status\)"](#) on page 113



**\*OPT (Option Identification)****C** (see [page 676](#))**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>,<Factory MSO>,<Upgraded MSO>,<Xilinx FPGA Probe>,<Memory>,<Low Speed Serial>,<Automotive Serial>,<reserved>,<Secure>,<Battery>,<Altera FPGA Probe>,<FlexRay Serial>,<reserved>,<RS-232/UART Serial>,<reserved>
```

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

```
<reserved> ::= 0
```

```
<Factory MSO> ::= {0 | MSO}
```

```
<Upgraded MSO> ::= {0 | MSO}
```

```
<Xilinx FPGA Probe> ::= {0 | FPG}
```

```
<Memory> ::= {0 | mem2M | mem8M}
```

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

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

```
<Secure> ::= {0 | SEC}
```

```
<Battery> ::= {0 | BAT}
```

```
<Altera FPGA Probe> ::= {0 | ALT}
```

```
<FlexRay Serial> ::= {0 | FRS}
```

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

The <Factory MSO> <Upgraded MSO> fields indicate whether the unit is a mixed-signal oscilloscope and, if so, whether it was factory installed or upgraded from an analog channels only oscilloscope (DSO).

The \*OPT? query returns the following:

Module	Module Id
No modules attached	0,0,0,0,MSO,0,0,mem8M,0,0,0,0,0,0,0,0,0

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 111
  - ["\\*IDN \(Identification Number\)"](#) on page 118

### \*RCL (Recall)



(see [page 676](#))

**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 111
  - ["\\*SAV \(Save\)"](#) on page 126

**\*RST (Reset)**

**C** (see [page 676](#))

**Command Syntax** \*RST

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

Acquire Menu	
Mode	Normal
Realtime	On
Averaging	Off
# Averages	8

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

Cursor Menu	
Source	Channel 1

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

Display Menu	
Definite persistence	Off
Grid	33%
Vectors	On

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

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

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

Trigger Menu	
Slope	Positive
HF Reject and noise reject	Off
Holdoff	60 ns
External probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1
External Units	Volts
External Impedance	1 M Ohm

**See Also** • ["Introduction to Common \(\\*\) Commands"](#) on page 111

**Example Code**

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

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

### \*SAV (Save)



(see [page 676](#))

**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 111
  - ["\\*RCL \(Recall\)"](#) on page 122

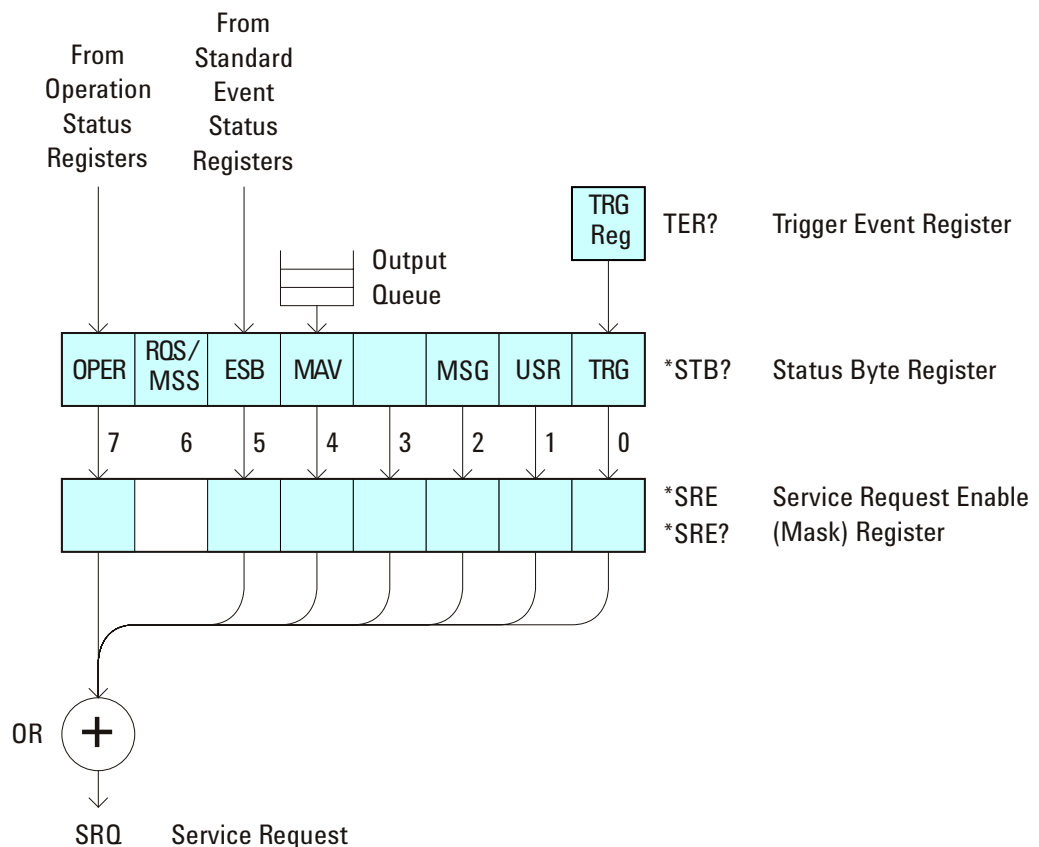
**\*SRE (Service Request Enable)**

**C** (see [page 676](#))

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

Bit	Name	Description	When Set (1 = High = True), Enables:
7	OPER	Operation Status Register	Interrupts when enabled conditions in the Operation Status Register (OPER) occur.
6	---	---	(Not used.)

**Table 39** Service Request Enable Register (SRE) (continued)

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

**Query Syntax** \*SRE?

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

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

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

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 111
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*CLS \(Clear Status\)"](#) on page 113



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

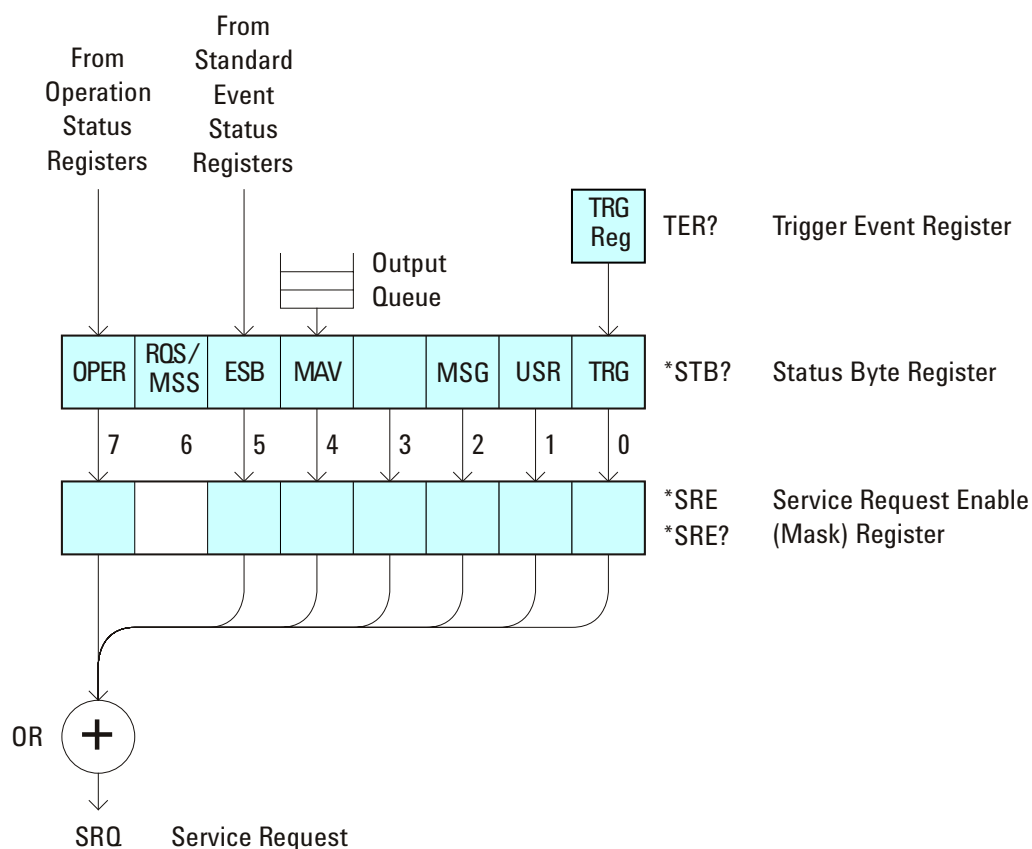
**C** (see [page 676](#))

**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 40** Status Byte Register (STB)

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	OPER	Operation Status Register	An enabled condition in the Operation Status Register (OPER) has occurred.

**Table 40** Status Byte Register (STB) (continued)

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

**NOTE**

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

- See Also**
- ["Introduction to Common \(\\*\) Commands"](#) on page 111
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127

## \*TRG (Trigger)

**C** (see [page 676](#))

### 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 111
  - [":DIGitize"](#) on page 145
  - [":RUN"](#) on page 165
  - [":STOP"](#) on page 169

### \*TST (Self Test)

**C** (see [page 676](#))

**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 111

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

**C** (see [page 676](#))

**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 111

## 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 136.

**Table 41** Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see <a href="#">page 137</a> )	:ACTivity? (see <a href="#">page 137</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 138</a> )	{0   1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see <a href="#">page 139</a> )	n/a	<source> ::= CHANnel<n> for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD1   POD2} for MSO models <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value> (see <a href="#">page 141</a> )	:AUToscale:AMODE? (see <a href="#">page 141</a> )	<value> ::= {NORMAL   CURRENT}}
:AUToscale:CHANnels <value> (see <a href="#">page 142</a> )	:AUToscale:CHANnels? (see <a href="#">page 142</a> )	<value> ::= {ALL   DISPLAYed}}
:BLANK [<source>] (see <a href="#">page 143</a> )	n/a	<source> ::= {CHANnel<n>}   FUNCTION   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see <a href="#">page 144</a> )	n/a	n/a

**Table 41** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,...,<source>]] (see <a href="#">page 145</a> )	n/a	<source> ::= {CHANnel<n>   FUNCTION   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n> (see <a href="#">page 147</a> )	:HWEenable? (see <a href="#">page 147</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister:CONDition? (see <a href="#">page 149</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENT]? (see <a href="#">page 151</a> )	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see <a href="#">page 153</a> )	n/a	<pixel memory> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:OPEE <n> (see <a href="#">page 154</a> )	:OPEE? (see <a href="#">page 155</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister:CONDition? (see <a href="#">page 156</a> )	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister[:EVENT]? (see <a href="#">page 158</a> )	<n> ::= 16-bit integer in NR1 format
:OVLenable <mask> (see <a href="#">page 160</a> )	:OVLenable? (see <a href="#">page 161</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 162</a> )	<value> ::= integer in NR1 format. See OVLenable for <value>

**Table 41** Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:PRINT [<options>] (see <a href="#">page 164</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 165</a> )	n/a	n/a
n/a	:SERial (see <a href="#">page 166</a> )	<return value> ::= unquoted string containing serial number
:SINGle (see <a href="#">page 167</a> )	n/a	n/a
n/a	:STATus? <display> (see <a href="#">page 168</a> )	{0   1} <display> ::= {CHANnel<n>   DIGital0,...,DIGital15   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} <n> ::= 1-2 or 1-4 in NR1 format
:STOP (see <a href="#">page 169</a> )	n/a	n/a
n/a	:TER? (see <a href="#">page 170</a> )	{0   1}
:VIEW <source> (see <a href="#">page 171</a> )	n/a	<source> ::= {CHANnel<n>   PMEMory{0   1   2   3   4   5   6   7   8   9}   FUNCTION   MATH   SBUS} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   PMEMory{0   1   2   3   4   5   6   7   8   9}   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

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



**:ACTivity**

**N** (see [page 676](#))

**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 136
  - [":POD<n>:THReshold"](#) on page 336
  - [":DIGital<n>:THReshold"](#) on page 229

**:AER (Arm Event Register)**

**C** (see [page 676](#))

**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 136
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154
  - [":OPERRegister:CONDition \(Operation Status Condition Register\)"](#) on page 156
  - [":OPERRegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 158
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127

**:AUToscale**

**C** (see [page 676](#))

**Command Syntax** :AUToscale

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

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

```
<source> ::= {DIGital0,...,DIGital15 | POD1 | POD2 | CHANnel<n>} for the  
MSO models
```

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

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

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

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

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see [":AUToscale:CHANnels"](#) on page 142) 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.
- Trace memories.
- Delayed time base mode.

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

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 136
  - [":AUToscale:CHANnels"](#) on page 142
  - [":AUToscale:AMODE"](#) on page 141

**Example Code**

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

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:AUToscale:AMODE**

**N** (see [page 676](#))

**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 136
  - [":AUToscale"](#) on page 139
  - [":AUToscale:CHANnels"](#) on page 142
  - [":ACQUIRE:TYPE"](#) on page 185
  - [":ACQUIRE:MODE"](#) on page 178

**:AUToscale:CHANnels**

**N** (see [page 676](#))

**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 136
  - [":AUToscale"](#) on page 139
  - [":AUToscale:AMODE"](#) on page 141
  - [":VIEW"](#) on page 171
  - [":BLANK"](#) on page 143

**:BLANK**

**N** (see [page 676](#))

**Command Syntax**

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

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

```
<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | POD{1 | 2}
              | BUS{1 | 2} | FUNCTION | MATH | SBUS} for the MSO models
```

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

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

The :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 136
- [":CDISplay"](#) on page 144
- [":CHANnel<n>:DISPlay"](#) on page 209
- [":DIGital<n>:DISPlay"](#) on page 225
- [":FUNCTION:DISPlay"](#) on page 254
- [":POD<n>:DISPlay"](#) on page 334
- [":STATus"](#) on page 168
- [":VIEW"](#) on page 171

**Example Code**

- ["Example Code"](#) on page 171

## :CDISplay

**C** (see [page 676](#))

**Command Syntax** :CDISplay

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

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 136
  - [":DISplay:CLEar"](#) on page 232



**:DIGitize**

**C** (see [page 676](#))

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

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | POD{1 | 2} | BUS{1 | 2} | FUNCTION | MATH | SBUS} for the MSO models

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

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

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

The :DIGitize command is a specialized RUN command. It causes the instrument to acquire waveforms according to the settings of the :ACQUIRE commands subsystem. When the acquisition is complete, the instrument is stopped. If no argument is given, :DIGitize acquires the channels currently displayed. If no channels are displayed, all channels are acquired.

**NOTE**

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

**NOTE**

MATH is an alias for FUNCTION.

- See Also**
- "[Introduction to Root \(:\)](#) Commands" on page 136
  - ":RUN" on page 165
  - ":SINGLE" on page 167
  - ":STOP" on page 169
  - ":ACQUIRE Commands" on page 172
  - ":WAVEform Commands" on page 523

**Example Code**

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

```
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured. If a
' measurement is immediately requested, there may have not been
' enough time for the data acquisition process to collect data, and
' the results may not be accurate. An error value of 9.9E+37 may be
' returned over the bus in this situation.
'
myScope.WriteString ":DIGITIZE CHAN1"
```

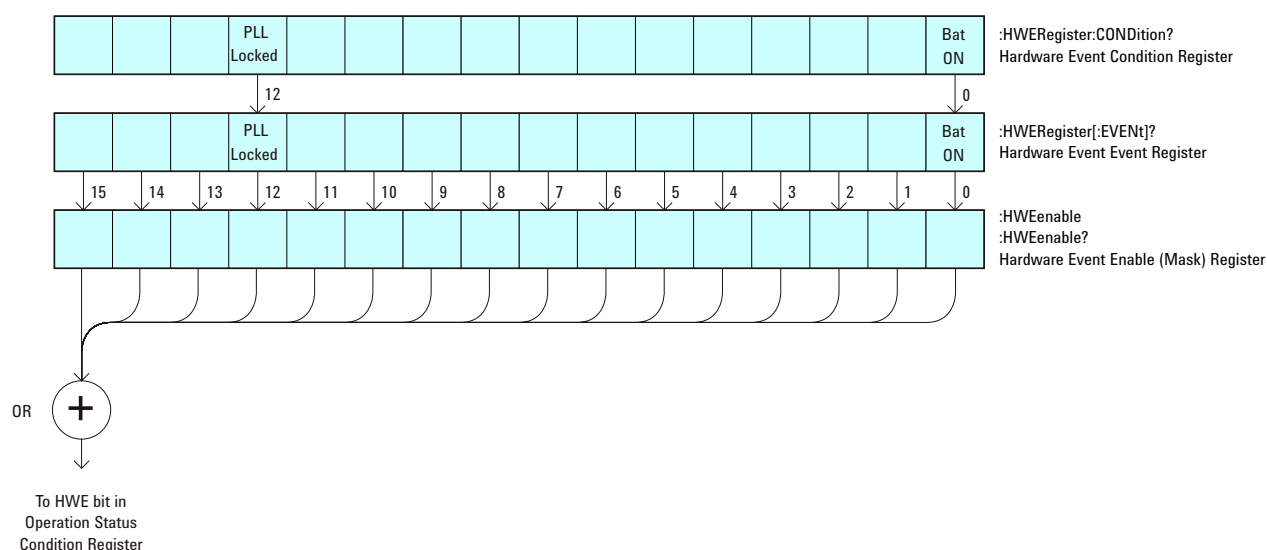
Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

## :HWEenable (Hardware Event Enable Register)

**N** (see page 676)

**Command Syntax** :HWEenable <mask>  
 <mask> ::= 16-bit integer

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



**Table 42** Hardware Event Enable Register (HWEenable)

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

**Query Syntax** :HWEenable?

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

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

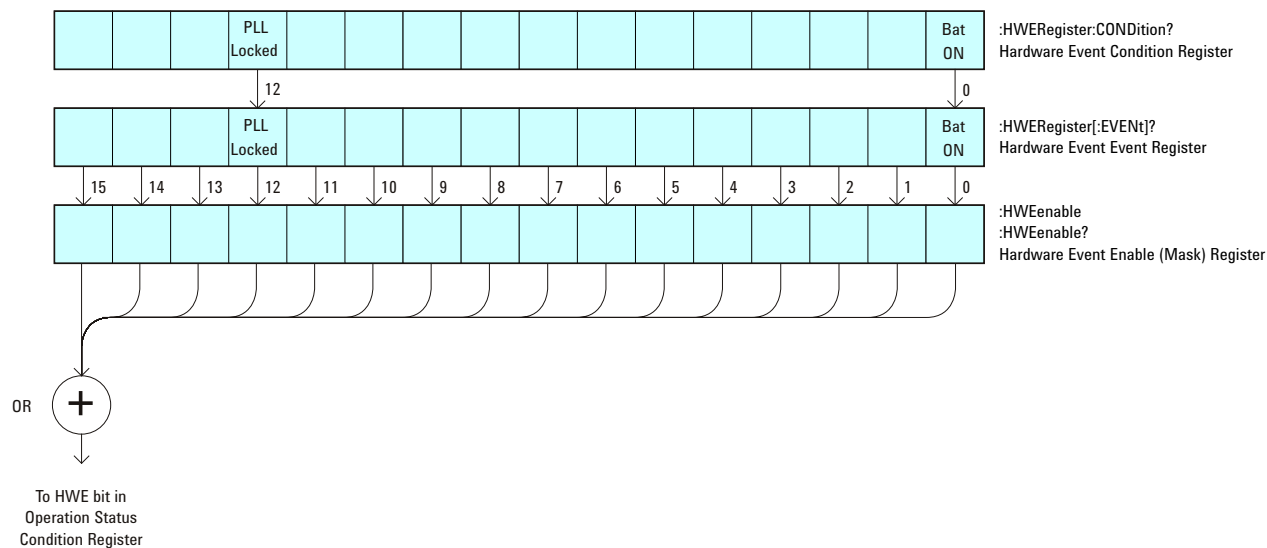
- See Also**
- ["Introduction to Root \(:\) Commands" on page 136](#)
  - [":AER \(Arm Event Register\)" on page 138](#)
  - [":CHANnel<n>:PROTection" on page 218](#)
  - [":EXTeRnal:PROTection" on page 247](#)
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)" on page 158](#)
  - [":OVLenable \(Overload Event Enable Register\)" on page 160](#)
  - [":OVLRegister \(Overload Event Register\)" on page 162](#)
  - ["\\*STB \(Read Status Byte\)" on page 129](#)
  - ["\\*SRE \(Service Request Enable\)" on page 127](#)

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

**N** (see [page 676](#))

**Query Syntax** :HWERegister:CONDition?

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



**Table 43** Hardware Event Condition Register

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

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

<value> ::= integer in NR1 format.

- See Also**
- ["Introduction to Root \(: \) Commands"](#) on page 136
  - [":CHANnel<n>:PROTection"](#) on page 218
  - [":EXTErnal:PROTection"](#) on page 247
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 158

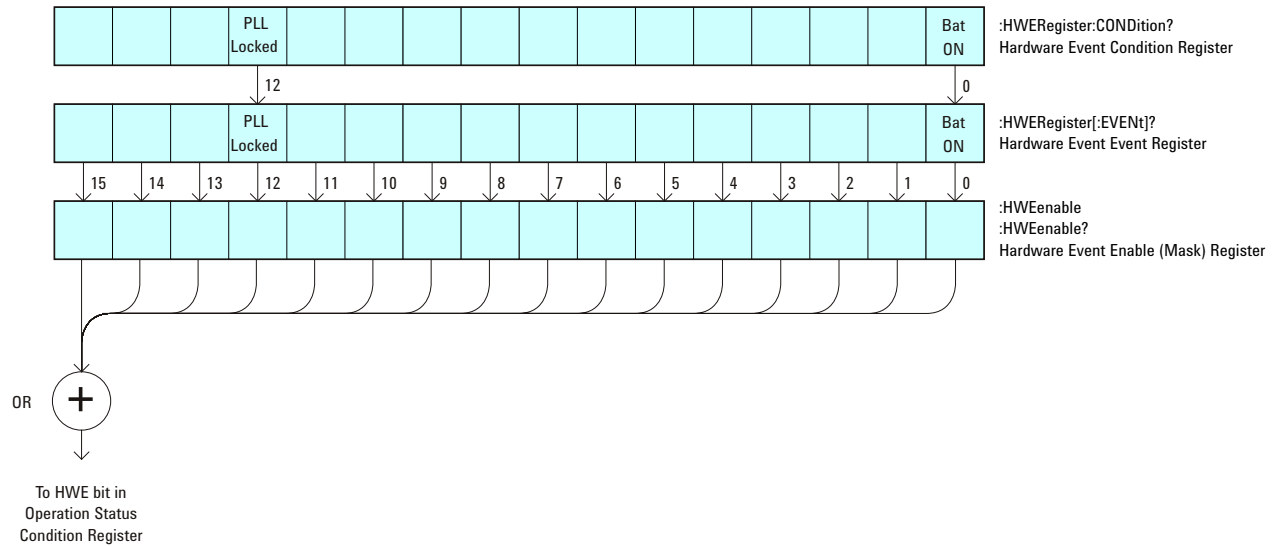
- [":OVLenable \(Overload Event Enable Register\)" on page 160](#)
- [":OVLRegister \(Overload Event Register\)" on page 162](#)
- ["\\*STB \(Read Status Byte\)" on page 129](#)
- ["\\*SRE \(Service Request Enable\)" on page 127](#)

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

**N** (see [page 676](#))

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

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



**Table 44** Hardware Event Event Register

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

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

<value> ::= integer in NR1 format.

- See Also**
- ["Introduction to Root \(: \) Commands"](#) on page 136
  - [":CHANnel<n>:PROTection"](#) on page 218
  - [":EXTeRnal:PROTection"](#) on page 247
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154

- [":OPERRegister:CONDition \(Operation Status Condition Register\)" on page 156](#)
- [":OVLenable \(Overload Event Enable Register\)" on page 160](#)
- [":OVLRegister \(Overload Event Register\)" on page 162](#)
- ["\\*STB \(Read Status Byte\)" on page 129](#)
- ["\\*SRE \(Service Request Enable\)" on page 127](#)



**:MERGe**

**N** (see [page 676](#))

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

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

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

- See Also**
- ["Introduction to Root \(: \) Commands"](#) on page 136
  - ["\\*SAV \(Save\)"](#) on page 126
  - ["\\*RCL \(Recall\)"](#) on page 122
  - [":VIEW"](#) on page 171
  - [":BLANK"](#) on page 143

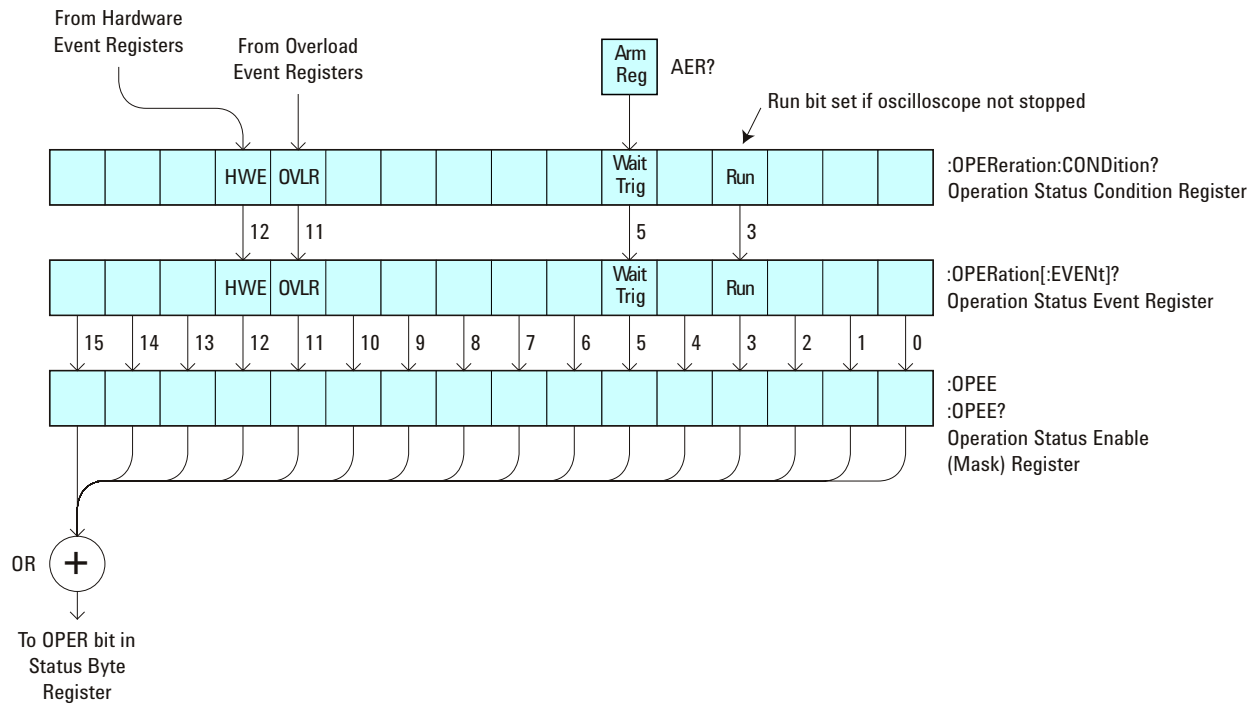
**:OPEE (Operation Status Enable Register)**

**C** (see [page 676](#))

**Command Syntax** :OPEE <mask>

<mask> ::= 16-bit integer

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



**Table 45** Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVL	Overload	Event when 50Ω input overload occurs.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)

**Table 45** Operation Status Enable Register (OPEE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
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.

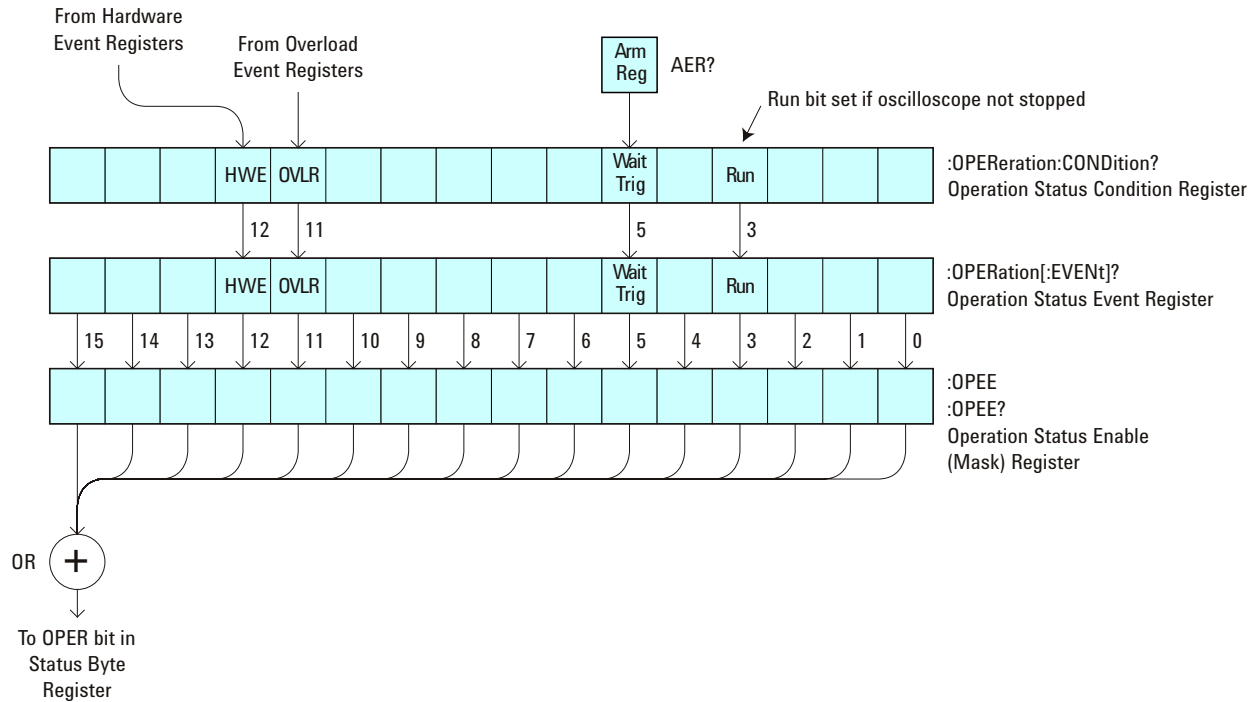
**Return Format**    <value><NL>

<value> ::= integer in NR1 format.

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 136
  - [":AER \(Arm Event Register\)"](#) on page 138
  - [":CHANnel<n>:PROTection"](#) on page 218
  - [":EXTeRnal:PROTection"](#) on page 247
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 158
  - [":OVLenable \(Overload Event Enable Register\)"](#) on page 160
  - [":OVLRegister \(Overload Event Register\)"](#) on page 162
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127

**:OPERRegister:CONDition (Operation Status Condition Register)****C** (see [page 676](#))**Query Syntax** :OPERRegister:CONDition?

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

**Table 46** Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred..
11	OVLr	Overload	A 50Ω input overload has occurred.
10-6	---	---	(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.

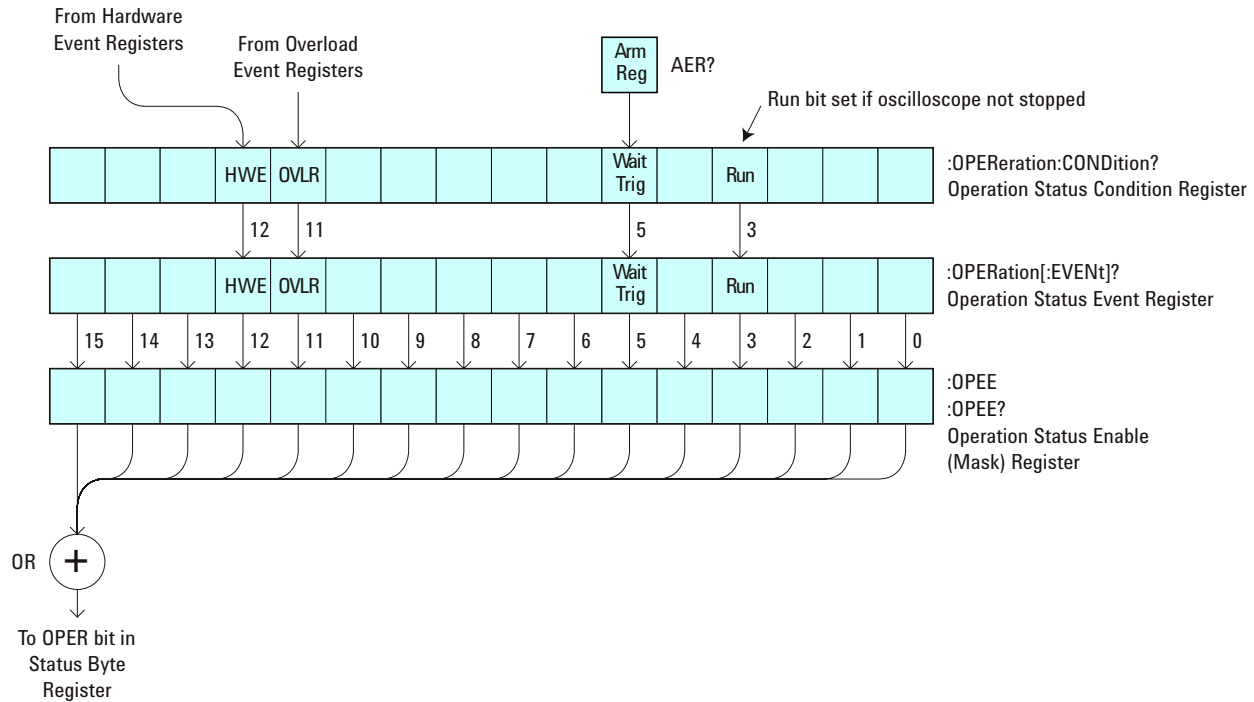
- See Also**
- ["Introduction to Root \(:\) Commands" on page 136](#)
  - [":CHANnel<n>:PROTection" on page 218](#)
  - [":EXTErnal:PROTection" on page 247](#)
  - [":OPEE \(Operation Status Enable Register\)" on page 154](#)
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)" on page 158](#)
  - [":OVLenable \(Overload Event Enable Register\)" on page 160](#)
  - [":OVLRegister \(Overload Event Register\)" on page 162](#)
  - ["\\*STB \(Read Status Byte\)" on page 129](#)
  - ["\\*SRE \(Service Request Enable\)" on page 127](#)
  - [":HWERegister\[:EVENT\] \(Hardware Event Event Register\)" on page 151](#)
  - [":HWEenable \(Hardware Event Enable Register\)" on page 147](#)

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

**C** (see [page 676](#))

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

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



**Table 47** Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred.
11	OVLr	Overload	A 50Ω input overload has occurred.
10-6	---	---	(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 136](#)
  - [":CHANnel<n>:PROTection" on page 218](#)
  - [":EXTErnal:PROTection" on page 247](#)
  - [":OPEE \(Operation Status Enable Register\)" on page 154](#)
  - [":OPERegister:CONDition \(Operation Status Condition Register\)" on page 156](#)
  - [":OVLenable \(Overload Event Enable Register\)" on page 160](#)
  - [":OVLRegister \(Overload Event Register\)" on page 162](#)
  - ["\\*STB \(Read Status Byte\)" on page 129](#)
  - ["\\*SRE \(Service Request Enable\)" on page 127](#)
  - [":HWERegister\[:EVENT\] \(Hardware Event Event Register\)" on page 151](#)
  - [":HWEenable \(Hardware Event Enable Register\)" on page 147](#)

**:OVLenable (Overload Event Enable Register)**

**C** (see [page 676](#))

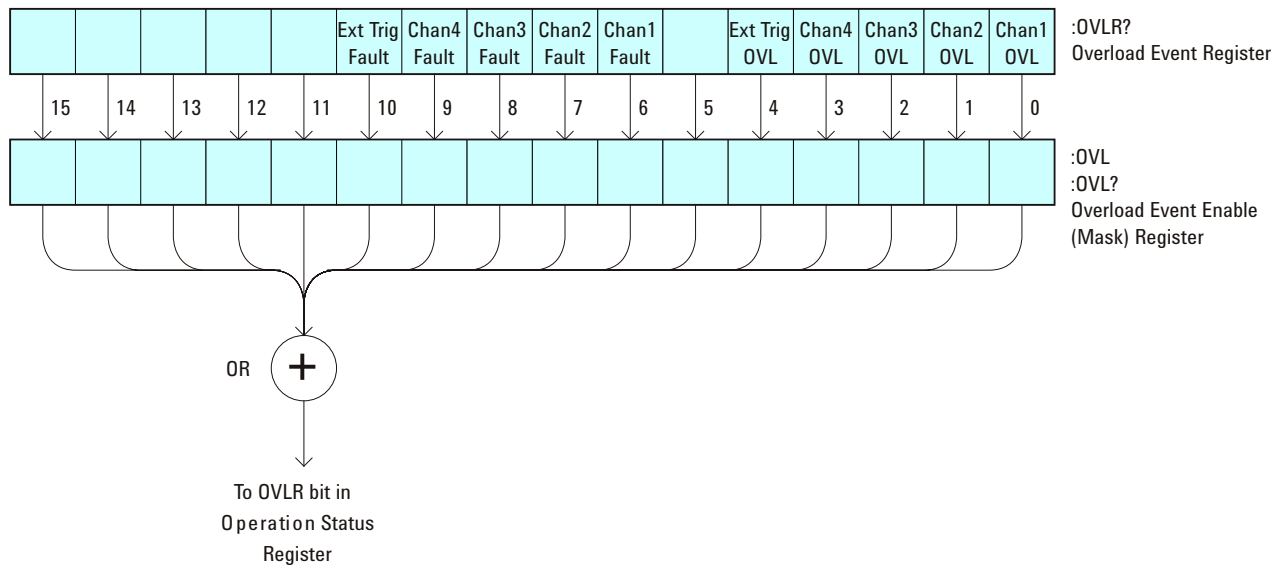
**Command Syntax** :OVLenable <enable\_mask>  
 <enable\_mask> ::= 16-bit integer

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 48** Overload Event Enable Register (OVL)

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



**Table 48** Overload Event Enable Register (OVL) (continued)

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

**Query Syntax** :OVLenable?

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

**Return Format** <enable\_mask><NL>  
 <enable\_mask> ::= integer in NR1 format.

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 136
  - [":CHANnel<n>:PROTection"](#) on page 218
  - [":EXTeRnal:PROTection"](#) on page 247
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154
  - [":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 156
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 158
  - [":OVLRegister \(Overload Event Register\)"](#) on page 162
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127

**:OVLRegister (Overload Event Register)**

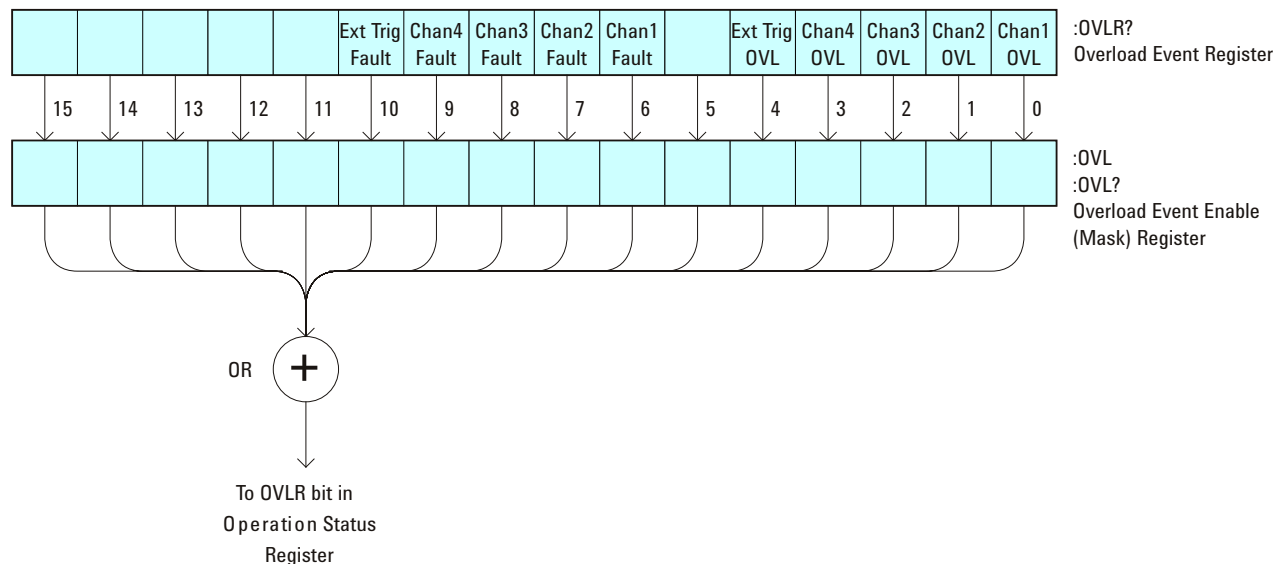
**C** (see [page 676](#))

**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 49** Overload Event Register (OVL)

Bit	Description	When Set (1 = High = True), Indicates:
15-11	---	(Not used.)
10	External Trigger Fault	Fault has occurred on External Trigger input.
9	Channel 4 Fault	Fault has occurred on Channel 4 input.
8	Channel 3 Fault	Fault has occurred on Channel 3 input.
7	Channel 2 Fault	Fault has occurred on Channel 2 input.
6	Channel 1 Fault	Fault has occurred on Channel 1 input.
5	---	(Not used.)

**Table 49** Overload Event Register (OVLr) (continued)

Bit	Description	When Set (1 = High = True), Indicates:
4	External Trigger OVL	Overload has occurred on External Trigger input.
3	Channel 4 OVL	Overload has occurred on Channel 4 input.
2	Channel 3 OVL	Overload has occurred on Channel 3 input.
1	Channel 2 OVL	Overload has occurred on Channel 2 input.
0	Channel 1 OVL	Overload has occurred on Channel 1 input.

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

<value> ::= integer in NR1 format.

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 136
  - [":CHANnel<n>:PROTection"](#) on page 218
  - [":EXTeRnal:PROTection"](#) on page 247
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154
  - [":OVLenable \(Overload Event Enable Register\)"](#) on page 160
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127

### :PRINt

**C** (see [page 676](#))

**Command Syntax** :PRINt [<options>]

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

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

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

The PRINt command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used. Refer to [":HARDcopy:FORMat"](#) on page 608 for more information.

- See Also**
- ["Introduction to Root \(: \) Commands"](#) on page 136
  - ["Introduction to :HARDcopy Commands"](#) on page 267
  - [":HARDcopy:FORMat"](#) on page 608
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:GRAYscale"](#) on page 609
  - [":DISPlay:DATA"](#) on page 233

**:RUN**

**C** (see [page 676](#))

**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 136
  - [":SINGle"](#) on page 167
  - [":STOP"](#) on page 169

**Example Code**

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

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

### :SERial

**N** (see [page 676](#))

**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 136

## :SINGle



(see [page 676](#))

**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 136
  - [":RUN"](#) on page 165
  - [":STOP"](#) on page 169

**:STATus**

**N** (see [page 676](#))

**Query Syntax**

:STATus? <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | POD{1 | 2}  
| BUS{1 | 2} | FUNCTION | MATH | SBUS} for the MSO models

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

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

The :STATus? query reports whether the channel, function, trace memory, 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 136
- [":BLANK"](#) on page 143
- [":CHANnel<n>:DISPlay"](#) on page 209
- [":DIGital<n>:DISPlay"](#) on page 225
- [":FUNCTION:DISPlay"](#) on page 254
- [":POD<n>:DISPlay"](#) on page 334
- [":VIEW"](#) on page 171



## :STOP

**C** (see [page 676](#))

**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 136
  - [":RUN"](#) on page 165
  - [":SINGle"](#) on page 167

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

**:TER (Trigger Event Register)**

**C** (see [page 676](#))

**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 136
  - ["\\*SRE \(Service Request Enable\)"](#) on page 127
  - ["\\*STB \(Read Status Byte\)"](#) on page 129

**:VIEW**

**N** (see [page 676](#))

**Command Syntax** :VIEW <source>

<source> ::= {CHANnel<n> | PMEMory0,...,PMEMory9 | FUNction | MATH  
| SBUS} for DSO models

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | PMEMory0,...,PMEMory9  
| POD{1 | 2} | BUS{1 | 2} | FUNction | MATH | SBUS} for  
MSO models

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

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

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

**NOTE**

MATH is an alias for FUNction.

- See Also**
- ["Introduction to Root \(:\) Commands" on page 136](#)
  - [":BLANK" on page 143](#)
  - [":CHANnel<n>:DISPlay" on page 209](#)
  - [":DIGital<n>:DISPlay" on page 225](#)
  - [":FUNction:DISPlay" on page 254](#)
  - [":POD<n>:DISPlay" on page 334](#)
  - [":STATus" on page 168](#)

**Example Code**

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

Example program from the start: ["VISA COM Example in Visual Basic" on page 764](#)

## :ACQUIRE Commands

Set the parameters for acquiring and storing data. See ["Introduction to :ACQUIRE Commands"](#) on page 172.

**Table 50** :ACQUIRE Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQUIRE:AALias? (see <a href="#">page 174</a> )	{1   0}
:ACQUIRE:COMPLete <complete> (see <a href="#">page 175</a> )	:ACQUIRE:COMPLete? (see <a href="#">page 175</a> )	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNt <count> (see <a href="#">page 176</a> )	:ACQUIRE:COUNt? (see <a href="#">page 176</a> )	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQUIRE:DAALias <mode> (see <a href="#">page 177</a> )	:ACQUIRE:DAALias? (see <a href="#">page 177</a> )	<mode> ::= {DISable   AUTO}
:ACQUIRE:MODE <mode> (see <a href="#">page 178</a> )	:ACQUIRE:MODE? (see <a href="#">page 178</a> )	<mode> ::= {RTIME   ETIME   SEGmented}
n/a	:ACQUIRE:POINts? (see <a href="#">page 179</a> )	<# points> ::= an integer in NR1 format
:ACQUIRE:RSIGnal <ref_signal_mode> (see <a href="#">page 180</a> )	:ACQUIRE:RSIGnal? (see <a href="#">page 180</a> )	<ref_signal_mode> ::= {OFF   OUT   IN}
:ACQUIRE:SEGmented:CO UNT <count> (see <a href="#">page 181</a> )	:ACQUIRE:SEGmented:CO UNT? (see <a href="#">page 181</a> )	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
:ACQUIRE:SEGmented:IN Dex <index> (see <a href="#">page 182</a> )	:ACQUIRE:SEGmented:IN Dex? (see <a href="#">page 182</a> )	<index> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:ACQUIRE:SRATe? (see <a href="#">page 184</a> )	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQUIRE:TYPE <type> (see <a href="#">page 185</a> )	:ACQUIRE:TYPE? (see <a href="#">page 185</a> )	<type> ::= {NORMal   AVERage   HRESolution   PEAK}

### Introduction to :ACQUIRE Commands

The ACQUIRE subsystem controls the way in which waveforms are acquired. These acquisition types are available: normal, averaging, peak detect, and high resolution. Two acquisition modes are available: real-time mode, and equivalent-time mode.

Normal

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

#### Averaging

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

#### High-Resolution

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

#### Peak Detect

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

#### Real-time Mode

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

#### Equivalent-time Mode

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

#### Reporting the Setup

Use `:ACQUIRE?` to query setup information for the `ACQUIRE` subsystem.

#### Return Format

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

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

## :ACquire:AALias

**N** (see [page 676](#))

**Query Syntax** :ACquire:AALias?

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

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

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 172
  - [":ACquire:DAALias"](#) on page 177

**:ACQUIRE:COMPLete**

**C** (see [page 676](#))

**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 172
  - [":ACQUIRE:TYPE"](#) on page 185
  - [":DIGitize"](#) on page 145
  - [":WAVEform:POINTS"](#) on page 536

**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"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:ACquire:COUNT**

**C** (see [page 676](#))

**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 172
  - [":ACquire:TYPE"](#) on page 185
  - [":DIGitize"](#) on page 145
  - [":WAVEform:COUNT"](#) on page 532



**:ACQUIRE:DAALias**

**N** (see [page 676](#))

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

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

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

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

**Query Syntax** :ACQUIRE:DAALias?

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

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

- See Also**
- ["Introduction to :ACQUIRE Commands"](#) on page 172
  - [":ACQUIRE:AALias"](#) on page 174

**:ACquire:MODE**

**C** (see [page 676](#))

**Command Syntax** :ACquire:MODE <mode>

<mode> ::= {RTIME | ETIME | SEGmented}

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

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

**NOTE**

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

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

**Query Syntax** :ACquire:MODE?

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

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

<mode\_argument> ::= {RTIM | ETIM | SEGM}

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 172
  - [":ACquire:TYPE"](#) on page 185

**:ACquire:POINts**(see [page 676](#))**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>

&lt;points\_argument&gt; ::= an integer in NR1 format

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 172
  - [":DIGitize"](#) on page 145
  - [":WAVEform:POINts"](#) on page 536

**:ACquire:RSIGnal**

**N** (see [page 676](#))

**Command Syntax** :ACquire:RSIGnal <ref\_signal\_mode>  
 <ref\_signal\_mode> ::= {OFF | OUT | IN}

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

- The OFF mode disables the oscilloscope's 10 MHz REF BNC connector.
- The OUT mode is used to synchronize the timebase of two or more instruments.
- The IN mode is used to supply a sample clock to the oscilloscope. A 10 MHz square or sine wave signal is input to the BNC connector labeled 10 MHz REF. The amplitude must be between 180 mV and 1 V, with an offset of between 0 V and 2 V.

**CAUTION**

Do not apply more than  $\pm 15$  V at the 10 MHz REF BNC connector on the rear panel, or damage to the instrument may occur.

**Query Syntax** :ACquire:RSIGnal?

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

**Return Format** <ref\_signal\_mode><NL>  
 <ref\_signal\_mode> ::= {OFF | OUT | IN}

- See Also**
- [":TIMEbase:REFClock"](#) on page 398
  - The *Agilent InfiniiVision 6000 Series Oscilloscope User's Guide* for information on using the 10 MHz reference clock.

**:ACQuire:SEGMented:COUNT**

**N** (see [page 676](#))

**Command Syntax** :ACQuire:SEGMented:COUNT <count>  
 <count> ::= an integer from 2 to 2000 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 command. The number of memory segments in the current acquisition is returned by the :WAVEform:SEGMented:COUNT? query.

**Query Syntax** :ACQuire:SEGMented:COUNT?

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

**Return Format** <count><NL>  
 <count> ::= an integer from 2 to 2000 in NR1 format

- See Also**
- [":ACQuire:MODE"](#) on page 178
  - [":DIGitize"](#) on page 145
  - [":WAVEform:SEGMented:COUNT"](#) on page 543
  - ["Introduction to :ACQuire Commands"](#) on page 172

**Example Code** • ["Example Code"](#) on page 182

**:ACquire:SEGMented:INdex**

**N** (see [page 676](#))

**Command Syntax** :ACquire:SEGMented:INdex <index>  
 <index> ::= an integer from 2 to 2000 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 command. The number of memory segments that have been acquired is returned by the :WAVEform:SEGMented:COUNT? query. The time tag of the currently indexed memory segment is returned by the :WAVEform:SEGMented:TTAG? query.

**Query Syntax** :ACquire:SEGMented:INdex?

The :ACquire:SEGMented:INdex? query returns the current segmented memory index setting.

**Return Format** <index><NL>  
 <index> ::= an integer from 2 to 2000 in NR1 format

- See Also**
- [":ACquire:MODE" on page 178](#)
  - [":ACquire:SEGMented:COUNT" on page 181](#)
  - [":DIGitize" on page 145](#)
  - [":WAVEform:SEGMented:COUNT" on page 543](#)
  - [":WAVEform:SEGMented:TTAG" on page 544](#)
  - ["Introduction to :ACquire Commands" on page 172](#)

**Example Code**

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

' Set the number of segments to 50.
myScope.WriteString ":ACquire:SEGMented:COUNT 50"
myScope.WriteString ":ACquire:SEGMented:COUNT?"
strQueryResult = myScope.ReadString
Debug.Print "Acquisition memory segments: " + strQueryResult
```

```

' Acquire data using :DIGitize.
myScope.WriteString ":DIGitize"
Debug.Print ":DIGitize to acquire data."

' Wait until the desired number of segments is acquired.
Do
    myScope.WriteString ":WAVEform:SEGmented:COUNT?"
    varQueryResult = myScope.ReadNumber
Loop Until varQueryResult = 50
Debug.Print "Number of segments in acquired data: " _
    + FormatNumber(varQueryResult)

Dim lngSegments As Long
lngSegments = varQueryResult

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

For lngI = lngSegments To 1 Step -1

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

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

Next lngI

```

## :ACQuire:SRATe

**N** (see [page 676](#))

**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 172
  - [":ACQuire:POINts"](#) on page 179



## :ACQUIRE:TYPE

**C** (see [page 676](#))

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

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

The :ACQUIRE:TYPE command selects the type of data acquisition that is to take place. The acquisition types are: NORMal, AVERage, HRESolution, and PEAK.

- The :ACQUIRE:TYPE NORMal command sets the oscilloscope in the normal mode.
- The :ACQUIRE:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQUIRE:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 1 to 65536. The COUNT value determines the number of averages that must be acquired.
- The :ACQUIRE:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

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

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

### NOTE

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

**Query Syntax** :ACQUIRE:TYPE?

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

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

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

- See Also**
- ["Introduction to :ACQUIRE Commands"](#) on page 172
  - [":ACQUIRE:COUNt"](#) on page 176
  - [":ACQUIRE:MODE"](#) on page 178

- [":DIGitize"](#) on page 145
- [":WAVeform:TYPE"](#) on page 550
- [":WAVeform:PREamble"](#) on page 540

### Example Code

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

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

## :BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See ["Introduction to :BUS<n> Commands"](#) on page 188.

**Table 51** :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0   OFF}   {1   ON}} (see <a href="#">page 189</a> )	:BUS<n>:BIT<m>? (see <a href="#">page 189</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0   OFF}   {1   ON}} (see <a href="#">page 190</a> )	:BUS<n>:BITS? (see <a href="#">page 190</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> ::= 0-15; an integer in NR1 format
:BUS<n>:CLear (see <a href="#">page 192</a> )	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 193</a> )	:BUS<n>:DISPlay? (see <a href="#">page 193</a> )	{0   1} <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:LABel <string> (see <a href="#">page 194</a> )	:BUS<n>:LABel? (see <a href="#">page 194</a> )	<string> ::= quoted ASCII string up to 16 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see <a href="#">page 195</a> )	:BUS<n>:MASK? (see <a href="#">page 195</a> )	<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 676](#))

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

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

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

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

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

**NOTE**

This command is only valid for the MSO models.

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

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

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

<display> ::= {0 | 1}

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 188
  - [":BUS<n>:BITS"](#) on page 190
  - [":BUS<n>:CLEar"](#) on page 192
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":BUS<n>:LABel"](#) on page 194
  - [":BUS<n>:MASK"](#) on page 195

**Example Code**

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

**:BUS<n>:BITS**

**N** (see [page 676](#))

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

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

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

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

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

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

**NOTE**

This command is only valid for the MSO models.

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

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

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

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

<display> ::= {0 | 1}

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 188
  - [":BUS<n>:BIT<m>"](#) on page 189
  - [":BUS<n>:CLEar"](#) on page 192
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":BUS<n>:LABel"](#) on page 194
  - [":BUS<n>:MASK"](#) on page 195

**Example Code**

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

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

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

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

### :BUS<n>:CLEar

**N** (see [page 676](#))

**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 188
  - [":BUS<n>:BIT<m>"](#) on page 189
  - [":BUS<n>:BITS"](#) on page 190
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":BUS<n>:LABel"](#) on page 194
  - [":BUS<n>:MASK"](#) on page 195



**:BUS<n>:DISPlay**

**N** (see [page 676](#))

**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 188
  - [":BUS<n>:BIT<m>"](#) on page 189
  - [":BUS<n>:BITS"](#) on page 190
  - [":BUS<n>:CLEar"](#) on page 192
  - [":BUS<n>:LABel"](#) on page 194
  - [":BUS<n>:MASK"](#) on page 195

**:BUS<n>:LABel**

**N** (see [page 676](#))

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

<quoted\_string> ::= any series of 16 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 16 characters or less, and may contain any commonly used ASCII characters. Labels with more than 16 characters are truncated to 16 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 16 or less characters as a quoted ASCII string.

- See Also**
- ["Introduction to :BUS<n> Commands"](#) on page 188
  - [":BUS<n>:BIT<m>"](#) on page 189
  - [":BUS<n>:BITS"](#) on page 190
  - [":BUS<n>:CLEar"](#) on page 192
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":BUS<n>:MASK"](#) on page 195
  - [":CHANnel<n>:LABel"](#) on page 212
  - [":DISPlay:LABList"](#) on page 236
  - [":DIGital<n>:LABel"](#) on page 226

**Example Code**

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

**:BUS<n>:MASK**

**N** (see [page 676](#))

**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 188
  - [":BUS<n>:BIT<m>"](#) on page 189
  - [":BUS<n>:BITS"](#) on page 190
  - [":BUS<n>:CLEar"](#) on page 192
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":BUS<n>:LABel"](#) on page 194

## :CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See ["Introduction to :CALibrate Commands"](#) on page 196.

**Table 52** :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see <a href="#">page 197</a> )	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string> (see <a href="#">page 198</a> )	:CALibrate:LABel? (see <a href="#">page 198</a> )	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:START (see <a href="#">page 199</a> )	n/a	n/a
n/a	:CALibrate:STATus? (see <a href="#">page 200</a> )	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITCh? (see <a href="#">page 201</a> )	{PROTected   UNPRotected}
n/a	:CALibrate:TEMPeratur e? (see <a href="#">page 202</a> )	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see <a href="#">page 203</a> )	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

### Introduction to :CALibrate Commands

The CALibrate subsystem provides utility commands for:

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

**:CALibrate:DATE**

**N** (see [page 676](#))

**Query Syntax** :CALibrate:DATE?

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

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

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

**See Also** • ["Introduction to :CALibrate Commands"](#) on page 196

**:CALibrate:LABel**

**N** (see [page 676](#))

**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 196

## :CALibrate:START

**N** (see [page 676](#))

**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 196
  - [":CALibrate:SWITCh"](#) on page 201

## **:CALibrate:STATus**

**N** (see [page 676](#))

**Query Syntax** :CALibrate:STATus?

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

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

**See Also** • ["Introduction to :CALibrate Commands"](#) on page 196



## :CALibrate:SWITCh

**N** (see [page 676](#))

**Query Syntax** :CALibrate:SWITCh?

The :CALibrate:SWITCh? query returns the rear-panel calibration protect (CAL PROTECT) switch 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 196

## :CALibrate:TEMPerature

**N** (see [page 676](#))

**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 196

**:CALibrate:TIME**

**N** (see [page 676](#))

**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 196

## :CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See ["Introduction to :CHANnel<n> Commands"](#) on page 205.

**Table 53** :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit { {0   OFF}   {1   ON} } (see <a href="#">page 207</a> )	:CHANnel<n>:BWLimit? (see <a href="#">page 207</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see <a href="#">page 208</a> )	:CHANnel<n>:COUPling? (see <a href="#">page 208</a> )	<coupling> ::= {AC   DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay { {0   OFF}   {1   ON} } (see <a href="#">page 209</a> )	:CHANnel<n>:DISPlay? (see <a href="#">page 209</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see <a href="#">page 210</a> )	:CHANnel<n>:IMPedance ? (see <a href="#">page 210</a> )	<impedance> ::= {ONEMeg   FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert { {0   OFF}   {1   ON} } (see <a href="#">page 211</a> )	:CHANnel<n>:INVert? (see <a href="#">page 211</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see <a href="#">page 212</a> )	:CHANnel<n>:LABel? (see <a href="#">page 212</a> )	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see <a href="#">page 213</a> )	:CHANnel<n>:OFFSet? (see <a href="#">page 213</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 214</a> )	:CHANnel<n>:PROBe? (see <a href="#">page 214</a> )	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see <a href="#">page 215</a> )	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKE W <skew_value> (see <a href="#">page 216</a> )	:CHANnel<n>:PROBe:SKE W? (see <a href="#">page 216</a> )	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format

**Table 53** :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:STYPe <signal type> (see <a href="#">page 217</a> )	:CHANnel<n>:PROBe:STYPe? (see <a href="#">page 217</a> )	<signal type> ::= {DIFFerential   SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see <a href="#">page 218</a> )	:CHANnel<n>:PROTection? (see <a href="#">page 218</a> )	{NORM   TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see <a href="#">page 219</a> )	:CHANnel<n>:RANGe? (see <a href="#">page 219</a> )	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see <a href="#">page 220</a> )	:CHANnel<n>:SCALe? (see <a href="#">page 220</a> )	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V   mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see <a href="#">page 221</a> )	:CHANnel<n>:UNITs? (see <a href="#">page 221</a> )	<units> ::= {VOLT   AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 222</a> )	:CHANnel<n>:VERNier? (see <a href="#">page 222</a> )	{0   1} <n> ::= 1-2 or 1-4 in NR1 format

### Introduction to :CHANnel<n> Commands

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

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

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

A label command provides identifying annotations of up to 6 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 676](#))

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

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

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

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

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

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

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

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

<bwlimit> ::= {1 | 0}

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205

**:CHANnel<n>:COUPling**(see [page 676](#))**Command Syntax** :CHANnel<n>:COUPling <coupling>

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

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

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

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

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

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

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205



**:CHANnel<n>:DISPlay**

**C** (see [page 676](#))

**Command Syntax** :CHANnel<n>:DISPlay <display value>

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

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

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

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

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

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

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

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 205
  - [":VIEW"](#) on page 171
  - [":BLANK"](#) on page 143
  - [":STATus"](#) on page 168
  - [":POD<n>:DISPlay"](#) on page 334
  - [":DIGital<n>:DISPlay"](#) on page 225

**:CHANnel<n>:IMPedance**

**C** (see [page 676](#))

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

<impedance> ::= {ONEMeg | FIFTy}

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

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

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

**NOTE**

The analog channel input impedance of the 100 MHz bandwidth oscilloscope models is fixed at ONEMeg (1 M $\Omega$ ).

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

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

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205

**:CHANnel<n>:INVert**

**N** (see [page 676](#))

**Command Syntax** :CHANnel<n>:INVert <invert value>

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

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

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

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

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

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

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205

**:CHANnel<n>:LABel**

**N** (see [page 676](#))

**Command Syntax**    :CHANnel<n>:LABel <string>

                         <string> ::= quoted ASCII string

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

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

**NOTE**

Label strings are six characters or less, and may contain any commonly used ASCII characters. Labels with more than 6 characters are truncated to six 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 205
  - [":DISPlay:LABel"](#) on page 235
  - [":DIGital<n>:LABel"](#) on page 226
  - [":DISPlay:LABList"](#) on page 236
  - [":BUS<n>:LABel"](#) on page 194

**Example Code**

```
' LABEL - This command allows you to write a name (six 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 channel1 "C
AL 1".
myScope.WriteString ":CHANNEL2:LABEL " "CAL2" "" ' Label channel1 "CA
L2".
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:CHANnel<n>:OFFSet**

**C** (see [page 676](#))

**Command Syntax** :CHANnel<n>:OFFSet <offset> [<suffix>]

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

<suffix> ::= {V | mV}

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

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

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

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

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

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

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 205
  - [":CHANnel<n>:RANGe"](#) on page 219
  - [":CHANnel<n>:SCALE"](#) on page 220
  - [":CHANnel<n>:PROBe"](#) on page 214

**:CHANnel<n>:PROBe**

**C** (see [page 676](#))

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

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

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

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

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

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

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

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

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

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

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 205
  - [":CHANnel<n>:RANGe"](#) on page 219
  - [":CHANnel<n>:SCALE"](#) on page 220
  - [":CHANnel<n>:OFFSet"](#) on page 213

**Example Code**

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
' channel. The probe attenuation factor may be set from 0.1 to 1000.
myScope.WriteString ":CHAN1:PROBE 10" ' Set Probe to 10:1.
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:CHANnel<n>:PROBe:ID**

**C** (see [page 676](#))

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

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

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

The :CHANnel<n>:PROBe:ID? query returns the type of probe attached to the specified oscilloscope channel.

**Return Format** <probe id><NL>

<probe id> ::= unquoted ASCII string up to 11 characters

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205

**:CHANnel<n>:PROBe:SKEW**(see [page 676](#))

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

The :CHANnel<n>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. Each analog channel can be adjusted + or -100 ns for a total of 200 ns difference between channels. You can use the oscilloscope's probe skew control to remove cable-delay errors between channels.

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

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

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

**See Also**            • ["Introduction to :CHANnel<n> Commands"](#) on page 205



**:CHANnel<n>:PROBe:STYPe**(see [page 676](#))**Command Syntax****NOTE**

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

```
:CHANnel<n>:PROBe:STYPe <signal type>
<signal type> ::= {DIFFerential | SINGLE}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

When single-ended is selected, the :CHANnel<n>:OFFset command changes the offset value of the probe amplifier. When differential is selected, the :CHANnel<n>:OFFset command changes the offset value of the channel amplifier.

**Query Syntax**

:CHANnel&lt;n&gt;: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 205
- [":CHANnel<n>:OFFSet"](#) on page 213

**:CHANnel<n>:PROTection**

**N** (see [page 676](#))

**Command Syntax** :CHANnel<n>:PROTection[:CLEar]

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

When the analog channel input impedance is set to 50Ω (on the 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the input channels are protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the channel is automatically changed to 1 MΩ. The :CHANnel<n>:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the channel to be used again in 50Ω mode after the signal that caused the overload has been removed from the channel input. Reset the analog channel input impedance to 50Ω (see [":CHANnel<n>:IMPedance"](#) on page 210) after clearing the overvoltage protection.

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

The :CHANnel<n>:PROTection query returns the state of the input protection for CHANnel<n>. If a channel input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

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

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 205
  - [":CHANnel<n>:COUPling"](#) on page 208
  - [":CHANnel<n>:IMPedance"](#) on page 210
  - [":CHANnel<n>:PROBe"](#) on page 214

**:CHANnel<n>:RANGe**

**C** (see [page 676](#))

**Command Syntax** :CHANnel<n>:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

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

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

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

- 8 mV to 40 V for the 100 MHz models.
- 16 mV to 8 V for the 300 MHz – 1 GHz models with the input impedance set to 50Ω

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 205
  - [":CHANnel<n>:SCALE"](#) on page 220
  - [":CHANnel<n>:PROBe"](#) on page 214

**Example Code**

```
' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
' range value is 8 times the volts per division.
myScope.WriteString ":CHANNEL1:RANGE 8"    ' Set the vertical range to
8 volts.
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:CHANnel<n>:SCALe**

**N** (see [page 676](#))

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

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

<suffix> ::= {V | mV}

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

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

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

- 1 mV to 5 V for the 100 MHz models.
- 2 mV to 1 V for the 300 MHz – 1 GHz models with the input impedance set to 50Ω.

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 205
  - [":CHANnel<n>:RANGe"](#) on page 219
  - [":CHANnel<n>:PROBe"](#) on page 214

**:CHANnel<n>:UNITs**

**N** (see [page 676](#))

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

<units> ::= {VOLT | AMPere}

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

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

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

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

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

**Return Format** <units><NL>

<units> ::= {VOLT | AMP}

- See Also**
- ["Introduction to :CHANnel<n> Commands"](#) on page 205
  - [":CHANnel<n>:RANGe"](#) on page 219
  - [":CHANnel<n>:PROBe"](#) on page 214
  - [":EXternal:UNITs"](#) on page 249

**:CHANnel<n>:VERNier**

**N** (see [page 676](#))

**Command Syntax** :CHANnel<n>:VERNier <vernier value>

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

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

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

The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

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

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

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

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

**See Also** • ["Introduction to :CHANnel<n> Commands"](#) on page 205

## :DIGital<n> Commands

Control all oscilloscope functions associated with individual digital channels. See ["Introduction to :DIGital<n> Commands"](#) on page 223.

**Table 54** :DIGital<n> Commands Summary

Command	Query	Options and Query Returns
:DIGital<n>:DISPlay { {0   OFF}   {1   ON} } (see <a href="#">page 225</a> )	:DIGital<n>:DISPlay? (see <a href="#">page 225</a> )	{0   1} <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:LABel <string> (see <a href="#">page 226</a> )	:DIGital<n>:LABel? (see <a href="#">page 226</a> )	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:POSition <position> (see <a href="#">page 227</a> )	:DIGital<n>:POSition? (see <a href="#">page 227</a> )	<n> ::= 0-15; an integer in NR1 format <position> ::= 0-7 if display size = large, 0-15 if size = medium, 0-31 if size = small
:DIGital<n>:SIZE <value> (see <a href="#">page 228</a> )	:DIGital<n>:SIZE? (see <a href="#">page 228</a> )	<value> ::= {SMALl   MEDium   LARGe}
:DIGital<n>:THReshold <value>[suffix] (see <a href="#">page 229</a> )	:DIGital<n>:THReshold? (see <a href="#">page 229</a> )	<n> ::= 0-15; an integer 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<n> Commands**

<n> ::= {0,...,15}

The DIGital subsystem commands control the viewing, labeling, and positioning of digital channels. They also control threshold settings for groups of digital channels (D0-D7, D8-D15).

### NOTE

These commands are only valid for the MSO models.

### Reporting the Setup

Use :DIGital<n>? 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<n>:DISPlay**

**N** (see [page 676](#))

**Command Syntax** :DIGital<n>:DISPlay <display>

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

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

The :DIGital<n>: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<n>:DISPlay?

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

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

<display> ::= {0 | 1}

- See Also**
- "[Introduction to :DIGital<n> Commands](#)" on page 223
  - "[:POD<n>:DISPlay](#)" on page 334
  - "[:CHANnel<n>:DISPlay](#)" on page 209
  - "[:VIEW](#)" on page 171
  - "[:BLANK](#)" on page 143
  - "[:STATus](#)" on page 168

**:DIGital<n>:LABel**

**N** (see [page 676](#))

**Command Syntax** :DIGital<n>:LABel <string>

<string> ::= any series of 6 or less characters as quoted ASCII string.

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

The :DIGital<n>: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 six characters or less, and may contain any commonly used ASCII characters. Labels with more than 6 characters are truncated to six characters.

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

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

**Return Format** <label string><NL>

<label string> ::= any series of 6 or less characters as a quoted ASCII string.

- See Also**
- "[Introduction to :DIGital<n> Commands](#)" on page 223
  - "[:CHANnel<n>:LABel](#)" on page 212
  - "[:DISPlay:LABList](#)" on page 236
  - "[:BUS<n>:LABel](#)" on page 194

**:DIGital<n>:POSition**

**N** (see [page 676](#))

**Command Syntax** :DIGital<n>:POSition <position>

<position> ::= integer in NR1 format.

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

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

The :DIGital<n>:POSition command sets the position of the specified channel.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<n>:POSition?

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

**Return Format** <position><NL>

<position> ::= integer in NR1 format.

**See Also** • ["Introduction to :DIGital<n> Commands"](#) on page 223

**:DIGital<n>:SIZE**

**N** (see [page 676](#))

**Command Syntax** :DIGital<n>:SIZE <value>

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

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

The :DIGital<n>: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 channels 1 through 15 as well.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :DIGital<n>:SIZE?

The :DIGital<n>: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<n> Commands"](#) on page 223
  - [":POD<n>:SIZE"](#) on page 335
  - [":DIGital<n>:POSition"](#) on page 227

**:DIGital<n>:THReshold**

**N** (see [page 676](#))

**Command Syntax** :DIGital<n>:THReshold <value>

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

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

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

<n> ::= An integer, 0, 1,...,15, is attached as a suffix to the command and defines the logic channel that is affected by the command.

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

The :DIGital<n>:THReshold command sets the logic threshold value for all channels grouped with the specified channel (D0-D7, D8-D15). 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<n>:THReshold?

The :DIGital<n>: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<n> Commands](#)" on page 223
- "[:POD<n>:THReshold](#)" on page 336
- "[:TRIGger\[:EDGE\]:LEVel](#)" on page 439

## :DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See ["Introduction to :DISPlay Commands"](#) on page 230.

**Table 55** :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLear (see <a href="#">page 232</a> )	n/a	n/a
:DISPlay:DATA [<format>][,][<area>] [,][<palette>]<displa y data> (see <a href="#">page 233</a> )	:DISPlay:DATA? [<format>][,][<area>] [,][<palette>] (see <a href="#">page 233</a> )	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF   BMP   BMP8bit   PNG} (query) <area> ::= {GRATicule   SCReen} (query) <palette> ::= {MONochrome   GRAYscale   COLor} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0   OFF}   {1   ON}} (see <a href="#">page 235</a> )	:DISPlay:LABel? (see <a href="#">page 235</a> )	{0   1}
:DISPlay:LABList <binary block> (see <a href="#">page 236</a> )	:DISPlay:LABList? (see <a href="#">page 236</a> )	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see <a href="#">page 237</a> )	:DISPlay:PERsistence? (see <a href="#">page 237</a> )	<value> ::= {MINimum   INFinite}}
:DISPlay:SOURce <value> (see <a href="#">page 238</a> )	:DISPlay:SOURce? (see <a href="#">page 238</a> )	<value> ::= {PMEMory{0   1   2   3   4   5   6   7   8   9}}
:DISPlay:VECTors {{1   ON}   {0   OFF}} (see <a href="#">page 239</a> )	:DISPlay:VECTors? (see <a href="#">page 239</a> )	{1   0}

### Introduction to :DISPlay Commands

The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.

- Set waveform persistence.
- Specify labels.
- Save and Recall display data.

#### Reporting the Setup

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

#### Return Format

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

```
:DISP:LAB 0;CONN 1;PERS MIN;SOUR PMEM9
```

## :DISPlay:CLEar

**N** (see [page 676](#))

**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 230
  - [":CDISplay"](#) on page 144



**:DISPlay:DATA**

**N** (see [page 676](#))

**Command Syntax** :DISPlay:DATA [<format>][,]<area>[,]<palette><display data>

<format> ::= {TIFF}

<area> ::= {GRATicule}

<palette> ::= {MONochrome}

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

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

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

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

**Query Syntax** :DISPlay:DATA? [<format>][,]<area>[,]<palette>

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

<area> ::= {GRATicule | SCReen}

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

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

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

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

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

**NOTE**

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

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

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

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 230
  - [":DISPlay:SOURce"](#) on page 238
  - [":HARDcopy:INKSaver"](#) on page 273
  - [":MERGe"](#) on page 153
  - [":PRINt"](#) on page 164
  - ["\\*RCL \(Recall\)"](#) on page 122
  - ["\\*SAV \(Save\)"](#) on page 126
  - [":VIEW"](#) on page 171

**Example Code**

```
' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1 ' Open file f
or output.
Put #1, , byteData ' Write data.
Close #1 ' Close file.
myScope.IO.Timeout = 5000
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:DISPlay:LABel**

**N** (see [page 676](#))

**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 230  
                  • [":CHANnel<n>:LABel"](#) on page 212

**Example Code**    ' DISP\_LABEL (not executed in this example)  
                          ' - Turns label names ON or OFF on the analyzer display.  
                          myScope.WriteString ":DISPLAY:LABEL ON"    ' Turn on labels.

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:DISPlay:LABList**

**N** (see [page 676](#))

**Command Syntax** :DISPlay:LABList <binary block data>

<binary block> ::= an ordered list of up to 75 labels, a maximum of six 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 "A12345", 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 six characters each, separated by newline characters.

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 230
  - [":DISPlay:LABel"](#) on page 235
  - [":CHANnel<n>:LABel"](#) on page 212
  - [":DIGital<n>:LABel"](#) on page 226

**:DISPlay:PERStence**

**N** (see [page 676](#))

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

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

**Query Syntax**    :DISPlay:PERStence?

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

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 230
  - [":DISPlay:CLEar"](#) on page 232
  - [":CDISplay"](#) on page 144

**:DISPlay:SOURce**

**N** (see [page 676](#))

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

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

```
PMEMory0-9 ::= pixel memory 0 through 9
```

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

**Query Syntax** :DISPlay:SOURce?

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

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

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

- See Also**
- ["Introduction to :DISPlay Commands"](#) on page 230
  - [":DISPlay:DATA"](#) on page 233

**:DISPlay:VECTors**

**N** (see [page 676](#))

**Command Syntax** :DISPlay:VECTors <vectors>

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

The :DISPlay:VECTors command turns vector display on or off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

**Query Syntax** :DISPlay:VECTors?

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

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

<vectors> ::= {1 | 0}

**See Also** • ["Introduction to :DISPlay Commands"](#) on page 230

## :EXternal Trigger Commands

Control the input characteristics of the external trigger input. See ["Introduction to :EXternal Trigger Commands"](#) on page 240.

**Table 56** :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see <a href="#">page 242</a> )	:EXternal:BWLimit? (see <a href="#">page 242</a> )	<bwlimit> ::= {0   OFF}
:EXternal:IMPedance <value> (see <a href="#">page 243</a> )	:EXternal:IMPedance? (see <a href="#">page 243</a> )	<impedance> ::= {ONEMeg   FIFTy}
:EXternal:PROBe <attenuation> (see <a href="#">page 244</a> )	:EXternal:PROBe? (see <a href="#">page 244</a> )	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXternal:PROBe:ID? (see <a href="#">page 245</a> )	<probe id> ::= unquoted ASCII string up to 11 characters
:EXternal:PROBe:SType <signal type> (see <a href="#">page 246</a> )	:EXternal:PROBe:SType? (see <a href="#">page 246</a> )	<signal type> ::= {DIFFerential   SINGLE}
:EXternal:PROTection[ :CLEar] (see <a href="#">page 247</a> )	:EXternal:PROTection? (see <a href="#">page 247</a> )	{NORM   TRIP}
:EXternal:RANge <range>[<suffix>] (see <a href="#">page 248</a> )	:EXternal:RANge? (see <a href="#">page 248</a> )	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V   mV}
:EXternal:UNITs <units> (see <a href="#">page 249</a> )	:EXternal:UNITs? (see <a href="#">page 249</a> )	<units> ::= {VOLT   AMPere}

### Introduction to :EXternal Trigger Commands

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

#### Reporting the Setup

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

#### Return Format



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

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

**:EXternal:BWLimit**

**C** (see [page 676](#))

**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 240
  - ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:HFReject"](#) on page 409

**:EXternal:IMPedance**

**C** (see [page 676](#))

**Command Syntax**    :EXternal:IMPedance <value>  
                          <value> ::= {ONEMeg | FIFTy}

The :EXternal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 M $\Omega$ ) and FIFTy (50 $\Omega$ ).

**NOTE**

You can set external trigger input impedance to FIFTy (50 $\Omega$ ) on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models.

**Query Syntax**    :EXternal:IMPedance?

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

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

- See Also**
- ["Introduction to :EXternal Trigger Commands"](#) on page 240
  - ["Introduction to :TRIGger Commands"](#) on page 405
  - [":CHANnel<n>:IMPedance"](#) on page 210

**:EXternal:PROBe**

**C** (see [page 676](#))

**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 240
  - [":EXternal:RANGe"](#) on page 248
  - ["Introduction to :TRIGger Commands"](#) on page 405
  - [":CHANnel<n>:PROBe"](#) on page 214

**:EXternal:PROBe:ID**

**C** (see [page 676](#))

**Query Syntax** :EXternal:PROBe:ID?

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

**Return Format** <probe id><NL>

<probe id> ::= unquoted ASCII string up to 11 characters

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

**See Also** • ["Introduction to :EXternal Trigger Commands"](#) on page 240

**:EXternal:PROBe:STYPe**

**C** (see [page 676](#))

**Command Syntax****NOTE**

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

```
:EXternal:PROBe:STYPe <signal type>
```

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

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

**Query Syntax** :EXternal:PROBe:STYPe?

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

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

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

**See Also** • ["Introduction to :EXternal Trigger Commands"](#) on page 240

**:EXternal:PROtection**

**N** (see [page 676](#))

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

When the external trigger input impedance is set to 50Ω (on the 2-channel, 300 MHz, 500 MHz, and 1 GHz bandwidth oscilloscope models), the external trigger input is protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the external trigger is automatically changed to 1 MΩ. The :EXternal:PROtection[:CLEar] command is used to clear (reset) the overload protection. It allows the external trigger to be used again in 50Ω mode after the signal that caused the overload has been removed from the external trigger input. Reset the external trigger input impedance to 50Ω (see [":EXternal:IMPedance"](#) on page 243) after clearing the overvoltage protection.

**Query Syntax** :EXternal:PROtection?

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

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

- See Also**
- ["Introduction to :EXternal Trigger Commands"](#) on page 240
  - [":EXternal:IMPedance"](#) on page 243
  - [":EXternal:PROBe"](#) on page 244

**:EXternal:RANGe**

**C** (see [page 676](#))

**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. The range can only be set to 5.0 V when using 1:1 probe attenuation. 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 = (5.0 V)  
) \* (probe attenuation factor)

- See Also**
- ["Introduction to :EXternal Trigger Commands"](#) on page 240
  - [":EXternal:PROBe"](#) on page 244
  - ["Introduction to :TRIGger Commands"](#) on page 405



**:EXternal:UNITs**

**N** (see [page 676](#))

**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 240
  - ["Introduction to :TRIGger Commands"](#) on page 405
  - [":EXternal:RANGe"](#) on page 248
  - [":EXternal:PROBe"](#) on page 244
  - [":CHANnel<n>:UNITs"](#) on page 221

## :FUNCTION Commands

Control functions in the measurement/storage module. See "[Introduction to :FUNCTION Commands](#)" on page 252.

**Table 57** :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTER <frequency> (see <a href="#">page 253</a> )	:FUNCTION:CENTER? (see <a href="#">page 253</a> )	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCTION:DISPLAY {{0   OFF}   {1   ON}} (see <a href="#">page 254</a> )	:FUNCTION:DISPLAY? (see <a href="#">page 254</a> )	{0   1}
:FUNCTION:GOFT:OPERATION <operation> (see <a href="#">page 255</a> )	:FUNCTION:GOFT:OPERATION? (see <a href="#">page 255</a> )	<operation> ::= {ADD   SUBTRACT   MULTIPLY}
:FUNCTION:GOFT:SOURCE 1 <source> (see <a href="#">page 256</a> )	:FUNCTION:GOFT:SOURCE 1? (see <a href="#">page 256</a> )	<source> ::= CHANNEL<n> <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models
:FUNCTION:GOFT:SOURCE 2 <source> (see <a href="#">page 257</a> )	:FUNCTION:GOFT:SOURCE 2? (see <a href="#">page 257</a> )	<source> ::= CHANNEL<n> <n> ::= {{1   2}   {3   4}} for 4ch models, depending on SOURCE1 selection <n> ::= {1   2} for 2ch models
:FUNCTION:OFFSET <offset> (see <a href="#">page 258</a> )	:FUNCTION:OFFSET? (see <a href="#">page 258</a> )	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCTION:OPERATION <operation> (see <a href="#">page 259</a> )	:FUNCTION:OPERATION? (see <a href="#">page 259</a> )	<operation> ::= {ADD   SUBTRACT   MULTIPLY   INTEGRATE   DIFFERENTIATE   FFT   SQRT}

**Table 57** :FUNction Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNction:RANGe <range> (see page 260)	:FUNction:RANGe? (see page 260)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.
:FUNction:REFerence <level> (see page 261)	:FUNction:REFerence? (see page 261)	<level> ::= the current reference level in NR3 format. The range of legal values is from 400.0 dBV to +400.0 dBV (depending on current range value).
:FUNction:SCALe <scale value>[<suffix>] (see page 262)	:FUNction:SCALe? (see page 262)	<scale value> ::= integer in NR1 format <suffix> ::= {V   dB}
:FUNction:SOURce1 <source> (see page 263)	:FUNction:SOURce1? (see page 263)	<source> ::= {CHANnel<n>   GOFT} <n> ::= {1   2   3   4} for 4ch models <n> ::= {1   2} for 2ch models GOFT is only for FFT, INTegrate, DIFFerentiate, and SQRT operations.
:FUNction:SOURce2 <source> (see page 264)	:FUNction:SOURce2? (see page 264)	<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 265)	:FUNction:SPAN? (see page 265)	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNction:WINDow <window> (see page 266)	:FUNction:WINDow? (see page 266)	<window> ::= {RECTangular   HANNing   FLATtop   BHARRis}

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

The SOURce1, DISPlay, RANGE, and OFFSet commands apply to any function. The SPAN, CENTER, and WINDow commands are only useful for FFT functions. When FFT is selected, the cursors change from volts and time to decibels (dB) and frequency (Hz).

### Reporting the Setup

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

### Return Format

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

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

**:FUNCTION:CENTer**

**N** (see [page 676](#))

**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 252
  - ":FUNCTION:SPAN" on page 265
  - ":TIMEbase:RANGe" on page 397
  - ":TIMEbase:SCALE" on page 400

**:FUNCTION:DISPlay****N** (see [page 676](#))

**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 252
  - [":VIEW"](#) on page 171
  - [":BLANK"](#) on page 143
  - [":STATus"](#) on page 168

**:FUNCTION:GOFT:OPERation**

**N** (see [page 676](#))

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

<operation> ::= {ADD | SUBTract | MULTiply}

The :FUNCTION:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, or SQRT functions:

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

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

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

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

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

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:GOFT:SOURce1"](#) on page 256
  - [":FUNCTION:GOFT:SOURce2"](#) on page 257
  - [":FUNCTION:SOURce1"](#) on page 263

**:FUNCTION:GOFT:SOURce1**

**N** (see [page 676](#))

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

The :FUNCTION:GOFT:SOURce1 command selects the first input channel for the g(t) source that can be used as the input to the FFT, INTEGRate, DIFFerentiate, or SQRT functions.

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

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

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:GOFT:SOURce2"](#) on page 257
  - [":FUNCTION:GOFT:OPERation"](#) on page 255



**:FUNCTION:GOFT:SOURce2**

**N** (see [page 676](#))

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

<value> ::= CHANnel<n>

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

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

The :FUNCTION:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to the FFT, INTEGRate, DIFFerentiate, or SQRT functions.

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

**Query Syntax** :FUNCTION:GOFT:SOURce2?

The :FUNCTION:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

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

<value> ::= CHAN<n>

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:GOFT:SOURce1"](#) on page 256
  - [":FUNCTION:GOFT:OPERation"](#) on page 255

**:FUNCTION:OFFSet**

**N** (see [page 676](#))

**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 252
  - [":FUNCTION:RANGe"](#) on page 260
  - [":FUNCTION:REfERENCE"](#) on page 261
  - [":FUNCTION:SCALE"](#) on page 262

**:FUNCTION:OPERation**

**N** (see [page 676](#))

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

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

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

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

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

**Query Syntax**    :FUNCTION:OPERation?

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

**Return Format**    <operation><NL>

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:SOURce1"](#) on page 263
  - [":FUNCTION:SOURce2"](#) on page 264

## :FUNcTion:RANGe

**N** (see [page 676](#))

**Command Syntax** :FUNcTion:RANGe <range>

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

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

**Query Syntax** :FUNcTion:RANGe?

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

**Return Format** <range><NL>

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

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

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

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

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

- See Also**
- ["Introduction to :FUNcTion Commands"](#) on page 252
  - [":FUNcTion:SCALE"](#) on page 262

**:FUNction:REFeRence**

**N** (see [page 676](#))

**Command Syntax** :FUNction:REFeRence <level>

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

The range of legal values is from -400.0 dBV to +400.0 dBV depending on the current :FUNction:RANGe value. If you set the reference level to a value outside of the legal range, it is automatically set to the nearest legal value.

The :FUNction:REFeRence command is only used when an FFT (Fast Fourier Transform) operation is selected. The :FUNction:REFeRence command sets the reference level represented by center screen.

**NOTE**

The FUNction:REFeRence command is equivalent to the :FUNction:OFFSet command.

**Query Syntax** :FUNction:REFeRence?

The :FUNction:REFeRence? query returns the current reference level in dBV.

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

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

- See Also**
- ["Introduction to :FUNction Commands"](#) on page 252
  - [":FUNction:OFFSet"](#) on page 258
  - [":FUNction:RANGe"](#) on page 260
  - [":FUNction:SCALE"](#) on page 262

**:FUNCTION:SCALE**

**N** (see [page 676](#))

**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 252  
                          • [":FUNCTION:RANGE"](#) on page 260

**:FUNCTION:SOURce1**

**N** (see [page 676](#))

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

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

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

**NOTE**

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

**Query Syntax** :FUNCTION:SOURce1?

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

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:OPERation"](#) on page 259
  - [":FUNCTION:GOFT:OPERation"](#) on page 255
  - [":FUNCTION:GOFT:SOURce1"](#) on page 256
  - [":FUNCTION:GOFT:SOURce2"](#) on page 257

**:FUNCTION:SOURce2**

**N** (see [page 676](#))

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

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

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

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

The :FUNCTION:SOURce2 command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCTION:OPERation command for more information about selecting an operation). The :FUNCTION:SOURce2 command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCTION:OPERation command.

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

**Query Syntax** :FUNCTION:SOURce2?

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

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

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

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:OPERation"](#) on page 259



**:FUNCTION:SPAN**

**N** (see [page 676](#))

**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 252
  - [":FUNCTION:CENTer"](#) on page 253
  - [":TIMEbase:RANGe"](#) on page 397
  - [":TIMEbase:SCALE"](#) on page 400

**:FUNCTION:WINDow**

**N** (see [page 676](#))

**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 252

## :HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See ["Introduction to :HARDcopy Commands"](#) on page 267.

**Table 58** :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see <a href="#">page 269</a> )	:HARDcopy:AREA? (see <a href="#">page 269</a> )	<area> ::= SCreen
:HARDcopy:APRinter <active_printer> (see <a href="#">page 270</a> )	:HARDcopy:APRinter? (see <a href="#">page 270</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 271</a> )	:HARDcopy:FACTors? (see <a href="#">page 271</a> )	{0   1}
:HARDcopy:FFEed {{0   OFF}   {1   ON}} (see <a href="#">page 272</a> )	:HARDcopy:FFEed? (see <a href="#">page 272</a> )	{0   1}
:HARDcopy:INKSaver {{0   OFF}   {1   ON}} (see <a href="#">page 273</a> )	:HARDcopy:INKSaver? (see <a href="#">page 273</a> )	{0   1}
:HARDcopy:PALette <palette> (see <a href="#">page 274</a> )	:HARDcopy:PALette? (see <a href="#">page 274</a> )	<palette> ::= {COLor   GRAYscale   NONE}
n/a	:HARDcopy:PRinter:LIS T? (see <a href="#">page 275</a> )	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y   N} <name> ::= name of printer
:HARDcopy:STARt (see <a href="#">page 276</a> )	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 0;PAL NONE
```

**:HARDcopy:AREA**

**N** (see [page 676](#))

**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 267
  - "[:HARDcopy:START](#)" on page 276
  - "[:HARDcopy:APRinter](#)" on page 270
  - "[:HARDcopy:PRinter:LIST](#)" on page 275
  - "[:HARDcopy:FACTors](#)" on page 271
  - "[:HARDcopy:FFEed](#)" on page 272
  - "[:HARDcopy:INKSaver](#)" on page 273
  - "[:HARDcopy:PALETTE](#)" on page 274

## :HARDcopy:APRinter

**N** (see [page 676](#))

**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 267
  - [":HARDcopy:PRinter:LIST"](#) on page 275
  - [":HARDcopy:START"](#) on page 276

**:HARDcopy:FACTors**

**N** (see [page 676](#))

**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 267
  - [":HARDcopy:STArt"](#) on page 276
  - [":HARDcopy:FFeEd"](#) on page 272
  - [":HARDcopy:INKSaver"](#) on page 273
  - [":HARDcopy:PALETTE"](#) on page 274

**:HARDcopy:FFeEd**

**N** (see [page 676](#))

**Command Syntax**    :HARDcopy:FFeEd <ffeed>  
                          <ffeed> ::= {{OFF | 0} | {ON | 1}}

The HARDcopy:FFeEd command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

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

**Query Syntax**       :HARDcopy:FFeEd?

The :HARDcopy:FFeEd? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

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

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 267
  - [":HARDcopy:STArT"](#) on page 276
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:INKSaver"](#) on page 273
  - [":HARDcopy:PALette"](#) on page 274



**:HARDcopy:INKSaver**

**N** (see [page 676](#))

**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 267
  - [":HARDcopy:STArt"](#) on page 276
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:FFEed"](#) on page 272
  - [":HARDcopy:PALETTE"](#) on page 274

**:HARDcopy:PALETTE**

**N** (see [page 676](#))

**Command Syntax** :HARDcopy:PALETTE <palette>  
 <palette> ::= {COLOR | GRAYscale | NONE}

The HARDcopy:PALETTE command sets the hardcopy palette color.

**NOTE**

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

**Query Syntax** :HARDcopy:PALETTE?

The :HARDcopy:PALETTE? query returns the selected hardcopy palette color.

**Return Format** <palette><NL>  
 <palette> ::= {COL | GRAY | NONE}

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 267
  - [":HARDcopy:START"](#) on page 276
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:FFEd"](#) on page 272
  - [":HARDcopy:INKSaver"](#) on page 273

**:HARDcopy:PRinter:LIST**

**N** (see [page 676](#))

**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 267
  - [":HARDcopy:APRinter"](#) on page 270
  - [":HARDcopy:START"](#) on page 276

## :HARDcopy:STARt

**N** (see [page 676](#))

**Command Syntax** :HARDcopy:STARt

The :HARDcopy:STARt command starts a print job.

- See Also**
- ["Introduction to :HARDcopy Commands"](#) on page 267
  - [":HARDcopy:APRinter"](#) on page 270
  - [":HARDcopy:PRinter:LIST"](#) on page 275
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:FFEed"](#) on page 272
  - [":HARDcopy:INKSaver"](#) on page 273
  - [":HARDcopy:PALETTE"](#) on page 274

## :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 278.

**Table 59** :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see <a href="#">page 279</a> )	:MARKer:MODE? (see <a href="#">page 279</a> )	<mode> ::= {OFF   MEASurement   MANUAL}
:MARKer:X1Position <position>[suffix] (see <a href="#">page 280</a> )	:MARKer:X1Position? (see <a href="#">page 280</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 281</a> )	:MARKer:X1Y1source? (see <a href="#">page 281</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see <a href="#">page 282</a> )	:MARKer:X2Position? (see <a href="#">page 282</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 283</a> )	:MARKer:X2Y2source? (see <a href="#">page 283</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see <a href="#">page 284</a> )	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see <a href="#">page 285</a> )	:MARKer:Y1Position? (see <a href="#">page 285</a> )	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V   mV   dB} <return_value> ::= Y1 cursor position value in NR3 format

**Table 59** :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:Y2Position <position>[suffix] (see <a href="#">page 286</a> )	:MARKer:Y2Position? (see <a href="#">page 286</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 287</a> )	<return_value> ::= Y cursors delta value in NR3 format

**Introduction to :MARKer Commands** The MARKer subsystem commands set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). You can set and query the marker mode and source, the position of the X and Y cursors, and query delta X and delta Y cursor values.

#### Reporting the Setup

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

#### Return Format

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

```
:MARK:X1Y1 NONE;X2Y2 NONE;MODE OFF
```

**:MARKer:MODE**

**N** (see [page 676](#))

**Command Syntax**    :MARKer:MODE <mode>  
                          <mode> ::= {OFF | MEASurement | MANual}

The :MARKer:MODE command sets the cursors mode. OFF removes the cursor information from the display. MANual mode enables manual placement of the X and Y cursors. In MEASurement mode the cursors track the most recent measurement.

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.

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.

**Query Syntax**    :MARKer:MODE?

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

**Return Format**    <mode><NL>  
                          <mode> ::= {OFF | MEAS | MAN}

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MEASure:SOURce"](#) on page 315

**:MARKer:X1Position**

**N** (see [page 676](#))

**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, sets the X1 cursor position and moves the X1 cursor to the specified value.

**Query Syntax** :MARKer:X1Position?

The :MARKer:X1Position? query returns the current X1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

**Return Format** <position><NL>

<position> ::= X1 cursor position in NR3 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 278
  - "[:MARKer:MODE](#)" on page 279
  - "[:MARKer:X2Position](#)" on page 282
  - "[:MARKer:X1Y1source](#)" on page 281
  - "[:MARKer:X2Y2source](#)" on page 283
  - "[:MEASure:TSTArt](#)" on page 618



**:MARKer:X1Y1source**

**N** (see [page 676](#))

**Command Syntax**    :MARKer:X1Y1source <source>

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode (see [":MARKer:MODE"](#) on page 279).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCTION, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

**NOTE**

MATH is an alias for FUNCTION. The query will return FUNC if the source is FUNCTION or MATH.

**Query Syntax**    :MARKer:X1Y1source?

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format**    <source><NL>

<source> ::= {CHAN<n> | FUNC | NONE}

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MEASure:SOURce"](#) on page 315

**:MARKer:X2Position**

**N** (see [page 676](#))

**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, sets the X2 cursor position and moves the X2 cursor to the specified value.

**Query Syntax** :MARKer:X2Position?

The :MARKer:X2Position? query returns current X2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

**Return Format** <position><NL>

<position> ::= X2 cursor position in NR3 format

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Position"](#) on page 280
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MEASure:TSTOp"](#) on page 619

**:MARKer:X2Y2source**

**N** (see [page 676](#))

**Command Syntax** :MARKer:X2Y2source <source>

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a MARKer:X2Y2source command puts the cursors in the MANual mode (see [":MARKer:MODE"](#) on page 279).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCTION, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

**NOTE**

MATH is an alias for FUNCTION. The query will return FUNC if the source is FUNCTION or MATH.

**Query Syntax** :MARKer:X2Y2source?

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

**Return Format** <source><NL>

<source> ::= {CHAN<n> | FUNC | NONE}

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MEASure:SOURce"](#) on page 315

**:MARKer:XDELta**

**N** (see [page 676](#))

**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 or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual 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 278
  - "[:MARKer:MODE](#)" on page 279
  - "[:MARKer:X1Position](#)" on page 280
  - "[:MARKer:X2Position](#)" on page 282
  - "[:MARKer:X1Y1source](#)" on page 281
  - "[:MARKer:X2Y2source](#)" on page 283

**:MARKer:Y1Position**

**N** (see [page 676](#))

**Command Syntax** :MARKer:Y1Position <position> [suffix]  
 <position> ::= Y1 cursor position in NR3 format  
 <suffix> ::= {mV | V | dB}

The :MARKer:Y1Position command sets :MARKer:MODE to MANual, sets the Y1 cursor position and moves the Y1 cursor to the specified value.

**Query Syntax** :MARKer:Y1Position?

The :MARKer:Y1Position? query returns current Y1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query

**Return Format** <position><NL>  
 <position> ::= Y1 cursor position in NR3 format

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MARKer:Y2Position"](#) on page 286
  - [":MEASure:VSTArt"](#) on page 624

**:MARKer:Y2Position**

**N** (see [page 676](#))

**Command Syntax** :MARKer:Y2Position <position> [suffix]  
 <position> ::= Y2 cursor position in NR3 format  
 <suffix> ::= {mV | V | dB}

The :MARKer:Y2Position command sets :MARKer:MODE to MANual, sets the Y2 cursor position and moves the Y2 cursor to the specified value.

**Query Syntax** :MARKer:Y2Position?

The :MARKer:Y2Position? query returns current Y2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

**Return Format** <position><NL>  
 <position> ::= Y2 cursor position in NR3 format

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MARKer:Y1Position"](#) on page 285
  - [":MEASure:VSTOp"](#) on page 625

**:MARKer:YDELta**

**N** (see [page 676](#))

**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 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 278
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MARKer:Y1Position"](#) on page 285
  - [":MARKer:Y2Position"](#) on page 286

## :MEASure Commands

Select automatic measurements to be made and control time markers. See ["Introduction to :MEASure Commands"](#) on page 293.

**Table 60** :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLear (see <a href="#">page 295</a> )	n/a	n/a
:MEASure:COUNter [<source>] (see <a href="#">page 296</a> )	:MEASure:COUNter? [<source>] (see <a href="#">page 296</a> )	<source> ::= {CHANnel<n>   EXternal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXternal} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DElay, <delay spec> (see <a href="#">page 297</a> )	:MEASure:DEFine? DElay (see <a href="#">page 298</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 297</a> )	:MEASure:DEFine? THResholds (see <a href="#">page 298</a> )	<threshold spec> ::= {STANdard}   {<threshold mode>,<upper>,<middle>,<lower>} <threshold mode> ::= {PERCent   ABSolute}
:MEASure:DElay [<source1>] [,<source2>] (see <a href="#">page 300</a> )	:MEASure:DElay? [<source1>] [,<source2>] (see <a href="#">page 300</a> )	<source1,2> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see <a href="#">page 302</a> )	:MEASure:DUTYcycle? [<source>] (see <a href="#">page 302</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNCTION   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format



**Table 60** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 303)	:MEASure:FALLtime? [<source>] (see page 303)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 304)	:MEASure:FREQuency? [<source>] (see page 304)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 305)	:MEASure:NWIDth? [<source>] (see page 305)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 306)	:MEASure:OVERshoot? [<source>] (see page 306)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 308)	:MEASure:PERiod? [<source>] (see page 308)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format

**Table 60** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PHASe [<source1>] [,<source2>] (see page 309)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 309)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 310)	:MEASure:PREShoot? [<source>] (see page 310)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 311)	:MEASure:PWIDth? [<source>] (see page 311)	<source> ::= {CHANnel<n>   FUNctIon   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISEtime [<source>] (see page 312)	:MEASure:RISEtime? [<source>] (see page 312)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 313)	:MEASure:SDEViation? [<source>] (see page 313)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1   ON} (see page 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   EXTErnal} for DSO models <source1,2> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctIon   MATH   EXTErnal} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= {<source>   NONE}

**Table 60** :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} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctioN   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see <a href="#">page 319</a> )	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n>   FUNctioN   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNctioN   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPlitude [<source>] (see <a href="#">page 321</a> )	:MEASure:VAMPlitude? [<source>] (see <a href="#">page 321</a> )	<source> ::= {CHANnel<n>   FUNctioN   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAverage [<source>] (see <a href="#">page 322</a> )	:MEASure:VAverage? [<source>] (see <a href="#">page 322</a> )	<source> ::= {CHANnel<n>   FUNctioN   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format

**Table 60** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VBASe [<source>] (see page 323)	:MEASure:VBASe? [<source>] (see page 323)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 324)	:MEASure:VMAX? [<source>] (see page 324)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 325)	:MEASure:VMIN? [<source>] (see page 325)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 326)	:MEASure:VPP? [<source>] (see page 326)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 309)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 327)	<source1,2> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 328)	:MEASure:VRMS? [<source>] (see page 328)	<source> ::= {CHANnel<n>   FUNctIon   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format

**Table 60** :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:VTIME? <vtime>[,<source>] (see <a href="#">page 329</a> )	<vtime> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n>   FUNCTION   MATH} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   FUNCTION   MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see <a href="#">page 330</a> )	:MEASure:VTOP? [<source>] (see <a href="#">page 330</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:XMAX [<source>] (see <a href="#">page 331</a> )	:MEASure:XMAX? [<source>] (see <a href="#">page 331</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see <a href="#">page 332</a> )	:MEASure:XMIN? [<source>] (see <a href="#">page 332</a> )	<source> ::= {CHANnel<n>   FUNCTION   MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

**Introduction to :MEASure Commands** The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

#### Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

### Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

### Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the delayed time base mode (:TIMEbase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the delayed 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,NONE
```

**:MEASure:CLEar**

**N** (see [page 676](#))

**Command Syntax** :MEASure:CLEar

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

**See Also** • ["Introduction to :MEASure Commands"](#) on page 293

**:MEASure:COUNter**

**N** (see [page 676](#))

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

<source> ::= {<digital channels> | CHANnel<n> | EXTernal}

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models

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

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

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

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

The Y cursor shows the the edge threshold level used in the measurement.

Only one counter measurement may be displayed at a time.

**NOTE**

This command is not available if the source is MATH.

**Query Syntax** :MEASure:COUNter? [<source>]

The :MEASure:COUNter? query measures and outputs the counter frequency of the specified source.

**NOTE**

The :MEASure:COUNter? query times out if the counter measurement is installed on the front panel. Use :MEASure:CLEar to remove the front-panel measurement before executing the :MEASure:COUNter? query.

**Return Format** <source><NL>

<source> ::= count in Hertz in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 293
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:FREQuency](#)" on page 304
  - "[:MEASure:CLEar](#)" on page 295



**:MEASure:DEFine**

**N** (see [page 676](#))

**Command Syntax** :MEASure:DEFine <meas\_spec>  
 <meas\_spec> ::= {DELay | THResholds}

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DELay	THResholds
DUTYcycle		x
DELay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PREShoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

**:MEASure:DEFine DELay Command Syntax** :MEASure:DEFine DELay,<delay\_spec>  
 <delay\_spec> ::= <edge\_spec1>,<edge\_spec2>  
 <edge\_spec1> ::= [<slope>]<occurrence>  
 <edge\_spec2> ::= [<slope>]<occurrence>  
 <slope> ::= {+ | -}  
 <occurrence> ::= integer

This command defines the behavior of the :MEASure:DElay? query by specifying the start and stop edge to be used. <edge\_spec1> specifies the slope and edge number on source1. <edge\_spec2> specifies the slope and edge number on source2. The measurement is taken as:

$$\text{delay} = t(\text{<edge\_spec2>}) - t(\text{<edge\_spec1>})$$
**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 220: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 &lt;meas\_spec&gt; = THResholds and &lt;threshold mode&gt; = 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 &lt;meas\_spec&gt; = THResholds and &lt;threshold mode&gt; = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>
```

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and lower threshold voltages in NR3 format.

## for &lt;threshold spec&gt; = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:DELay"](#) on page 300
  - [":MEASure:SOURce"](#) on page 315
  - [":CHANnel<n>:RANGe"](#) on page 219
  - [":CHANnel<n>:SCALe"](#) on page 220
  - [":CHANnel<n>:PROBe"](#) on page 214
  - [":CHANnel<n>:UNITs"](#) on page 221

**:MEASure:DElay**

**N** (see [page 676](#))

**Command Syntax** :MEASure:DElay [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNction | MATH}

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

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

The :MEASure:DElay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

$$\text{delay} = t(\text{<edge spec 2>}) - t(\text{<edge spec 1>})$$

where the <edge spec> definitions are set by the :MEASure:DEFine command

**NOTE**

The :MEASure:DElay command and the front-panel delay measurement differ from the :MEASure:DElay? query.

The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DElay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

**Query Syntax** :MEASure:DElay? [<source1>][,<source2>]

The :MEASure:DElay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also in the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and

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 293
  - [":MEASure:DEFine"](#) on page 297
  - [":MEASure:PHASe"](#) on page 309

**:MEASure:DUTYcycle**

**C** (see [page 676](#))

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

<source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH}

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models

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

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

The :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 293
  - [":MEASure:PERiod"](#) on page 308
  - [":MEASure:PWIDth"](#) on page 311
  - [":MEASure:SOURce"](#) on page 315

**Example Code** • ["Example Code"](#) on page 316

**:MEASure:FALLtime**

**C** (see [page 676](#))

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

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:FALLtime? [<source>]

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

$$\text{fall time} = \text{time at lower threshold} - \text{time at upper threshold}$$

**Return Format** <value><NL>  
 <value> ::= time in seconds between the lower threshold and upper threshold in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:RISetime"](#) on page 312
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:FREQuency**

**C** (see [page 676](#))

**Command Syntax** :MEASure:FREQuency [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH}  
 <digital channels> ::= DIGital0,...,DIGital15 for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :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 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:PERiod"](#) on page 308

**Example Code** • ["Example Code"](#) on page 316



**:MEASure:NWIDth**

**C** (see [page 676](#))

**Command Syntax** :MEASure:NWIDth [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH}  
 <digital channels> ::= DIGital0,...,DIGital15 for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:NWIDth? [<source>]

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

**Return Format** <value><NL>  
 <value> ::= negative pulse width in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:PWIDth"](#) on page 311
  - [":MEASure:PERiod"](#) on page 308

**:MEASure:OVERshoot**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:OVERshoot? [<source>]

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{overshoot} = ((V_{\text{max}} - V_{\text{top}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

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 293
  - [":MEASure:PREShoot"](#) on page 310
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324

- `":MEASure:VTOP"` on page 330
- `":MEASure:VBASe"` on page 323
- `":MEASure:VMIN"` on page 325

**:MEASure:PERiod**

**C** (see [page 676](#))

**Command Syntax** :MEASure:PERiod [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH}  
 <digital channels> ::= DIGital0,...,DIGital15 for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :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 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:NWIDth"](#) on page 305
  - [":MEASure:PWIDth"](#) on page 311
  - [":MEASure:FREQuency"](#) on page 304

**Example Code** • ["Example Code"](#) on page 316

**:MEASure:PHASe**

**N** (see [page 676](#))

**Command Syntax** :MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNction | MATH}

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

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

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

**Query Syntax** :MEASure:PHASe? [<source1>][,<source2>]

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELAy for more detail on selecting the 2nd edge.

The phase is calculated as follows:

$$\text{phase} = (\text{delay} / \text{period of input 1}) \times 360$$

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

<value> ::= the phase angle value in degrees in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:DELAy"](#) on page 300
  - [":MEASure:PERiod"](#) on page 308
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:PREShoot**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:PREShoot? [<source>]

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

$$\text{preshoot} = ((V_{\text{min}} - V_{\text{base}}) / (V_{\text{top}} - V_{\text{base}})) \times 100$$

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 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VTOP"](#) on page 330
  - [":MEASure:VBASe"](#) on page 323

**:MEASure:PWIDth**

**C** (see [page 676](#))

**Command Syntax** :MEASure:PWIDth [<source>]  
 <source> ::= {<digital channels> | CHANnel<n> | FUNction | MATH}  
 <digital channels> ::= DIGital0,...,DIGital15 for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :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 293
  - ":MEASure:SOURce" on page 315
  - ":MEASure:NWIDth" on page 305
  - ":MEASure:PERiod" on page 308

**:MEASure:RISetime**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure: RISetime? [<source>]

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

$$\text{rise time} = \text{time at upper threshold} - \text{time at lower threshold}$$

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

<value> ::= rise time in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:FALLtime"](#) on page 303



**:MEASure:SDEVIation**

**N** (see [page 676](#))

**Command Syntax** :MEASure:SDEVIation [<source>]  
 <source> ::= {CHANnel<n> | FUNCTION | MATH}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models  
 The :MEASure:SDEVIation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:SDEVIation? [<source>]

The :MEASure:SDEVIation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

**Return Format** <value><NL>  
 <value> ::= calculated std deviation value in NR3 format

**See Also**

- ["Introduction to :MEASure Commands"](#) on page 293
- [":MEASure:SOURce"](#) on page 315

### :MEASure:SHOW

**N** (see [page 676](#))

**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 293

**:MEASure:SOURce**

**C** (see [page 676](#))

**Command Syntax** :MEASure:SOURce <source1>[,<source2>]

<source1>,<source2> ::= {<digital channels> | CHANnel<n> | FUNction  
| MATH | EXtErnal}

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models

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

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

The :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 | EXT  
| NONE}

- See Also:**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MEASure:DElay"](#) on page 300
  - [":MEASure:PHASe"](#) on page 309

**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"

```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:MEASure:TEDGe**

**N** (see [page 676](#))

**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}

<digital channels> ::= DIGital0,...,DIGital15 for the MSO models

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

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

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)
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:TVALue"](#) on page 319
  - [":MEASure:VTIME"](#) on page 329

**:MEASure:TVALue**

**C** (see [page 676](#))

**Query Syntax** :MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross. The value can be volts or a math function value such as dB, Vs, or V/s.

<slope> ::= direction of the waveform. A rising slope is indicated by a plus sign (+). A falling edge is indicated by a minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence number is one, the first crossing is reported. If the number is two, the second crossing is reported, etc.

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

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

<value> ::= time in seconds of the specified value crossing in  
NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:TEDGE"](#) on page 317
  - [":MEASure:VTIME"](#) on page 329



**:MEASure:VAMPlitude**(see [page 676](#))**Command Syntax** :MEASure:VAMPlitude [<source>]

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

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

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

The :MEASure: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>

&lt;value&gt; ::= the amplitude of the selected waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VBASe"](#) on page 323
  - [":MEASure:VTOP"](#) on page 330
  - [":MEASure:VPP"](#) on page 326

**:MEASure:VAVerage**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VAVerage? [<source>]

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

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

<value> ::= calculated average value in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:VBASe**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VBASe? [<source>]

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

**Return Format** <base\_voltage><NL>

<base\_voltage> ::= value at the base of the selected waveform in  
NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VTOP"](#) on page 330
  - [":MEASure:VAMPLitude"](#) on page 321
  - [":MEASure:VMIN"](#) on page 325

**:MEASure:VMAX**(see [page 676](#))**Command Syntax** :MEASure:VMAX [<source>]

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

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

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

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VMAX? [<source>]

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

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

<value> ::= maximum vertical value of the selected waveform in  
NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VPP"](#) on page 326
  - [":MEASure:VTOP"](#) on page 330

**:MEASure:VMIN**(see [page 676](#))**Command Syntax** :MEASure:VMIN [<source>]

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

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

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

The :MEASure: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 293
  - [":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 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

**Query Syntax** :MEASure:VPP? [<source>]

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

$$V_{pp} = V_{max} - V_{min}$$

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

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

<value> ::= vertical peak to peak value in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VMIN"](#) on page 325
  - [":MEASure:VAMPLitude"](#) on page 321

**:MEASure:VRATio**

**N** (see [page 676](#))

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

The :MEASure:VRATio command places the instrument in the continuous measurement mode and starts a ratio measurement.

**Query Syntax** :MEASure:VRATio? [<source1>][,<source2>]

The :MEASure:VRATio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

**Return Format** <value><NL>  
 <value> ::= the ratio value in dB in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:VRMS"](#) on page 328
  - [":MEASure:SOURce"](#) on page 315

**:MEASure:VRMS**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VRMS command installs a screen measurement and starts a dc RMS value measurement. If the optional source parameter is specified, the current source is modified.

**NOTE**

This command is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VRMS? [<source>]

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

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

<value> ::= calculated dc RMS value in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315



**:MEASure:VTime**

**N** (see [page 676](#))

**Query Syntax** :MEASure:VTime? <vtime\_argument>[,<source>]  
 <vtime\_argument> ::= time from trigger in seconds  
 <source> ::= {<digital channels> | CHANnel<n> | FUNCTION | MATH}  
 <digital channels> ::= DIGital0,...,DIGital15 for the MSO models  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :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 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:TEDGe"](#) on page 317
  - [":MEASure:TVALue"](#) on page 319

**:MEASure:VTOP**

**C** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

**NOTE**

This query is not available if the source is FFT (Fast Fourier Transform).

**Query Syntax** :MEASure:VTOP? [<source>]

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

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

<value> ::= vertical value at the top of the waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:SOURce"](#) on page 315
  - [":MEASure:VMAX"](#) on page 324
  - [":MEASure:VAMplitude"](#) on page 321
  - [":MEASure:VBASe"](#) on page 323

**:MEASure:XMAX**

**N** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

**NOTE**

:MEASure:XMAX is an alias for :MEASure:TMAX.

**Query Syntax** :MEASure:XMAX? [<source>]

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

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

<value> ::= horizontal value of the maximum in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:XMIN"](#) on page 332
  - [":MEASure:TMAX"](#) on page 616

**:MEASure:XMIN**

**N** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

**NOTE**

:MEASure:XMIN is an alias for :MEASure:TMIN.

**Query Syntax** :MEASure:XMIN? [<source>]

The :MEASure:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

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

<value> ::= horizontal value of the minimum in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:XMAX"](#) on page 331
  - [":MEASure:TMIN"](#) on page 617

## :POD Commands

Control all oscilloscope functions associated with groups of digital channels. See ["Introduction to :POD<n> Commands"](#) on page 333.

**Table 61** :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 334</a> )	:POD<n>:DISPlay? (see <a href="#">page 334</a> )	{0   1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see <a href="#">page 335</a> )	:POD<n>:SIZE? (see <a href="#">page 335</a> )	<value> ::= {SMALl   MEDium   LARGe}
:POD<n>:THReshold <type>[suffix] (see <a href="#">page 336</a> )	:POD<n>:THReshold? (see <a href="#">page 336</a> )	<n> ::= 1-2 in NR1 format <type> ::= {CMOS   ECL   TTL   <user defined value>} <user defined value> ::= value in NR3 format [suffix] ::= {V   mV   uV }

### Introduction to :POD<n> Commands

<n> ::= {1 | 2}

The POD subsystem commands control the viewing and threshold of groups of digital channels.

POD1 ::= D0-D7

POD2 ::= D8-D15

### NOTE

These commands are only valid for the MSO models.

### Reporting the Setup

Use :POD1? or :POD2? to query setup information for the POD subsystem.

### Return Format

The following is a sample response from the :POD1? query. In this case, the query was issued following a \*RST command.

```
:POD1:DISP 0;THR +1.40E+00
```

**:POD<n>:DISPlay**

**N** (see [page 676](#))

**Command Syntax** :POD<n>:DISPlay <display>

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

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

POD1 ::= D0-D7

POD2 ::= D8-D15

The :POD<n>:DISPlay command turns displaying of the specified group of channels on or off.

**NOTE**

This command is only valid for the MSO models.

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

The :POD<n>:DISPlay? query returns the current display setting of the specified group of channels.

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

<display> ::= {0 | 1}

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 333
  - [":DIGital<n>:DISPlay"](#) on page 225
  - [":CHANnel<n>:DISPlay"](#) on page 209
  - [":VIEW"](#) on page 171
  - [":BLANK"](#) on page 143
  - [":STATus"](#) on page 168

**:POD<n>:SIZE**

**N** (see [page 676](#))

**Command Syntax** :POD<n>:SIZE <value>

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

POD1 ::= D0-D7

POD2 ::= D8-D15

<value> ::= {SMALL | MEDIUM | LARGE}

The :POD<n>:SIZE command specifies the size of digital channels on the display.

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :POD<n>:SIZE?

The :POD<n>:SIZE? query returns the size setting for the specified group of channels.

**Return Format** <size\_value><NL>

<size\_value> ::= {SMALL | MEDIUM | LARGE}

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 333
  - [":DIGital<n>:SIZE"](#) on page 228
  - [":DIGital<n>:POSition"](#) on page 227

**:POD<n>:THReshold**

**N** (see [page 676](#))

**Command Syntax** :POD<n>:THReshold <type>[<suffix>]

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

<type> ::= {CMOS | ECL | TTL | <user defined value>}

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

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

POD1 ::= D0-D7

POD2 ::= D8-D15

TTL ::= 1.4V

CMOS ::= 2.5V

ECL ::= -1.3V

The :POD<n>:THReshold command sets the threshold for the specified group of channels. The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold).

**NOTE**

This command is only valid for the MSO models.

**Query Syntax** :POD<n>:THReshold?

The :POD<n>:THReshold? query returns the threshold value for the specified group of channels.

**Return Format** <threshold><NL>

<threshold> ::= Floating point number in NR3 format

- See Also**
- ["Introduction to :POD<n> Commands"](#) on page 333
  - [":DIGital<n>:THReshold"](#) on page 229
  - [":TRIGger\[:EDGE\]:LEVel"](#) on page 439

**Example Code**

```
' THRESHOLD - This command is used to set the voltage threshold for
' the waveforms. There are three preset values (TTL, CMOS, and ECL)
' and you can also set a user-defined threshold value between
' -8.0 volts and +8.0 volts.
'
' In this example, we set channels 0-7 to CMOS, then set channels
' 8-15 to a user-defined 2.0 volts, and then set the external trigger
' to TTL. Of course, you only need to set the thresholds for the
' channels you will be using in your program.
```



```
' Set channels 0-7 to CMOS threshold.  
myScope.WriteString ":POD1:THRESHOLD CMOS"  
  
' Set channels 8-15 to 2.0 volts.  
myScope.WriteString ":POD2:THRESHOLD 2.0"  
  
' Set external channel to TTL threshold (short form).  
myScope.WriteString ":TRIG:LEV TTL,EXT"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

## :RECall Commands

Recall previously saved oscilloscope setups and traces. See "[Introduction to :RECall Commands](#)" on page 338.

**Table 62** :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILENAME <base_name> (see <a href="#">page 339</a> )	:RECall:FILENAME? (see <a href="#">page 339</a> )	<base_name> ::= quoted ASCII string
:RECall:IMAGE[:START] [<file_spec>] (see <a href="#">page 340</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
n/a	:RECall:PWD? (see <a href="#">page 341</a> )	<path_info> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see <a href="#">page 342</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string

### Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups and traces.

#### Reporting the Setup

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

#### Return Format

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

```
:REC:FIL "scope_0"
```

**:RECall:FILEname**

**N** (see [page 676](#))

**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 338
  - [":RECall:IMAGe\[:STArT\]"](#) on page 340
  - [":RECall:SEtUp\[:STArT\]"](#) on page 342
  - [":SAVE:FILEname"](#) on page 345

### :RECall:IMAGe[:STARt]

**N** (see [page 676](#))

**Command Syntax**    :RECall:IMAGe[:STARt] [<file\_spec>]  
                         <file\_spec> ::= {<internal\_loc> | <file\_name>}  
                         <internal\_loc> ::= 0-9; an integer in NR1 format  
                         <file\_name> ::= quoted ASCII string

The :RECall:IMAGe[:STARt] command recalls a trace (TIFF) image.

#### NOTE

If a file extension is provided as part of a specified <file\_name>, it must be ".tif".

- 
- See Also**
- ["Introduction to :RECall Commands"](#) on page 338
  - [":RECall:FILEname"](#) on page 339
  - [":SAVE:IMAGe\[:STARt\]"](#) on page 346

**:RECall:PWD**

**N** (see [page 676](#))

**Query Syntax** :RECall:PWD?

The :RECall:PWD? query returns the current recall path information.

**Return Format** <path\_info><NL>

<path\_info> ::= quoted ASCII string

- See Also**
- ["Introduction to :RECall Commands"](#) on page 338
  - [":SAVE:PWD"](#) on page 352

## :RECall:SETup[:START]

**N** (see [page 676](#))

**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 338
  - [":RECall:FILENAME"](#) on page 339
  - [":SAVE:SETup\[:START\]"](#) on page 353

## :SAVE Commands

Save oscilloscope setups and traces, screen images, and data. See ["Introduction to :SAVE Commands"](#) on page 344.

**Table 63** :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILENAME <base_name> (see <a href="#">page 345</a> )	:SAVE:FILENAME? (see <a href="#">page 345</a> )	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_spec>] (see <a href="#">page 346</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:IMAGE:AREA <area> (see <a href="#">page 347</a> )	:SAVE:IMAGE:AREA? (see <a href="#">page 347</a> )	<area> ::= {GRATICule   SCReen}
:SAVE:IMAGE:FACTors {0   OFF}   {1   ON} (see <a href="#">page 348</a> )	:SAVE:IMAGE:FACTors? (see <a href="#">page 348</a> )	{0   1}
:SAVE:IMAGE:FORMat <format> (see <a href="#">page 349</a> )	:SAVE:IMAGE:FORMat? (see <a href="#">page 349</a> )	<format> ::= {TIFF   {BMP   BMP24bit}   BMP8bit   PNG   NONE}
:SAVE:IMAGE:INKSaver {0   OFF}   {1   ON} (see <a href="#">page 350</a> )	:SAVE:IMAGE:INKSaver? (see <a href="#">page 350</a> )	{0   1}
:SAVE:IMAGE:PALette <palette> (see <a href="#">page 351</a> )	:SAVE:IMAGE:PALette? (see <a href="#">page 351</a> )	<palette> ::= {COLor   GRAYscale   MONochrome}
n/a	:SAVE:PWD? (see <a href="#">page 352</a> )	<path_info> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see <a href="#">page 353</a> )	n/a	<file_spec> ::= {<internal_loc>   <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVEform[:START] [<file_name>] (see <a href="#">page 354</a> )	n/a	<file_name> ::= quoted ASCII string

**Table 63** :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WAVEform:FORMat <format> (see page 355)	:SAVE:WAVEform:FORMat ? (see page 355)	<format> ::= {ALB   ASCiixy   CSV   BINary   NONE}
:SAVE:WAVEform:LENGth <length> (see page 356)	:SAVE:WAVEform:LENGth ? (see page 356)	<length> ::= 100 to max. length; an integer in NR1 format

**Introduction to :SAVE Commands** The :SAVE subsystem provides commands to save oscilloscope setups and traces, screen images, and data.

:SAV is an acceptable short form for :SAVE.

#### Reporting the Setup

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

#### Return Format

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

```
:SAVE:FIL "scope_0";:SAVE:IMAG:AREA GRAT;FACT 0;FORM TIFF;INKS 0;
PAL MON;:SAVE:WAV:FORM NONE
```



**:SAVE:FiLename**

**N** (see [page 676](#))

**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 344
  - [":SAVE:IMAGe\[:STARt\]"](#) on page 346
  - [":SAVE:SETup\[:STARt\]"](#) on page 353
  - [":SAVE:WAVEform\[:STARt\]"](#) on page 354
  - [":SAVE:PWD"](#) on page 352
  - [":RECall:FiLename"](#) on page 339

**:SAVE:IMAGe[:START]**

**N** (see [page 676](#))

**Command Syntax**    :SAVE:IMAGe[:START] [<file\_spec>]  
                          <file\_spec> ::= {<internal\_loc> | <file\_name>}  
                          <internal\_loc> ::= 0-9; an integer in NR1 format  
                          <file\_name> ::= quoted ASCII string

The :SAVE:IMAGe[:START] command saves an image.

**NOTE**

If a file extension is provided as part of a specified <file\_name>, it must match the extension expected by the format specified in :SAVE:IMAGe:FORMat.

**NOTE**

The <internal\_loc> option is only valid if :SAVE:IMAGe:FORMat is TIFF.

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:IMAGe:AREA"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351
  - [":SAVE:FILEname"](#) on page 345
  - [":RECall:IMAGe\[:START\]"](#) on page 340

**:SAVE:IMAGe:AREA**

**N** (see [page 676](#))

**Command Syntax** :SAVE:IMAGe:AREA <area>

<area> ::= {GRATicule | SCReen}

The :SAVE:IMAGe:AREA command sets the area that will be saved as part of the image. If the :SAVE:IMAGe:FORMat is TIFF, the area is GRATicule. Otherwise, it is SCReen.

**Query Syntax** :SAVE:IMAGe:AREA?

The :SAVE:IMAGe:AREA? query returns the selected image area.

**Return Format** <area><NL>

<area> ::= {GRAT | SCR}

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:IMAGe\[:START\]"](#) on page 346
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351

**:SAVE:IMAGe:FACTors**

**N** (see [page 676](#))

**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 344
  - [":SAVE:IMAGe\[:STArT\]"](#) on page 346
  - [":SAVE:IMAGe:AREa"](#) on page 347
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351

**:SAVE:IMAGe:FORMat**

**N** (see [page 676](#))

**Command Syntax** :SAVE:IMAGe:FORMat <format>  
 <format> ::= {TIFF | {BMP | BMP24bit} | BMP8bit | PNG}

The :SAVE:IMAGe:FORMat command sets the image format type.

**Query Syntax** :SAVE:IMAGe:FORMat?

The :SAVE:IMAGe:FORMat? query returns the selected image format type.

**Return Format** <format><NL>  
 <format> ::= {TIFF | BMP | BMP8 | PNG | NONE}

When NONE is returned, it indicates that a waveform data file format is currently selected.

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:IMAGe\[:STArT\]"](#) on page 346
  - [":SAVE:IMAGe:AREa"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351
  - [":SAVE:WAVEform:FORMat"](#) on page 355

### :SAVE:IMAGe:INKSaver

**N** (see [page 676](#))

**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 344
  - [":SAVE:IMAGe\[:START\]"](#) on page 346
  - [":SAVE:IMAGe:AREA"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:PALette"](#) on page 351

**:SAVE:IMAGe:PALette**

**N** (see [page 676](#))

**Command Syntax** :SAVE:IMAGe:PALette <palette>  
 <palette> ::= {COLor | GRAYscale | MONochrome}

The :SAVE:IMAGe:PALette command sets the image palette color.

**NOTE**

MONochrome is the only valid choice when the :SAVE:IMAGe:FORMat is TIFF. COLor and GRAYscale are the only valid choices when the format is not TIFF.

**Query Syntax** :SAVE:IMAGe:PALette?

The :SAVE:IMAGe:PALette? query returns the selected image palette color.

**Return Format** <palette><NL>  
 <palette> ::= {COL | GRAY | MON}

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:IMAGe\[:STArT\]"](#) on page 346
  - [":SAVE:IMAGe:AREa"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350

## :SAVE:PWD

**N** (see [page 676](#))

**Query Syntax** :SAVE:PWD?

The :SAVE:PWD? query returns the current save path information.

**Return Format** <path\_info><NL>

<path\_info> ::= quoted ASCII string

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:FILENAME"](#) on page 345
  - [":RECALL:PWD"](#) on page 341



**:SAVE:SETup[:START]**

**N** (see [page 676](#))

**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 344
  - [":SAVE:FILENAME"](#) on page 345
  - [":RECall:SETup\[:START\]"](#) on page 342

**:SAVE:WAVeform[:START]****N** (see [page 676](#))**Command Syntax**    :SAVE:WAVeform[:START] [[<file\\_name>](#)][<file\\_name>](#) ::= quoted ASCII string

The :SAVE:WAVeform[:START] command saves oscilloscope waveform data to a file.

**NOTE**

If a file extension is provided as part of a specified [<file\\_name>](#), it must match the extension expected by the format specified in :SAVE:WAVeform:FORMat.

- See Also**
- ["Introduction to :SAVE Commands"](#) on page 344
  - [":SAVE:WAVeform:FORMat"](#) on page 355
  - [":SAVE:WAVeform:LENGth"](#) on page 356
  - [":SAVE:FILEname"](#) on page 345
  - [":RECall:SETup\[:START\]"](#) on page 342

**:SAVE:WAVEform:FORMat**

**N** (see [page 676](#))

**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 344
  - [":SAVE:WAVEform\[:START\]"](#) on page 354
  - [":SAVE:WAVEform:LENGth"](#) on page 356
  - [":SAVE:IMAGe:FORMat"](#) on page 349

## **:SAVE:WAVEform:LENGth**

**N** (see [page 676](#))

**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 344
  - [":SAVE:WAVEform\[:START\]"](#) on page 354
  - [":WAVEform:POINTS"](#) on page 536
  - [":SAVE:WAVEform:FORMat"](#) on page 355

## :SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See ["Introduction to :SBUS Commands"](#) on page 358.

**Table 64** :SBUS Commands Summary

Command	Query	Options and Query Returns
:SBUS:BUSDoctor:ADDRe ss <value> (see <a href="#">page 360</a> )	:SBUS:BUSDoctor:ADDRe ss? (see <a href="#">page 360</a> )	<value> ::= <field value>, <field value>, <field value>, <field value> <field value> ::= integer from 0-255 in NR1 format
:SBUS:BUSDoctor:BAUDr ate <baudrate> (see <a href="#">page 361</a> )	:SBUS:BUSDoctor:BAUDr ate? (see <a href="#">page 361</a> )	<baudrate> ::= {2500000   5000000   10000000}
:SBUS:BUSDoctor:CHANn el <channel> (see <a href="#">page 362</a> )	:SBUS:BUSDoctor:CHANn el? (see <a href="#">page 362</a> )	<channel> ::= {A   B}
:SBUS:BUSDoctor:MODE <mode> (see <a href="#">page 363</a> )	:SBUS:BUSDoctor:MODE? (see <a href="#">page 363</a> )	<mode> ::= {ASYNchronous   SYNchronous   PC}
n/a	:SBUS:CAN:COUNT:ERRor ? (see <a href="#">page 364</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:OVERl oad? (see <a href="#">page 365</a> )	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see <a href="#">page 366</a> )	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal ? (see <a href="#">page 367</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILi zation? (see <a href="#">page 368</a> )	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0   OFF}   {1   ON}} (see <a href="#">page 369</a> )	:SBUS:DISPlay? (see <a href="#">page 369</a> )	{0   1}
n/a	:SBUS:FLEXray:COUNT:N ULL? (see <a href="#">page 370</a> )	<frame_count> ::= integer in NR1 format
:SBUS:FLEXray:COUNT:R ESet (see <a href="#">page 371</a> )	n/a	n/a
n/a	:SBUS:FLEXray:COUNT:S YNC? (see <a href="#">page 372</a> )	<frame_count> ::= integer in NR1 format

**Table 64** :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:FLEXray:COUNT:TOTal? (see <a href="#">page 373</a> )	<frame_count> ::= integer in NR1 format
:SBUS:IIC:ASize <size> (see <a href="#">page 374</a> )	:SBUS:IIC:ASize? (see <a href="#">page 374</a> )	<size> ::= {BIT7   BIT8}
:SBUS:LIN:PARity {{0   OFF}   {1   ON}} (see <a href="#">page 375</a> )	:SBUS:LIN:PARity? (see <a href="#">page 375</a> )	{0   1}
:SBUS:MODE <mode> (see <a href="#">page 376</a> )	:SBUS:MODE? (see <a href="#">page 376</a> )	<mode> ::= {IIC   SPI   CAN   LIN   FLEXray   UART}
:SBUS:SPI:WIDTh <word_width> (see <a href="#">page 377</a> )	:SBUS:SPI:WIDTh? (see <a href="#">page 377</a> )	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see <a href="#">page 378</a> )	:SBUS:UART:BASE? (see <a href="#">page 378</a> )	<base> ::= {ASCIi   BINary   HEX}
n/a	:SBUS:UART:COUNT:ERROr? (see <a href="#">page 379</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESe t (see <a href="#">page 380</a> )	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFR ames? (see <a href="#">page 381</a> )	<frame_count> ::= integer in NR1 format
n/a	:SBUS:UART:COUNT:TXFR ames? (see <a href="#">page 382</a> )	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see <a href="#">page 383</a> )	:SBUS:UART:FRAMing? (see <a href="#">page 383</a> )	<value> ::= {OFF   <decimal>   <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary

### Introduction to :SBUS Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

#### NOTE

These commands are only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

Reporting the Setup

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

#### Return Format

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

```
:SBUS:DISP 0;MODE IIC
```

**:SBUS:BUSDoctor:ADDRess**

**N** (see [page 676](#))

**Command Syntax** :SBUS:BUSDoctor:ADDRess <value>  
 <value> ::= <field value>, <field value>, <field value>, <field value>  
 <field value> ::= integer from 0-255 in NR1 format

The :SBUS:BUSDoctor:ADDRess command sets the four byte values that make up the BusDoctor's IP address.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :SBUS:BUSDoctor:ADDRess?

The :SBUS:BUSDoctor:ADDRess? query returns the current BusDoctor IP address byte values.

**Return Format** <value><NL>  
 <value> ::= <field value>, <field value>, <field value>, <field value>  
 <field value> ::= integer from 0-255 in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:FLEXray Commands"](#) on [page 443](#)



**:SBUS:BUSDoctor:BAUDrate**

**N** (see [page 676](#))

**Command Syntax** :SBUS:BUSDoctor:BAUDrate <baudrate>  
 <baudrate> ::= {2500000 | 5000000 | 10000000}

The :SBUS:BUSDoctor:BAUDrate command sets the baud rate for the BusDoctor to 2.5 Mb/s, 5 Mb/s, or 10 Mb/s.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :SBUS:BUSDoctor:BAUDrate?

The :SBUS:BUSDoctor:BAUDrate? query returns the current BusDoctor baud rate setting.

**Return Format** <baudrate><NL>  
 <baudrate> ::= {2500000 | 5000000 | 10000000}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:FLEXray Commands"](#) on [page 443](#)

**:SBUS:BUSDoctor:CHANnel**

**N** (see [page 676](#))

**Command Syntax** :SBUS:BUSDoctor:CHANnel <channel>

<channel> ::= {A | B}

The :SBUS:BUSDoctor:BAUDrate command sets the channel that the BusDoctor analyzes/preprocesses.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :SBUS:BUSDoctor:CHANnel?

The :SBUS:BUSDoctor:CHANnel? query returns the current BusDoctor channel setting.

**Return Format** <channel><NL>

<channel> ::= {A | B}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:FLEXray Commands"](#) on [page 443](#)

**:SBUS:BUSDoctor:MODE**

**N** (see [page 676](#))

**Command Syntax** :SBUS:BUSDoctor:MODE <mode>  
 <mode> ::= {ASYNchronous | SYNChronous | PC}

The :SBUS:BUSDoctor:MODE command sets the operating mode of the BusDoctor:

- ASYNchronous – Oscilloscope controls BusDoctor, asynchronous mode monitoring (LAN connection required).
- SYNChronous – Oscilloscope controls BusDoctor, synchronous mode monitoring (LAN connection required).
- PC – PC running Decomsys VISION software controls BusDoctor.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the FlexRay triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :SBUS:BUSDoctor:MODE?

The :SBUS:BUSDoctor:MODE? query returns the current BusDoctor operating mode setting.

**Return Format** <mode><NL>  
 <mode> ::= {ASYN | SYNC | PC}

**Errors** • "-241, Hardware missing" on page 637

**See Also** • "Introduction to :SBUS Commands" on page 358  
 • ":TRIGger:FLEXray Commands" on page 443

## **:SBUS:CAN:COUNT:ERRor**

**N** (see [page 676](#))

**Query Syntax** :SBUS:CAN:COUNT:ERRor?

Returns the error frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:CAN:COUNT:RESet"](#) on [page 366](#)  
• ["Introduction to :SBUS Commands"](#) on [page 358](#)  
• [":SBUS:MODE"](#) on [page 376](#)  
• [":TRIGger:CAN Commands"](#) on [page 416](#)

**:SBUS:CAN:COUNT:OVERload****N** (see [page 676](#))**Query Syntax** :SBUS:CAN:COUNT:OVERload?

Returns the overload frame count.

**Return Format** <frame\_count><NL>

&lt;frame\_count&gt; ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)**See Also** • ":SBUS:CAN:COUNT:RESet" on [page 366](#)  
• "Introduction to :SBUS Commands" on [page 358](#)  
• ":SBUS:MODE" on [page 376](#)  
• ":TRIGger:CAN Commands" on [page 416](#)

## **:SBUS:CAN:COUNt:RESet**

**N** (see [page 676](#))

**Command Syntax** :SBUS:CAN:COUNt:RESet

Resets the frame counters.

**Errors** • ["-241, Hardware missing"](#) on page 637

- See Also**
- [":SBUS:CAN:COUNt:ERRor"](#) on page 364
  - [":SBUS:CAN:COUNt:OVERload"](#) on page 365
  - [":SBUS:CAN:COUNt:TOTal"](#) on page 367
  - [":SBUS:CAN:COUNt:UTILization"](#) on page 368
  - ["Introduction to :SBUS Commands"](#) on page 358
  - [":SBUS:MODE"](#) on page 376
  - [":TRIGger:CAN Commands"](#) on page 416

**:SBUS:CAN:COUNT:TOTal**

**N** (see [page 676](#))

**Query Syntax** :SBUS:CAN:COUNT:TOTal?

Returns the total frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ":SBUS:CAN:COUNT:RESet" on [page 366](#)  
 • "Introduction to :SBUS Commands" on [page 358](#)  
 • ":SBUS:MODE" on [page 376](#)  
 • ":TRIGger:CAN Commands" on [page 416](#)

## **:SBUS:CAN:COUNT:UTILization**

**N** (see [page 676](#))

**Query Syntax** :SBUS:CAN:COUNT:UTILization?

Returns the percent utilization.

**Return Format** <percent><NL>

<percent> ::= floating-point in NR3 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:CAN:COUNT:RESet"](#) on [page 366](#)  
• ["Introduction to :SBUS Commands"](#) on [page 358](#)  
• [":SBUS:MODE"](#) on [page 376](#)  
• [":TRIGger:CAN Commands"](#) on [page 416](#)



**:SBUS:DISPlay**

**N** (see [page 676](#))

**Command Syntax** :SBUS:DISPlay <display>  
 <display> ::= {{1 | ON} | {0 | OFF}}

The :SBUS:DISPlay command turns displaying of the serial decode bus on or off.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax** :SBUS:DISPlay?

The :SBUS:DISPlay? query returns the current display setting of the serial decode bus.

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

**Errors** • ["-241, Hardware missing"](#) on page 637

**See Also** • ["Introduction to :SBUS Commands"](#) on page 358  
 • [":CHANnel<n>:DISPlay"](#) on page 209  
 • [":DIGital<n>:DISPlay"](#) on page 225  
 • [":POD<n>:DISPlay"](#) on page 334  
 • [":VIEW"](#) on page 171  
 • [":BLANK"](#) on page 143  
 • [":STATus"](#) on page 168

## **:SBUS:FLEXray:COUNT:NULL**

**N** (see [page 676](#))

**Query Syntax** :SBUS:FLEXray:COUNT:NULL?

Returns the FlexRay null frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:FLEXray:COUNT:RESet"](#) on [page 371](#)  
• ["Introduction to :SBUS Commands"](#) on [page 358](#)  
• [":SBUS:MODE"](#) on [page 376](#)  
• [":TRIGger:FLEXray Commands"](#) on [page 443](#)

**:SBUS:FLEXray:COUNT:RESet****N** (see [page 676](#))**Command Syntax** :SBUS:FLEXray:COUNT:RESet

Resets the FlexRay frame counters.

**Errors** • "-241, Hardware missing" on [page 637](#)**See Also** • [":SBUS:FLEXray:COUNT:NULL"](#) on [page 370](#)  
• [":SBUS:FLEXray:COUNT:SYNC"](#) on [page 372](#)  
• [":SBUS:FLEXray:COUNT:TOTaI"](#) on [page 373](#)  
• ["Introduction to :SBUS Commands"](#) on [page 358](#)  
• [":SBUS:MODE"](#) on [page 376](#)  
• [":TRIGger:FLEXray Commands"](#) on [page 443](#)

## **:SBUS:FLEXray:COUNT:SYNC**

**N** (see [page 676](#))

**Query Syntax** :SBUS:FLEXray:COUNT:SYNC?

Returns the FlexRay sync frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:FLEXray:COUNT:RESet"](#) on [page 371](#)  
• ["Introduction to :SBUS Commands"](#) on [page 358](#)  
• [":SBUS:MODE"](#) on [page 376](#)  
• [":TRIGger:FLEXray Commands"](#) on [page 443](#)

**:SBUS:FLEXray:COUNT:TOTal**

**N** (see [page 676](#))

**Query Syntax** :SBUS:FLEXray:COUNT:TOTal?

Returns the FlexRay total frame count.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ":SBUS:FLEXray:COUNT:RESet" on [page 371](#)  
 • "Introduction to :SBUS Commands" on [page 358](#)  
 • ":SBUS:MODE" on [page 376](#)  
 • ":TRIGger:FLEXray Commands" on [page 443](#)

**:SBUS:IIC:ASIZe**

**N** (see [page 676](#))

**Command Syntax** :SBUS:IIC:ASIZe <size>  
 <size> ::= {BIT7 | BIT8}

The :SBUS:IIC:ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

**Query Syntax** :SBUS:IIC:ASIZe?

The :SBUS:IIC:ASIZe? query returns the current IIC address width setting.

**Return Format** <mode><NL>  
 <mode> ::= {BIT7 | BIT8}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:IIC Commands"](#) on [page 464](#)

**:SBUS:LIN:PARity**

**N** (see [page 676](#))

**Command Syntax** :SBUS:LIN:PARity <display>  
 <display> ::= {{1 | ON} | {0 | OFF}}

The :SBUS:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :SBUS:LIN:PARity?

The :SBUS:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

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

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:LIN Commands"](#) on [page 473](#)

**:SBUS:MODE**

**N** (see [page 676](#))

**Command Syntax** :SBUS:MODE <mode>

<mode> ::= {IIC | SPI | CAN | LIN | FLEXray | UART}

The :SBUS:MODE command determines the decode mode for the serial bus.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

**Query Syntax** :SBUS:MODE?

The :SBUS:MODE? query returns the current serial bus decode mode setting.

**Return Format** <mode><NL>

<mode> ::= {IIC | SPI | CAN | LIN | FLEX | UART | NONE}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)

- [":TRIGger:MODE"](#) on [page 411](#)
- [":TRIGger:IIC Commands"](#) on [page 464](#)
- [":TRIGger:SPI Commands"](#) on [page 489](#)
- [":TRIGger:CAN Commands"](#) on [page 416](#)
- [":TRIGger:LIN Commands"](#) on [page 473](#)
- [":TRIGger:FLEXray Commands"](#) on [page 443](#)
- [":TRIGger:UART Commands"](#) on [page 504](#)



**:SBUS:SPI:WIDTH**

**N** (see [page 676](#))

**Command Syntax** :SBUS:SPI:WIDTH <word\_width>  
 <word\_width> ::= integer 4-16 in NR1 format

The :SBUS:SPI:WIDTH command determines the number of bits in a word of data for SPI.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

**Query Syntax** :SBUS:SPI:WIDTH?

The :SBUS:SPI:WIDTH? query returns the current SPI decode word width.

**Return Format** <word\_width><NL>  
 <word\_width> ::= integer 4-16 in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":SBUS:MODE"](#) on [page 376](#)  
 • [":TRIGger:SPI Commands"](#) on [page 489](#)

**:SBUS:UART:BASE**

**N** (see [page 676](#))

**Command Syntax** :SBUS:UART:BASE <base>  
 <base> ::= {ASCIi | BINary | HEX}

The :SBUS:UART:BASE command determines the base to use for the UART decode display.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Query Syntax** :SBUS:UART:BASE?

The :SBUS:UART:BASE? query returns the current UART decode base setting.

**Return Format** <base><NL>  
 <base> ::= {ASCIi | BINary | HEX}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":TRIGger:UART Commands"](#) on [page 504](#)

**:SBUS:UART:COUNt:ERRor**

**N** (see [page 676](#))

**Query Syntax** :SBUS:UART:COUNt:ERRor?

Returns the UART error frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:UART:COUNt:RESet"](#) on [page 380](#)  
 • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":SBUS:MODE"](#) on [page 376](#)  
 • [":TRIGger:UART Commands"](#) on [page 504](#)

## :SBUS:UART:COUNt:RESet

**N** (see [page 676](#))

**Command Syntax** :SBUS:UART:COUNt:RESet

Resets the UART frame counters.

### NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ":SBUS:UART:COUNt:ERRor" on [page 379](#)  
• ":SBUS:UART:COUNt:RXFRames" on [page 381](#)  
• ":SBUS:UART:COUNt:TXFRames" on [page 382](#)  
• "Introduction to :SBUS Commands" on [page 358](#)  
• ":SBUS:MODE" on [page 376](#)  
• ":TRIGger:UART Commands" on [page 504](#)

**:SBUS:UART:COUNt:RXFRames**

**N** (see [page 676](#))

**Query Syntax** :SBUS:UART:COUNt:RXFRames?

Returns the UART Rx frame count.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • [":SBUS:UART:COUNt:RESet"](#) on [page 380](#)  
 • ["Introduction to :SBUS Commands"](#) on [page 358](#)  
 • [":SBUS:MODE"](#) on [page 376](#)  
 • [":TRIGger:UART Commands"](#) on [page 504](#)

## :SBUS:UART:COUNt:TXFRames

**N** (see [page 676](#))

**Query Syntax** :SBUS:UART:COUNt:TXFRames?

Returns the UART Tx frame count.

### NOTE

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Return Format** <frame\_count><NL>

<frame\_count> ::= integer in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

- See Also**
- ":SBUS:UART:COUNt:RESet" on [page 380](#)
  - "Introduction to :SBUS Commands" on [page 358](#)
  - ":SBUS:MODE" on [page 376](#)
  - ":TRIGger:UART Commands" on [page 504](#)

**:SBUS:UART:FRAMing**

**N** (see [page 676](#))

**Command Syntax** :SBUS:UART:FRAMing <value>

<value> ::= {OFF | <decimal> | <nondecimal>}

<decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)

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

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

The :SBUS:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

**NOTE**

This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

**Query Syntax** :SBUS:UART:FRAMing?

The :SBUS:UART:FRAMing? query returns the current UART decode base setting.

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

<value> ::= {OFF | <decimal>}

<decimal> ::= 8-bit integer in decimal from 0-255

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :SBUS Commands"](#) on [page 358](#)

• [":TRIGger:UART Commands"](#) on [page 504](#)

## :SYSTem Commands

Control basic system functions of the oscilloscope. See ["Introduction to :SYSTem Commands"](#) on page 384.

**Table 65** :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:DATE <date> (see <a href="#">page 385</a> )	:SYSTem:DATE? (see <a href="#">page 385</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 386</a> )	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see <a href="#">page 387</a> )	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see <a href="#">page 635</a> ).
:SYSTem:LOCK <value> (see <a href="#">page 388</a> )	:SYSTem:LOCK? (see <a href="#">page 388</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:PROTection:LOCK <value> (see <a href="#">page 389</a> )	:SYSTem:PROTection:LOCK? (see <a href="#">page 389</a> )	<value> ::= {{1   ON}   {0   OFF}}
:SYSTem:SETup <setup_data> (see <a href="#">page 390</a> )	:SYSTem:SETup? (see <a href="#">page 390</a> )	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see <a href="#">page 392</a> )	:SYSTem:TIME? (see <a href="#">page 392</a> )	<time> ::= hours,minutes,seconds in NR1 format

**Introduction to :SYSTem Commands** SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.



**:SYSTem:DATE**

**N** (see [page 676](#))

**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 384
  - [":SYSTem:TIME"](#) on page 392

## :SYSTem:DSP

**N** (see [page 676](#))

**Command Syntax**    :SYSTem:DSP <string>  
                         <string> ::= quoted ASCII string (up to 254 characters)

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYSTem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

**See Also**    • ["Introduction to :SYSTem Commands"](#) on page 384

**:SYSTem:ERRor**

**C** (see [page 676](#))

**Query Syntax** :SYSTem:ERRor?

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

**Return Format** <error number>,<error string><NL>

<error number> ::= an integer error code in NR1 format

<error string> ::= quoted ASCII string containing the error message

Error messages are listed in "[Error Messages](#)" on page 635.

- See Also**
- "[Introduction to :SYSTem Commands](#)" on page 384
  - "[\\*ESR \(Standard Event Status Register\)](#)" on page 116
  - "[\\*CLS \(Clear Status\)](#)" on page 113

**:SYSTem:LOCK**

**N** (see [page 676](#))

**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 384

**:SYSTem:PROTection:LOCK**

**N** (see [page 676](#))

**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 384

**:SYSTem:SETup**

**C** (see [page 676](#))

**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 data in IEEE 488.2 # format

- See Also**
- ["Introduction to :SYSTem Commands"](#) on page 384
  - ["\\*LRN \(Learn Device Setup\)"](#) on page 119

**Example Code**

```
' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
' message that contains the current state of the instrument. Its
' format is a definite-length binary block, for example,
' #800002204<setup string><NL>
' where the setup string is 2204 bytes in length.
myScope.WriteString ":SYSTEM:SETUP?"
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors ' After reading query results.

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"

' Open file for output.
Close #1 ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult ' Write data.
Close #1 ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and
' write it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"

' Open file for input.
Open strPath For Binary Access Read As #1
Get #1, , varSetupString ' Read data.
Close #1 ' Close file.
```

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"  
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:SYSTem:TIME**

**N** (see [page 676](#))

**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 384
  - [":SYSTem:DATE"](#) on page 385



## :TIMebase Commands

Control all horizontal sweep functions. See ["Introduction to :TIMebase Commands"](#) on page 394.

**Table 66** :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see <a href="#">page 395</a> )	:TIMebase:MODE? (see <a href="#">page 395</a> )	<value> ::= {MAIN   WINDow   XY   ROLL}
:TIMebase:POSition <pos> (see <a href="#">page 396</a> )	:TIMebase:POSition? (see <a href="#">page 396</a> )	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see <a href="#">page 397</a> )	:TIMebase:RANGe? (see <a href="#">page 397</a> )	<range_value> ::= 5 ns through 500 s in NR3 format
:TIMebase:REFClock {{0   OFF}   {1   ON}} (see <a href="#">page 398</a> )	:TIMebase:REFClock? (see <a href="#">page 398</a> )	{0   1}
:TIMebase:REFerence {LEFT   CENTer   RIGHT} (see <a href="#">page 399</a> )	:TIMebase:REFerence? (see <a href="#">page 399</a> )	<return_value> ::= {LEFT   CENTer   RIGHT}
:TIMebase:SCALe <scale_value> (see <a href="#">page 400</a> )	:TIMebase:SCALe? (see <a href="#">page 400</a> )	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0   OFF}   {1   ON}} (see <a href="#">page 401</a> )	:TIMebase:VERNier? (see <a href="#">page 401</a> )	{0   1}
:TIMebase:WINDow:POSi tion <pos> (see <a href="#">page 402</a> )	:TIMebase:WINDow:POSi tion? (see <a href="#">page 402</a> )	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANG e <range_value> (see <a href="#">page 403</a> )	:TIMebase:WINDow:RANG e? (see <a href="#">page 403</a> )	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCAL e <scale_value> (see <a href="#">page 404</a> )	:TIMebase:WINDow:SCAL e? (see <a href="#">page 404</a> )	<scale_value> ::= scale value in seconds in NR3 format for the delayed 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 (delayed) 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 676](#))

**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 (delayed) time base mode, measurements are made in the delayed time base if possible; otherwise, the measurements are made in the main time base.
- **XY** – In the XY mode, the :TIMEbase:RANGe, :TIMEbase:POSition, and :TIMEbase:REFeRence commands are not available. No measurements are available in this mode.
- **ROLL** – In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMEbase:REFeRence selection changes to RIGHT.

**NOTE**

If a :DIGitize command is executed when the :TIMEbase:MODE is not MAIN, the :TIMEbase:MODE is set to MAIN.

**Query Syntax** :TIMEbase:MODE?

The :TIMEbase:MODE query returns the current time base mode.

**Return Format** <value><NL>  
 <value> ::= {MAIN | WIND | XY | ROLL}

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 394
  - ["\\*RST \(Reset\)"](#) on page 123
  - [":TIMEbase:RANGe"](#) on page 397
  - [":TIMEbase:POSition"](#) on page 396
  - [":TIMEbase:REFeRence"](#) on page 399

**Example Code**

```
' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
myScope.WriteString ":TIMEBASE:MODE MAIN"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:TIMebase:POSition**

**C** (see [page 676](#))

**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 394
  - [":TIMebase:REFeRence"](#) on page 399
  - [":TIMebase:RANGe"](#) on page 397
  - [":TIMebase:SCALe"](#) on page 400
  - [":TIMebase:WINDow:POSition"](#) on page 402
  - [":TIMebase:DELaY"](#) on page 628

**:TIMEbase:RANGe**(see [page 676](#))**Command Syntax** :TIMEbase:RANGe <range\_value>

&lt;range\_value&gt; ::= 5 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>

&lt;range\_value&gt; ::= 5 ns through 500 s in NR3 format

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 394
  - [":TIMEbase:MODE"](#) on page 395
  - [":TIMEbase:SCALE"](#) on page 400
  - [":TIMEbase:WINDow:RANGe"](#) on page 403

**Example Code**

```
' TIME_RANGE - Sets the full scale horizontal time in seconds. The
' range value is 10 times the time per division.
myScope.WriteString ":TIM:RANG 2e-3" ' Set the time range to 0.002
seconds.
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:TIMEbase:REFClock**

**N** (see [page 676](#))

**Command Syntax**    :TIMEbase:REFClock <value>  
                          <value> ::= {{1 | ON} | {0 | OFF}}

The :TIMEbase:REFClock command enables or disables the 10 MHz REF BNC located on the rear panel of the oscilloscope.

The 10 MHz REF BNC can be used as an input for the oscilloscope's reference clock (instead of the internal 10 MHz reference), or it can be used to output the internal 10 MHz reference clock when synchronizing multiple instruments (see ":ACQuire:RSIGnal" on page 180).

The :TIMEbase:REFClock ON command enables the 10 MHz REF BNC and sets the reference signal mode to IN. The :TIMEbase:REFClock OFF command disables the 10 MHz REF BNC (the same as setting the reference signal mode to OFF).

**Query Syntax**    :TIMEbase:REFClock?

The :TIMEbase:REFClock? query returns the current state of the 10 MHz reference signal mode. A "1" indicates that the 10 MHz REF input is enabled (on), and a "0" indicates that either the 10 MHz REF BNC is disabled (off) or that it is set as an output (by the :ACQuire:RSIGnal command).

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

**See Also**    • ":ACQuire:RSIGnal" on page 180

**:TIMEbase:REference**

**C** (see [page 676](#))

**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 394
  - [":TIMEbase:MODE"](#) on page 395

**Example Code**

```
' TIME_REFERENCE - Possible values are LEFT and CENTER.
' - LEFT sets the display reference on time division from the left.
' - CENTER sets the display reference to the center of the screen.
myScope.WriteString ":TIMEBASE:REFERENCE CENTER" ' Set reference to
center.
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

### :TIMebase:SCALe

**N** (see [page 676](#))

**Command Syntax** :TIMebase:SCALe <scale\_value>

<scale\_value> ::= 500 ps 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> ::= 500 ps through 50 s in NR3 format

- See Also**
- ["Introduction to :TIMebase Commands"](#) on page 394
  - [":TIMebase:RANGe"](#) on page 397
  - [":TIMebase:WINDow:SCALe"](#) on page 404
  - [":TIMebase:WINDow:RANGe"](#) on page 403



**:TIMEbase:VERNier**

**N** (see [page 676](#))

**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 394

**:TImebase:WINDow:POSition**(see [page 676](#))**Command Syntax** :TImebase:WINDow:POSition <pos value><pos value> ::= time from the trigger event to the delayed view  
reference point in NR3 format

The :TImebase:WINDow:POSition command sets the horizontal position in the 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 delayed 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 delayed view.

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

&lt;value&gt; ::= position value in seconds

- See Also**
- ["Introduction to :TImebase Commands"](#) on page 394
  - [":TImebase:MODE"](#) on page 395
  - [":TImebase:POSition"](#) on page 396
  - [":TImebase:RANGe"](#) on page 397
  - [":TImebase:SCALe"](#) on page 400
  - [":TImebase:WINDow:RANGe"](#) on page 403
  - [":TImebase:WINDow:SCALe"](#) on page 404

**:TIMebase:WINDow:RANGe**(see [page 676](#))**Command Syntax** :TIMebase:WINDow:RANGe <range value>

&lt;range value&gt; ::= range value in seconds in NR3 format

The :TIMebase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the delayed window. The range is 10 times the current delayed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMebase:RANGe value.

**Query Syntax** :TIMebase:WINDow:RANGe?

The :TIMebase:WINDow:RANGe? query returns the current window timebase range setting.

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

&lt;value&gt; ::= range value in seconds

- See Also**
- ["Introduction to :TIMebase Commands"](#) on page 394
  - [":TIMebase:RANGe"](#) on page 397
  - [":TIMebase:POSition"](#) on page 396
  - [":TIMebase:SCALe"](#) on page 400

**:TIMEbase:WINDow:SCALe**

**N** (see [page 676](#))

**Command Syntax** :TIMEbase:WINDow:SCALe <scale\_value>

<scale\_value> ::= scale value in seconds in NR3 format

The :TIMEbase:WINDow:SCALe command sets the 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 delayed window scale setting.

**Return Format** <scale\_value><NL>

<scale\_value> ::= current seconds per division for the delayed window

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 394
  - [":TIMEbase:RANGe"](#) on page 397
  - [":TIMEbase:POSition"](#) on page 396
  - [":TIMEbase:SCALe"](#) on page 400
  - [":TIMEbase:WINDow:RANGe"](#) on page 403

## :TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "Introduction to :TRIGger Commands" on page 405
- "General :TRIGger Commands" on page 408
- ":TRIGger:CAN Commands" on page 416
- ":TRIGger:DURation Commands" on page 427
- ":TRIGger:EBURst Commands" on page 433
- ":TRIGger[:EDGE] Commands" on page 437
- ":TRIGger:FLEXray Commands" on page 443
- ":TRIGger:GLITCh Commands" on page 455 (Pulse Width trigger)
- ":TRIGger:IIC Commands" on page 464
- ":TRIGger:LIN Commands" on page 473
- ":TRIGger:SEquence Commands" on page 481
- ":TRIGger:SPI Commands" on page 489
- ":TRIGger:TV Commands" on page 498
- ":TRIGger:USB Commands" on page 518
- ":TRIGger:UART Commands" on page 504

### 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 415) 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 411).

- **CAN (Controller Area Network) triggering** will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on CAN data and identifier patterns, set the bit sample point, and have the module send an acknowledge to the bus when it receives a valid message.

**NOTE**

The CAN and LIN serial decode option (Option ASM) replaces the functionality that was available with the N2758A CAN trigger module for the 54620/54640 Series oscilloscopes.

- **Edge triggering** identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Nth Edge Burst triggering** lets you trigger on the Nth edge of a burst that occurs after an idle time.
- **Pulse width triggering** (:TRIGger:GLITCh commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering** identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels.
- **Duration triggering** lets you define a pattern, then trigger on a specified time duration.
- **FlexRay triggering** will, when used with a BusDoctor 2 protocol analyzer and a four-channel mixed-signal oscilloscope with Option FRS, trigger on FlexRay bus frames, times, or errors.
- **IIC (Inter-IC bus) triggering** consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering** will trigger on LIN sync break at the beginning of a message frame. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on Frame IDs.

- **Sequence triggering** allows you to trigger the oscilloscope after finding a sequence of events. Defining a sequence trigger requires three steps:
  - a Define the event to find before you trigger on the next event. This event can be a pattern, and edge from a single channel, or the combination of a pattern and a channel edge.
  - b Define the trigger event. This event can be a pattern, and edge from a single channel, the combination of a pattern and a channel edge, or the nth occurrence of an edge from a single channel.
  - c Set an optional reset event. This event can be a pattern, an edge from a single channel, the combination of a pattern and a channel edge, or a timeout value.
- **SPI (Serial Peripheral Interface) triggering** consists of connecting the oscilloscope to a clock, data, and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 32 bits long.
- **TV triggering** is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than  $\frac{1}{2}$  division of sync amplitude with any analog channel as the trigger source.
- **UART/RS-232 triggering** (with Option 232) lets you trigger on RS-232 serial data.
- **USB (Universal Serial Bus) triggering** will trigger on a Start of Packet (SOP), End of Packet (EOP), Reset Complete, Enter Suspend, or Exit Suspend signal on the differential USB data lines. USB Low Speed and Full Speed are supported by this trigger.

### Reporting the Setup

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

### Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a \*RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.00000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```

## General :TRIGger Commands

**Table 67** General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0   OFF}   {1   ON}} (see <a href="#">page 409</a> )	:TRIGger:HFReject? (see <a href="#">page 409</a> )	{0   1}
:TRIGger:HOLDoff <holdoff_time> (see <a href="#">page 410</a> )	:TRIGger:HOLDoff? (see <a href="#">page 410</a> )	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:MODE <mode> (see <a href="#">page 411</a> )	:TRIGger:MODE? (see <a href="#">page 411</a> )	<mode> ::= {EDGE   GLITCh   PATtern   CAN   DURation   IIC   EBURst   LIN   SEquence   SPI   TV   USB   FLEXray} <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 412</a> )	:TRIGger:NREJect? (see <a href="#">page 412</a> )	{0   1}
:TRIGger:PATtern <value>, <mask> [, <edge source>, <edge>] (see <a href="#">page 413</a> )	:TRIGger:PATtern? (see <a href="#">page 414</a> )	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnnnnnn"; n ::= {0,...,9   A,...,F} (# bits = # channels) <edge source> ::= {CHANnel<n>   EXternal   NONE} for DSO models <edge source> ::= {CHANnel<n>   DIGital0,...,DIGital15   NONE} for MSO models <edge> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see <a href="#">page 415</a> )	:TRIGger:SWEep? (see <a href="#">page 415</a> )	<sweep> ::= {AUTO   NORMal}



**:TRIGger:HFReject**(see [page 676](#))

**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 405
  - [":TRIGger\[:EDGE\]:REJECT"](#) on page 440

**:TRIGger:HOLDoff**

**C** (see [page 676](#))

**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 405

**:TRIGger:MODE**

**C** (see [page 676](#))

**Command Syntax** :TRIGger:MODE <mode>

```
<mode> ::= {EDGE | GLITCh | PATTeRn | CAN | DURation | IIC | EBURst
            | LIN | SEQuence | SPI | TV | USB | FLEXray | UART}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

**Query Syntax** :TRIGger:MODE?

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMEbase:MODE is ROLL or XY, the query returns "NONE."

**Return Format** <mode><NL>

```
<mode> ::= {NONE | EDGE | GLIT | PATT | CAN | DUR | IIC
            | EBUR | LIN | SEQ | SPI | TV | USB | FLEX | UART}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SWEep"](#) on page 415
  - [":TIMEbase:MODE"](#) on page 395

**Example Code**

```
' TRIGGER_MODE - Set the trigger mode to EDGE, GLITCh, PATTeRn, CAN,
' DURATION, IIC, EBURSt, LIN, SEQuence, SPI, TV, USB, FLEXray, or
' UART.

' Set the trigger mode to EDGE.
myScope.WriteString ":TRIGGER:MODE EDGE"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:TRIGger:NREJect**

**C** (see [page 676](#))

**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 405

**:TRIGger:PATtern**

**C** (see [page 676](#))

**Command Syntax** :TRIGger:PATtern <pattern>

<pattern> ::= <value>, <mask> [, <edge source>, <edge>]

<value> ::= integer in NR1 format or <string>

<mask> ::= integer in NR1 format or <string>

<string> ::= "0xn timer"; n ::= {0,...,9 | A,...,F}  
(# bits = # channels, see following table)

<edge source> ::= {CHANnel<n> | EXtErnal | NONE} for DSO models

<edge source> ::= {CHANnel<n> | DIGital0,...,DIGital15  
| NONE} for MSO models

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

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

<edge> ::= {POSitive | NEGative}

The :TRIGger:PATtern command defines the specified pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
<b>2 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
<b>4 analog channels only</b>	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
<b>2 analog channels only</b>	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**NOTE**

The optional source and the optional edge should be sent together or not at all. The edge will be set in the simple pattern if it is included. If the edge source is also specified in the mask, the edge takes precedence.

**Query Syntax**     `:TRIGger:PATtern?`

The `:TRIGger:PATtern?` query returns the pattern value, the mask, and the edge of interest in the simple pattern.

**Return Format**     `<pattern><NL>`

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411

**:TRIGger:SWEep**

**C** (see [page 676](#))

**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 405

**:TRIGger:CAN Commands****Table 68** :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATtern:DATA <value>, <mask> (see <a href="#">page 418</a> )	:TRIGger:CAN:PATtern:DATA? (see <a href="#">page 418</a> )	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:CAN:PATtern:DATA:LENGth <length> (see <a href="#">page 419</a> )	:TRIGger:CAN:PATtern:DATA:LENGth? (see <a href="#">page 419</a> )	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATtern:ID <value>, <mask> (see <a href="#">page 420</a> )	:TRIGger:CAN:PATtern:ID? (see <a href="#">page 420</a> )	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:CAN:PATtern:ID:MODE <value> (see <a href="#">page 421</a> )	:TRIGger:CAN:PATtern:ID:MODE? (see <a href="#">page 421</a> )	<value> ::= {STANDARD   EXTENDED} (with Option AMS)
:TRIGger:CAN:SAMPlepo int <value> (see <a href="#">page 422</a> )	:TRIGger:CAN:SAMPlepo int? (see <a href="#">page 422</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:CAN:SIGNAL:B AUDrate <baudrate> (see <a href="#">page 423</a> )	:TRIGger:CAN:SIGNAL:B AUDrate? (see <a href="#">page 423</a> )	<baudrate> ::= {10000   20000   33300   50000   62500   83300   100000   125000   250000   500000   800000   1000000}



**Table 68** :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see <a href="#">page 424</a> )	:TRIGger:CAN:SOURce? (see <a href="#">page 424</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15  } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see <a href="#">page 425</a> )	:TRIGger:CAN:TRIGger? (see <a href="#">page 426</a> )	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF   DATA   ERROR   IDData   IDEither   IDRemote   ALLerrors   OVERload   ACKerror} (with Option AMS)

**:TRIGger:CAN:PATtern:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:PATtern:DATA <value>,<mask>

<value> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 64-bit integer in decimal, <nondecimal>, or <string>

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

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

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

The :TRIGger:CAN:PATtern:DATA command defines the CAN data pattern resource according to the value and the mask. This pattern, along with the data length (set by the :TRIGger:CAN:PATtern:DATA:LENGth command), control the data pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

**NOTE**

If more bytes are sent for <value> or <mask> than specified by the :TRIGger:CAN:PATtern:DATA:LENGth command, the most significant bytes will be truncated. If the data length is changed after the <value> and <mask> are programmed, the added or deleted bytes will be added to or deleted from the least significant bytes.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATtern:DATA?

The :TRIGger:CAN:PATtern:DATA? query returns the current settings of the specified CAN data pattern resource.

**Return Format** <value>,<mask><NL> in nondecimal format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)

• [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on [page 419](#)

• [":TRIGger:CAN:PATtern:ID"](#) on [page 420](#)

**:TRIGger:CAN:PATtern:DATA:LENGth**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:PATtern:DATA:LENGth <length>  
 <length> ::= integer from 1 to 8 in NR1 format

The :TRIGger:CAN:PATtern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIGger:CAN:PATtern:DATA command.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATtern:DATA:LENGth?

The :TRIGger:CAN:PATtern:DATA:LENGth? query returns the current CAN data pattern length setting.

**Return Format** <count><NL>  
 <count> ::= integer from 1 to 8 in NR1 format

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)  
 • [":TRIGger:CAN:PATtern:DATA"](#) on [page 418](#)  
 • [":TRIGger:CAN:SOURce"](#) on [page 424](#)

**:TRIGger:CAN:PATtern:ID**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:PATtern:ID <value>, <mask>

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

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

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

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

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

The :TRIGger:CAN:PATtern:ID command defines the CAN identifier pattern resource according to the value and the mask. This pattern, along with the identifier mode (set by the :TRIGger:CAN:PATtern:ID:MODE command), control the identifier pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the identifier pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the identifier stream. Only bits with a "1" set on the mask are used.

**NOTE**

If more bits are sent than allowed (11 bits in standard mode, 29 bits in extended mode) by the :TRIGger:CAN:PATtern:ID:MODE command, the most significant bytes will be truncated. If the ID mode is changed after the <value> and <mask> are programmed, the added or deleted bits will be added to or deleted from the most significant bits.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATtern:ID?

The :TRIGger:CAN:PATtern:ID? query returns the current settings of the specified CAN identifier pattern resource.

**Return Format** <value>, <mask><NL> in nondecimal format

**Errors** • ["-241, Hardware missing"](#) on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)

• [":TRIGger:CAN:PATtern:ID:MODE"](#) on [page 421](#)

• [":TRIGger:CAN:PATtern:DATA"](#) on [page 418](#)

**:TRIGger:CAN:PATtern:ID:MODE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:PATtern:ID:MODE <value>

<value> ::= {STANdard | EXTended}

The :TRIGger:CAN:PATtern:ID:MODE command sets the CAN identifier mode. STANdard selects the standard 11-bit identifier. EXTended selects the extended 29-bit identifier. The CAN identifier is set by the :TRIGger:CAN:PATtern:ID command.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:CAN:PATtern:ID:MODE?

The :TRIGger:CAN:PATtern:ID:MODE? query returns the current setting of the CAN identifier mode.

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

<value> ::= {STAN | EXT}

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)

- [":TRIGger:MODE"](#) on [page 411](#)
- [":TRIGger:CAN:PATtern:DATA"](#) on [page 418](#)
- [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on [page 419](#)
- [":TRIGger:CAN:PATtern:ID"](#) on [page 420](#)

**:TRIGger:CAN:SAMPlendpoint**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:SAMPlendpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger:CAN:SAMPlendpoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

**Query Syntax** :TRIGger:CAN:SAMPlendpoint?

The :TRIGger:CAN:SAMPlendpoint? query returns the current CAN sample point setting.

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

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:CAN:TRIGger"](#) on page 425

**:TRIGger:CAN:SIGNal:BAUDrate**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000  
| 125000 | 250000 | 500000 | 800000 | 1000000}

The :TRIGger:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 1 Mb/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

If the baud rate you select does not match the system baud rate, false triggers may occur.

**Query Syntax** :TRIGger:CAN:SIGNal:BAUDrate?

The :TRIGger:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

**Return Format** <baudrate><NL>

<baudrate> ::= integer = {10000 | 20000 | 33300 | 50000 | 62500  
| 83300 | 100000 | 125000 | 250000 | 500000  
| 800000 | 1000000}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:CAN:TRIGger"](#) on page 425
  - [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 630
  - [":TRIGger:CAN:SOURce"](#) on page 424

**:TRIGger:CAN:SOURce**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:SOURce <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:CAN:SOURce command sets the source for the CAN signal. The source setting is only valid when :TRIGger:CAN:TRIGger is set to SOF (start of frame).

**Query Syntax** :TRIGger:CAN:SOURce?

The :TRIGger:CAN:SOURce? query returns the current source for the CAN signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:CAN:TRIGger"](#) on page 425
  - [":TRIGger:CAN:SIGNAL:DEFinition"](#) on page 630



**:TRIGger:CAN:TRIGger**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:TRIGger <condition>

<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote |  
ALLerrors | OVERload | ACKerror}

The :TRIGger:CAN:TRIGger command sets the CAN trigger on condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- ERRor - will trigger on CAN Error frame.
- IDData - will trigger on CAN frames matching the specified Id of a Data frame.
- IDEither - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- IDRemote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	Front-panel Trigger on: softkey selection (softkey text - softkey popup text)
SOF	SOF - Start of Frame
DATA	Id & Data - Data Frame Id and Data
ERRor	Error - Error frame
IDData	Id & ~RTR - Data Frame Id (~RTR)
IDEither	Id - Remote or Data Frame Id
IDRemote	Id & RTR - Remote Frame Id (RTR)
ALLerrors	All Errors - All Errors
OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the `:TRIGger:CAN:PATtern:ID` and `:TRIGger:CAN:PATtern:ID:MODE` commands.

CAN Data specification is set by the `:TRIGger:CAN:PATtern:DATA` command.

CAN Data Length Code is set by the `:TRIGger:CAN:PATtern:DATA:LENGth` command.

**NOTE**

SOF is the only valid selection for analog oscilloscopes. If the automotive CAN and LIN serial decode option (Option AMS) has not been licensed, SOF is the only valid selection.

**Query Syntax** `:TRIGger:CAN:TRIGger?`

The `:TRIGger:CAN:TRIGger?` query returns the current CAN trigger on condition.

**Return Format** `<condition><NL>`

`<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}`

**Errors** • ["-241, Hardware missing"](#) on page 637

**See Also** • ["Introduction to :TRIGger Commands"](#) on page 405

- [":TRIGger:MODE"](#) on page 411
- [":TRIGger:CAN:PATtern:DATA"](#) on page 418
- [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 419
- [":TRIGger:CAN:PATtern:ID"](#) on page 420
- [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 421
- [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 630
- [":TRIGger:CAN:SOURce"](#) on page 424

## :TRIGger:DURation Commands

**Table 69** :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see <a href="#">page 428</a> )	:TRIGger:DURation:GREaterthan? (see <a href="#">page 428</a> )	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see <a href="#">page 429</a> )	:TRIGger:DURation:LESSthan? (see <a href="#">page 429</a> )	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:DURation:PATtern <value>, <mask> (see <a href="#">page 430</a> )	:TRIGger:DURation:PATtern? (see <a href="#">page 430</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xxxxxxxx" n ::= {0,...,9   A,...,F}
:TRIGger:DURation:QUALifier <qualifier> (see <a href="#">page 431</a> )	:TRIGger:DURation:QUALifier? (see <a href="#">page 431</a> )	<qualifier> ::= {GREaterthan   LESSthan   INRange   OUTRange   TIMEout}
:TRIGger:DURation:RANGE <greater than time>[suffix], <less than time>[suffix] (see <a href="#">page 432</a> )	:TRIGger:DURation:RANGE? (see <a href="#">page 432</a> )	<greater than time> ::= min duration from 10 ns to 9.99 seconds in NR3 format <less than time> ::= max duration from 15 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}

**:TRIGger:DURation:GREaterthan**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:DURation:GREaterthan <greater than time>[<suffix>]

```
<greater than time> ::= minimum trigger duration in seconds
                        (5 ns - 10 seconds) in NR3 format
```

```
<suffix> ::= {s | ms | us | ns | ps }
```

The `:TRIGger:DURation:GREaterthan` command sets the minimum duration for the defined pattern when `:TRIGger:DURation:QUALifier` is set to `GREaterthan`. The command also sets the timeout value when the `:TRIGger:DURation:QUALifier` is set to `TIMEout`.

### Query Syntax

The :TRIGger:DURation:GREaterthan? query returns the minimum duration time for the defined pattern.

**Return Format** <greater than time><NL>

**See Also**

- [":TRIGger:Commands"](#) on page 405
- [":TRIGger:DURation:PATtern"](#) on page 430
- [":TRIGger:DURation:QUALifier"](#) on page 431
- [":TRIGger:MODE"](#) on page 411

**:TRIGger:DURation:LESSthan**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:DURation:LESSthan <less than time>[<suffix>]

<less than time> ::= maximum trigger duration in seconds  
(5 ns - 10 seconds) in NR3 format

```
<suffix> ::= {s | ms | us | ns | ps}
```

The `:TRIGger:DURation:LESSthan` command sets the maximum duration for the defined pattern when `:TRIGger:DURation:QUALifier` is set to `LESSthan`.

### Query Syntax

The `:TRIGger:DURation:LESSthan?` query returns the duration time for the defined pattern.

**Return Format** <less than time><NL>

**See Also**

- [":TRIGger:Commands"](#) on page 405
- [":TRIGger:DURation:PATtern"](#) on page 430
- [":TRIGger:DURation:QUALifier"](#) on page 431
- [":TRIGger:MODE"](#) on page 411

**:TRIGger:DURation:PATtern**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:DURation:PATtern <value>, <mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xn timer"; n ::= {0,...,9 | A,...,F}

The :TRIGger:DURation:PATtern command defines the specified duration pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
<b>2 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
<b>4 analog channels only</b>	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
<b>2 analog channels only</b>	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**Query Syntax** :TRIGger:DURation:PATtern?

The :TRIGger:DURation:PATtern? query returns the pattern value.

**Return Format** <value>, <mask><NL>

<value> ::= a 32-bit integer in NR1 format.

<mask> ::= a 32-bit integer in NR1 format.

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:PATtern"](#) on page 413

**:TRIGger:DURation:QUALifier**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:DURation:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMEout}

The :TRIGger:DURation:QUALifier command qualifies the trigger duration.

Set the GREaterthan qualifier value with the :TRIGger:DURation:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:DURation:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:DURation:RANGE command.

Set the TIMEout qualifier value with the :TRIGger:DURation:GREaterthan command.

**Query Syntax** :TRIGger:DURation:QUALifier?

The :TRIGger:DURation:QUALifier? query returns the trigger duration qualifier.

**Return Format** <qualifier><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:DURation:GREaterthan"](#) on page 428
  - [":TRIGger:DURation:LESSthan"](#) on page 429
  - [":TRIGger:DURation:RANGE"](#) on page 432

## :TRIGger:DURation:RANGe

**N**

(see [page 676](#))

**Command Syntax**

```
:TRIGger:DURation:RANGE <greater than time>[<suffix>],  
                           <less than time>[<suffix>]
```

<greater than time> ::= minimum duration in seconds  
(10 ns - 9.99 seconds) in NR3 format

<less than time> ::= maximum duration in seconds  
(15 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}

The `:TRIGger:DURation:RANge` command sets the duration for the defined pattern when the `:TRIGger:DURation:QUALifier` command is set to `INRange` or `OUTRange`.

## NOTE

If you set the minimum duration longer than the maximum duration, the order of the parameters is automatically reversed.

### Query Syntax

The `:TRIGger:DURation:RANGe?` query returns the duration time for the defined pattern.

**Return Format** <greater than time>,<less than time><NL>

**See Also**

- ["Introduction to :TRIGger Commands" on page 405](#)
- [":TRIGger:DURation:PATtern" on page 430](#)
- [":TRIGger:DURation:QUALifier" on page 431](#)
- [":TRIGger:MODE" on page 411](#)



## :TRIGger:EBURst Commands

**Table 70** :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNT <count> (see <a href="#">page 434</a> )	:TRIGger:EBURst:COUNT? (see <a href="#">page 434</a> )	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see <a href="#">page 435</a> )	:TRIGger:EBURst:IDLE? (see <a href="#">page 435</a> )	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see <a href="#">page 436</a> )	:TRIGger:EBURst:SLOPe? (see <a href="#">page 436</a> )	<slope> ::= {NEGative   POSitive}

The :TRIGger:EDGE:SOURce command is used to specify the source channel for the Nth Edge Burst trigger. If an analog channel is selected as the source, the :TRIGger:EDGE:LEVel command is used to set the Nth Edge Burst trigger level. If a digital channel is selected as the source, the :DIGital<n>:THReshold or :POD<n>:THReshold command is used to set the Nth Edge Burst trigger level.

## :TRIGger:EBURst:COUNT

**N** (see [page 676](#))

**Command Syntax** :TRIGger:EBURst:COUNT <count>

<count> ::= integer in NR1 format

The :TRIGger:EBURst:COUNT command sets the Nth edge at burst counter resource. The edge counter is used in the trigger stage to determine which edge in a burst will generate a trigger.

**Query Syntax** :TRIGger:EBURst:COUNT?

The :TRIGger:EBURst:COUNT? query returns the current Nth edge of burst edge counter setting.

**Return Format** <count><NL>

<count> ::= integer in NR1 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:EBURst:SLOPe"](#) on page 436
  - [":TRIGger:EBURst:IDLE"](#) on page 435

**:TRIGger:EBURst:IDLE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:EBURst:IDLE <time\_value>

<time\_value> ::= time in seconds in NR3 format

The :TRIGger:EBURst:IDLE command sets the Nth edge in a burst idle resource in seconds from 10 ns to 10 s. The timer is used to set the minimum time before the next burst.

**Query Syntax** :TRIGger:EBURst:IDLE?

The :TRIGger:EBURst:IDLE? query returns current Nth edge in a burst idle setting.

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

<time\_value> ::= time in seconds in NR3 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:EBURst:SLOPe"](#) on page 436
  - [":TRIGger:EBURst:COUnT"](#) on page 434

**:TRIGger:EBURst:SLOPe**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:EBURst:SLOPe <slope>  
 <slope> ::= {NEGative | POSitive}

The :TRIGger:EBURst:SLOPe command specifies whether the rising edge (POSitive) or falling edge (NEGative) of the Nth edge in a burst will generate a trigger.

**Query Syntax** :TRIGger:EBURst:SLOPe?

The :TRIGger:EBURst:SLOPe? query returns the current Nth edge in a burst slope.

**Return Format** <slope><NL>  
 <slope> ::= {NEG | POS}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:EBURst:IDLE"](#) on page 435
  - [":TRIGger:EBURst:COUNt"](#) on page 434

## :TRIGger[:EDGE] Commands

**Table 71** :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC   DC   LF} (see <a href="#">page 438</a> )	:TRIGger[:EDGE]:COUPling? (see <a href="#">page 438</a> )	{AC   DC   LF}
:TRIGger[:EDGE]:LEVel <level> [, <source>] (see <a href="#">page 439</a> )	:TRIGger[:EDGE]:LEVel ? [<source>] (see <a href="#">page 439</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. For digital channels (MSO models), <level> ::= 8 V. <source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXTERNAL} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF   LF   HF} (see <a href="#">page 440</a> )	:TRIGger[:EDGE]:REJect? (see <a href="#">page 440</a> )	{OFF   LF   HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see <a href="#">page 441</a> )	:TRIGger[:EDGE]:SLOPe ? (see <a href="#">page 441</a> )	<polarity> ::= {POSitive   NEGative   EITHER   ALTERNate}
:TRIGger[:EDGE]:SOURC e <source> (see <a href="#">page 442</a> )	:TRIGger[:EDGE]:SOURC e? (see <a href="#">page 442</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15   EXTERNAL} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

**:TRIGger[:EDGE]:COUPling**(see [page 676](#))**Command Syntax** :TRIGger[:EDGE]:COUPling <coupling>

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

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger\[:EDGE\]:REJect"](#) on page 440

**:TRIGger[:EDGE]:LEVel**

**C** (see [page 676](#))

**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> ::= 2 V with probe attenuation at 1:1 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> | DIGital0,...,DIGital15 | EXTernal} for the MSO models

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

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

The :TRIGger[: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 405
  - [":TRIGger\[:EDGE\]:SOURce"](#) on page 442
  - [":POD<n>:THReshold"](#) on page 336
  - [":DIGital<n>:THReshold"](#) on page 229

**:TRIGger[:EDGE]:REJect**

**C** (see [page 676](#))

**Command Syntax**    :TRIGger[:EDGE]:REJect <reject>  
                          <reject> ::= {OFF | LFRreject | HFRreject}

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 405
  - "[:TRIGger:HFRreject](#)" on page 409
  - "[:TRIGger\[:EDGE\]:COUPling](#)" on page 438



**:TRIGger[:EDGE]:SLOPe**

**C** (see [page 676](#))

**Command Syntax** :TRIGger[:EDGE]:SLOPe <slope>  
 <slope> ::= {NEGative | POSitive | EITHer | ALTerenate}

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 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:TV:POLarity"](#) on page 501

**Example Code**

```
' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:TRIGger[:EDGE]:SOURce**

**C** (see [page 676](#))

**Command Syntax** :TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXternal | LINE} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15 | EXternal | LINE}  
for the MSO models

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

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

The :TRIGger[:EDGE]:SOURce command selects the channel that produces the trigger.

**Query Syntax** :TRIGger[:EDGE]:SOURce?

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

**Return Format** <source><NL>

<source> ::= {CHAN<n> | EXT | LINE | NONE} for the DSO models

<source> ::= {CHAN<n> | DIG0,...,DIG15 | EXternal | LINE | NONE}  
for the MSO models

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411

**Example Code**

```
' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces the
edge trigger. Any channel can be selected.
myScope.WriteString ":TRIGGER:EDGE:SOURCE CHANNEL1"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

## :TRIGger:FLEXray Commands

**Table 72** :TRIGger:FLEXray Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FLEXray:ERRor:TYPE <error_type> (see <a href="#">page 444</a> )	:TRIGger:FLEXray:ERRor:TYPE? (see <a href="#">page 444</a> )	<error_type> ::= {ALL   CODE   TSS   HCRC   FCRC   END   BOUNDary   IDLE   SYMbol   SLOT   NULL   SOS   FID   CCOunt   PLENgth}
:TRIGger:FLEXray:FRAMe:CBase <cycle_count_base> (see <a href="#">page 446</a> )	:TRIGger:FLEXray:FRAMe:CBase? (see <a href="#">page 446</a> )	<cycle_count_base> ::= integer from 0-63
:TRIGger:FLEXray:FRAMe:CRepetition <cycle_count_repetition> (see <a href="#">page 447</a> )	:TRIGger:FLEXray:FRAMe:CRepetition? (see <a href="#">page 447</a> )	<cycle_count_repetition> ::= {ALL   <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:FRAMe:ID <frame_id> (see <a href="#">page 448</a> )	:TRIGger:FLEXray:FRAMe:ID? (see <a href="#">page 448</a> )	<frame_id> ::= {ALL   <frame #>} <frame #> ::= integer from 1-2047
:TRIGger:FLEXray:FRAMe:TYPE <frame_type> (see <a href="#">page 449</a> )	:TRIGger:FLEXray:FRAMe:TYPE? (see <a href="#">page 449</a> )	<frame_type> ::= {NORMal   STARTup   NULL   SYNC   NSTartup   NNULl   NSYNc   ALL}
:TRIGger:FLEXray:TIME:CBase <cycle_base> (see <a href="#">page 450</a> )	:TRIGger:FLEXray:TIME:CBase? (see <a href="#">page 450</a> )	<cycle_base> ::= integer from 0-63
:TRIGger:FLEXray:TIME:CREPetition <cycle_repetition> (see <a href="#">page 451</a> )	:TRIGger:FLEXray:TIME:CREPetition? (see <a href="#">page 451</a> )	<cycle_repetition> ::= {ALL   <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:TIME:SEGMENT <segment_type> (see <a href="#">page 452</a> )	:TRIGger:FLEXray:TIME:SEGMENT? (see <a href="#">page 452</a> )	<segment_type> ::= {STATic   DYNamic   SYMbol   IDLE}
:TRIGger:FLEXray:TIME:SLOT <slot_type>, <slot_id> (see <a href="#">page 453</a> )	:TRIGger:FLEXray:TIME:SLOT? (see <a href="#">page 453</a> )	<slot_type> ::= {ALL   EMPTY} <slot_id> ::= {ALL   <slot #>} <slot #> ::= integer from 1-2047
:TRIGger:FLEXray:TRIGger <condition> (see <a href="#">page 454</a> )	:TRIGger:FLEXray:TRIGger? (see <a href="#">page 454</a> )	<condition> ::= {FRAME   TIME   ERRor}

**:TRIGger:FLEXray:ERRor:TYPE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:ERRor:TYPE <error\_type>

```
<error_type> ::= {ALL | CODE | TSS | HCRC | FCRC | END | BOUNDary |
                  IDLE | SYMBol | SLOT | NULL | SOS | FID | CCOunt |
                  PLENgth}
```

Selects the FlexRay error type to trigger on. The error type setting is only valid when the FlexRay trigger mode is set to ERROR.

- ALL – triggers on ALL errors.
- CODE – triggers on only CODE errors (NRZ).
- TSS – triggers on only TSS violations.
- HCRC – triggers on only Header CRC errors.
- FCRC – triggers on only Frame CRC errors.
- END – triggers on only frame END sequence errors.

The following error types are NOT valid when the BUSDoctor is in ASYNchronous mode:

- BOUNDary – triggers on only BOUNDary violations.
- IDLE – triggers only on Network IDLE time violations.
- SYMBol – triggers only on SYMBol window violations.
- SLOT – triggers only on SLOT overbooked errors.
- NULL – triggers only on NULL frame errors.
- SOS – triggers only on Sync Or Startup errors.
- FID – triggers only on Frame ID errors.
- CCOunt – triggers only on Cycle Count errors.
- PLENgth – triggers only on static Payload LENgth errors.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:ERRor:TYPE?

The :TRIGger:FLEXray:ERRor:TYPE? query returns the currently selected ified FLEXray error type.

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

```
<error_type> ::= {ALL | CODE | TSS | HCRC | FCRC | END | BOUN |
                  IDLE | SYM | SLOT | NULL | SOS | FID | CCO |
                  PLEN}
```

- Errors**
- ["-241, Hardware missing"](#) on page 637
- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:FRAME:CCBase**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:FRAME:CCBase <cycle\_count\_base>  
 <cycle\_count\_base> ::= integer from 0-63

The :TRIGger:FLEXray:FRAME:CCBase command sets the base of the FlexRay cycle count (in the frame header) to trigger on. The cycle count base setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:FRAME:CCBase?

The :TRIGger:FLEXray:FRAME:CCBase? query returns the current cycle count base setting for the FlexRay frame trigger setup.

**Return Format** <cycle\_count\_base><NL>  
 <cycle\_count\_base> ::= integer from 0-63

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • "Introduction to :TRIGger Commands" on [page 405](#)  
 • ":TRIGger:FLEXray:TRIGger" on [page 454](#)

**:TRIGger:FLEXray:FRAME:CCRepetition**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:FRAME:CCRepetition <cycle\_count\_repetition>  
 <cycle\_count\_repetition> ::= {ALL | <rep #>}  
 <rep #> ::= integer from 2-64

The :TRIGger:FLEXray:FRAME:CCRepetition command sets the repetition number of the FlexRay cycle count (in the frame header) to trigger on. The cycle count repetition setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:FRAME:CCRepetition?

The :TRIGger:FLEXray:FRAME:CCRepetition? query returns the current cycle count repetition setting for the FlexRay frame trigger setup.

**Return Format** <cycle\_count\_repetition><NL>  
 <cycle\_count\_repetition> ::= {ALL | <rep #>}  
 <rep #> ::= integer from 2-64

**Errors** • "-241, Hardware missing" on page 637

**See Also** • "Introduction to :TRIGger Commands" on page 405  
 • ":TRIGger:FLEXray:TRIGger" on page 454

**:TRIGger:FLEXray:FRAME:ID**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:FLEXray:FRAME:ID <frame\_id>  
                          <frame\_id> ::= {ALL | <frame #>}  
                          <frame #> ::= integer from 1-2047

The :TRIGger:FLEXray:FRAME:ID command sets the FlexRay frame ID to trigger on . The frame IF setting is only valid when the FlexRay trigger mode is set to FRAME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax**       :TRIGger:FLEXray:FRAME:ID?

The :TRIGger:FLEXray:FRAME:ID? query returns the current frame ID setting for the FlexRay frame trigger setup.

**Return Format**      <frame\_id><NL>  
                          <frame\_id> ::= {ALL | <frame #>}  
                          <frame #> ::= integer from 1-2047

**Errors**            • "-241, Hardware missing" on [page 637](#)

**See Also**          • ["Introduction to :TRIGger Commands"](#) on [page 405](#)  
                          • [":TRIGger:MODE"](#) on [page 411](#)  
                          • [":TRIGger:FLEXray:TRIGger"](#) on [page 454](#)



**:TRIGger:FLEXray:FRAME:TYPE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:FRAME:TYPE <frame\_type>

```
<frame_type> ::= {NORMal | STARTup | NULL | SYNC | NSTArtup | NNULl |
                  NSYNc | ALL}
```

The :TRIGger:FLEXray:FRAME:TYPE command sets the FlexRay frame type to trigger on. The frame type setting is only valid when the FlexRay trigger mode is set to FRAME.

- NORMal – will trigger on only normal (NSTArtup & NNULl & NSYNc) frames.
- STARTup – will trigger on only startup frames.
- NULL – will trigger on only null frames.
- SYNC – will trigger on only sync frames.
- NSTArtup – will trigger on frames other than startup frames.
- NNULl – will trigger on frames other than null frames.
- NSYNc – will trigger on frames other than sync frames.
- ALL – will trigger on all FlexRay frame types.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:FRAME:TYPE?

The :TRIGger:FLEXray:FRAME:TYPE? query returns the current frame type setting for the FlexRay frame trigger setup.

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

```
<frame_type> ::= {NORM | STAR | NULL | SYNC | NSTA | NNUL |
                  NSYN | ALL}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:TIME:CBASe**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:TIME:CBASe <cycle\_base>  
 <cycle\_base> ::= integer from 0-63

The :TRIGger:FLEXray:TIME:CBASe command sets the base of the FlexRay cycle to trigger on. The cycle base setting is only valid when the FlexRay trigger mode is set to TIME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:TIME:CBASe?

The :TRIGger:FLEXray:TIME:CBASe? query returns the current cycle base setting for the FlexRay time trigger setup.

**Return Format** <cycle\_base><NL>  
 <cycle\_base> ::= integer from 0-63

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:TIME:CREPetition**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:TIME:CREPetition <cycle\_repetition>

<cycle\_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer from 2-64

The :TRIGger:FLEXray:TIME:CREPetition command sets the repetition number of the FlexRay cycle to trigger on. The cycle repetition setting is only valid when the FlexRay trigger mode is set to TIME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:TIME:CREPetition?

The :TRIGger:FLEXray:TIME:CREPetition? query returns the current cycle repetition setting for the FlexRay time trigger setup.

**Return Format** <cycle\_repetition><NL>

<cycle\_repetition> ::= {ALL | <rep #>}

<rep #> ::= integer from 2-64

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:TIME:SEGment**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:TIME:SEGment <segment\_type>  
 <segment\_type> ::= {STATic | DYNamic | SYMbol | IDLE}

The :TRIGger:FLEXray:TIME:SEGment command sets the FlexRay segment type. The segment setting is only valid when the FlexRay trigger mode is set to TIME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:TIME:SEGment?

The :TRIGger:FLEXray:TIME:SEGment? query returns the current segment setting for the FlexRay time trigger setup.

**Return Format** <segment\_type><NL>  
 <segment\_type> ::= {STAT | DYN | SYM | IDLE}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:TIME:SLOT**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:FLEXray:TIME:SLOT <slot\_type>, <slot\_id>

                         <slot\_type> ::= {ALL | EMPTY}

                         <slot\_id> ::= {ALL | <slot #>}

                         <slot #> ::= integer from 1-2047

The :TRIGger:FLEXray:TIME:SLOT command sets the FlexRay slot type and ID. The slot setting is only valid when the FlexRay trigger mode is set to TIME.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax**        :TRIGger:FLEXray:TIME:SLOT?

The :TRIGger:FLEXray:TIME:SLOT? query returns the current source for the FLEXray signal.

**Return Format**       <slot\_type>, <slot\_id><NL>

                         <slot\_type> ::= {ALL | EMPTY}

                         <slot\_id> ::= {ALL | <slot #>}

                         <slot #> ::= integer from 1-2047

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454

**:TRIGger:FLEXray:TRIGger**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:FLEXray:TRIGger <condition>

<condition> ::= {FRAME | TIME | ERROR}

The :TRIGger:FLEXray:TRIGger command sets the FLEXray trigger on condition:

- FRAME — triggers on specified frames (without errors).
- TIME — triggers on specified bus cycles, segments, and slots.
- ERROR — triggers on selected active error frames and unknown bus conditions.

**NOTE**

This command is only valid when the FLEXray triggering and serial decode option (Option FRS) has been licensed.

**Query Syntax** :TRIGger:FLEXray:TRIGger?

The :TRIGger:FLEXray:TRIGger? query returns the current FLEXray trigger on condition.

**Return Format** <condition><NL>

<condition> ::= {FRAM | TIME | ERR}

**Errors** • ["-241, Hardware missing"](#) on [page 637](#)

- See Also**
- ["Introduction to :TRIGger Commands"](#) on [page 405](#)
  - [":TRIGger:MODE"](#) on [page 411](#)
  - [":TRIGger:FLEXray:ERROR:TYPE"](#) on [page 444](#)
  - [":TRIGger:FLEXray:FRAME:CCBase"](#) on [page 446](#)
  - [":TRIGger:FLEXray:FRAME:CCRepetition"](#) on [page 447](#)
  - [":TRIGger:FLEXray:FRAME:ID"](#) on [page 448](#)
  - [":TRIGger:FLEXray:FRAME:TYPE"](#) on [page 449](#)
  - [":TRIGger:FLEXray:TIME:CBase"](#) on [page 450](#)
  - [":TRIGger:FLEXray:TIME:CREpetition"](#) on [page 451](#)
  - [":TRIGger:FLEXray:TIME:SEGment"](#) on [page 452](#)
  - [":TRIGger:FLEXray:TIME:SLOT"](#) on [page 453](#)

## :TRIGger:GLITch Commands

**Table 73** :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREAt erthan <greater than time>[suffix] (see <a href="#">page 457</a> )	:TRIGger:GLITch:GREAt erthan? (see <a href="#">page 457</a> )	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:LESSt han <less than time>[suffix] (see <a href="#">page 458</a> )	:TRIGger:GLITch:LESSt han? (see <a href="#">page 458</a> )	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see <a href="#">page 459</a> )	:TRIGger:GLITch:LEVel ? (see <a href="#">page 459</a> )	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. For digital channels (MSO models), <level> ::= 6 V. <source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLar ity <polarity> (see <a href="#">page 460</a> )	:TRIGger:GLITch:POLar ity? (see <a href="#">page 460</a> )	<polarity> ::= {POSitive   NEGative}
:TRIGger:GLITch:QUALi fier <qualifier> (see <a href="#">page 461</a> )	:TRIGger:GLITch:QUALi fier? (see <a href="#">page 461</a> )	<qualifier> ::= {GREATERthan   LESSthan   RANGE}

**Table 73** :TRIGger:GLITCh Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITCh:RANGe <greater than time>[suffix], <less than time>[suffix] (see <a href="#">page 462</a> )	:TRIGger:GLITCh:RANGe? (see <a href="#">page 462</a> )	<greater than time> ::= start time from 10 ns to 9.99 seconds in NR3 format <less than time> ::= stop time from 15 ns to 10 seconds in NR3 format [suffix] ::= {s   ms   us   ns   ps}
:TRIGger:GLITCh:SOURCe <source> (see <a href="#">page 463</a> )	:TRIGger:GLITCh:SOURCe? (see <a href="#">page 463</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format



**:TRIGger:GLITch:GREaterthan**

**N** (see [page 676](#))

### Command Syntax

**:TRIGger:GLITch:GREaterthan** <greater\_than\_time>[<suffix>]

```
<greater_than_time> ::= 32-bit floating-point number (5 ns - 10 seconds)
                        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

The :TRIGger:GLITch:GREaterthan? query returns the minimum pulse width duration time for :TRIGger:GLITch:SOURce.

**Return Format** <greater than time><NL>.

**See Also**

- [":TRIGger Commands"](#) on page 405
- [":TRIGger:GLITch:SOURce"](#) on page 463
- [":TRIGger:GLITch:QUALifier"](#) on page 461
- [":TRIGger:MODE"](#) on page 411

**:TRIGger:GLITch:LESSthan**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:GLITch:LESSthan <less\_than\_time>[<suffix>]  
 <less\_than\_time> ::= floating-point number (5 ns - 10 seconds)  
 <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> ::= a 32-bit floating-point number in NR3 format.

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:GLITch:SOURce"](#) on page 463
  - [":TRIGger:GLITch:QUALifier"](#) on page 461
  - [":TRIGger:MODE"](#) on page 411

**:TRIGger:GLITCh:LEVel**

**N** (see [page 676](#))

**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> ::= 2 V with probe attenuation at 1:1 in NR3 format for external triggers

<level> ::= 6 V for digital channels (MSO models)

<source> ::= {CHANnel<n> | EXTernal} for DSO models

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for MSO models

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

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

The :TRIGger:GLITCh:LEVel command sets the trigger level voltage for the active pulse width trigger.

**Query Syntax** :TRIGger:GLITCh:LEVel?

The :TRIGger:GLITCh:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:GLITCh:SOURce"](#) on page 463

**:TRIGger:GLITch:POLarity****N** (see [page 676](#))

**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 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:GLITch:SOURce"](#) on page 463

**:TRIGger:GLITch:QUALifier**

**N** (see [page 676](#))

**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 405
  - [":TRIGger:GLITch:SOURce"](#) on page 463
  - [":TRIGger:MODE"](#) on page 411



**:TRIGger:GLITch:SOURce**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:GLITch:SOURce <source>

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

<source> ::= {DIGital0,...,DIGital15 | CHANnel<n>} for the MSO models

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

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

The :TRIGger: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 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:GLITch:LEVel"](#) on page 459
  - [":TRIGger:GLITch:POLarity"](#) on page 460
  - [":TRIGger:GLITch:QUALifier"](#) on page 461
  - [":TRIGger:GLITch:RANGe"](#) on page 462

**Example Code** • ["Example Code"](#) on page 442

**:TRIGger:IIC Commands****Table 74** :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATtern:ADDRess <value> (see <a href="#">page 465</a> )	:TRIGger:IIC:PATtern:ADDRess? (see <a href="#">page 465</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATtern:DATA <value> (see <a href="#">page 466</a> )	:TRIGger:IIC:PATtern:DATA? (see <a href="#">page 466</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC:PATtern:DATA2 <value> (see <a href="#">page 467</a> )	:TRIGger:IIC:PATtern:DATA2? (see <a href="#">page 467</a> )	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9   A,...,F}
:TRIGger:IIC[:SOURce]:CLOCK <source> (see <a href="#">page 468</a> )	:TRIGger:IIC[:SOURce]:CLOCK? (see <a href="#">page 468</a> )	<source> ::= {CHANnel<n>   EXTeRnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see <a href="#">page 469</a> )	:TRIGger:IIC[:SOURce]:DATA? (see <a href="#">page 469</a> )	<source> ::= {CHANnel<n>   EXTeRnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see <a href="#">page 470</a> )	:TRIGger:IIC:TRIGger:QUALifier? (see <a href="#">page 470</a> )	<value> ::= {EQUal   NOTequal   LESSthan   GREATERthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see <a href="#">page 471</a> )	:TRIGger:IIC:TRIGger[:TYPE]? (see <a href="#">page 471</a> )	<type> ::= {STARt   STOP   READ7   READEprom   WRITe7   WRITe10   NACKnowledge   ANACKnowledge   R7Data2   W7Data2   REStArt}



**:TRIGger:IIC:PATtern:ADDress**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:PATtern:ADDress <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC:PATtern:ADDress command sets the address for IIC data. The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

**Query Syntax** :TRIGger:IIC:PATtern:ADDress?

The :TRIGger:IIC:PATtern:ADDress? query returns the current address for IIC data.

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

<value> ::= integer

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:IIC:PATtern:DATA"](#) on page 466
  - [":TRIGger:IIC:PATtern:DATA2"](#) on page 467
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471

**:TRIGger:IIC:PATtern:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:PATtern:DATA <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC:PATtern:DATA command sets IIC data. The data value can range from 0x00 to 0xFF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

**Query Syntax** :TRIGger:IIC:PATtern:DATA?

The :TRIGger:IIC:PATtern:DATA? query returns the current pattern for IIC data.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:IIC:PATtern:ADDRESS"](#) on page 465
  - [":TRIGger:IIC:PATtern:DATA2"](#) on page 467
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471

**:TRIGger:IIC:PATtern:DATa2**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:PATtern:DATa2 <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC:PATtern:DATa2 command sets IIC data 2. The data value can range from 0x00 to 0xFF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

**Query Syntax** :TRIGger:IIC:PATtern:DATa2?

The :TRIGger:IIC:PATtern:DATa2? query returns the current pattern for IIC data 2.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:IIC:PATtern:ADDRes"](#) on page 465
  - [":TRIGger:IIC:PATtern:DATA"](#) on page 466
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471

**:TRIGger:IIC:SOURce:CLOCK**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:[SOURce:]CLOCK <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:IIC:[SOURce:]CLOCK command sets the source for the IIC serial clock (SCL).

**Query Syntax** :TRIGger:IIC:[SOURce:]CLOCK?

The :TRIGger:IIC:[SOURce:]CLOCK? query returns the current source for the IIC serial clock.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:IIC:SOURce:DATA"](#) on page 469

**:TRIGger:IIC:SOURce:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:[SOURce:]DATA <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:IIC:[SOURce:]DATA command sets the source for IIC serial data (SDA).

**Query Syntax** :TRIGger:IIC:[SOURce:]DATA?

The :TRIGger:IIC:[SOURce:]DATA? query returns the current source for IIC serial data.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 468

**:TRIGger:IIC:TRIGger:QUALifier**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:TRIGger:QUALifier <value>  
 <value> ::= {EQUAL | NOTequal | LESSthan | GREATERthan}

The :TRIGger:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEprom.

**Query Syntax** :TRIGger:IIC:TRIGger:QUALifier?

The :TRIGger:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

**Return Format** <value><NL>  
 <value> ::= {EQUAL | NOTequal | LESSthan | GREATERthan}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471

**:TRIGger:IIC:TRIGger[:TYPE]**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:IIC:TRIGger[:TYPE] <value>

<value> ::= {START | STOP | READ7 | READEprom | WRITe7 | WRITe10  
| NACKnowledge | ANACKnowledge | R7Data2 | W7Data2 | REStart}

The :TRIGger:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- START – Start condition.
- STOP – Stop condition.
- READ7 – 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- R7Data2 – 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- READEprom – EEPROM data read.
- WRITe7 – 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITE is also accepted for WRITe7.
- W7Data2 – 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- WRITe10 – 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- NACKnowledge – Missing acknowledge.
- ANACKnowledge – Address with no acknowledge.
- REStart – Another start condition occurs before a stop condition.

**NOTE**

The short form of READ7 (READ7), READEprom (READE), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules (see [page 678](#)).

**Query Syntax** :TRIGger:IIC:TRIGger[:TYPE] ?

The :TRIGger:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

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

<value> ::= {STAR | STOP | READ7 | READE | WRIT7 | WRIT10 | NACK | ANAC  
| R7D2 | W7D2 | REST}

- See Also**
- "Introduction to :TRIGger Commands" on page 405
  - ":TRIGger:MODE" on page 411

- [":TRIGger:IIC:PAiTern:ADDRess"](#) on page 465
- [":TRIGger:IIC:PAiTern:DATA"](#) on page 466
- [":TRIGger:IIC:PAiTern:DATA2"](#) on page 467
- [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 470
- ["Long Form to Short Form Truncation Rules"](#) on page 678



## :TRIGger:LIN Commands

**Table 75** :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see <a href="#">page 474</a> )	:TRIGger:LIN:ID? (see <a href="#">page 474</a> )	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary <string> ::= "0xnn" where n ::= {0,...,9   A,...,F} for hexadecimal
:TRIGger:LIN:SAMPlEpo int <value> (see <a href="#">page 475</a> )	:TRIGger:LIN:SAMPlEpo int? (see <a href="#">page 475</a> )	<value> ::= {60   62.5   68   70   75   80   87.5} in NR3 format
:TRIGger:LIN:SIGNAL:B AUDrate <baudrate> (see <a href="#">page 476</a> )	:TRIGger:LIN:SIGNAL:B AUDrate? (see <a href="#">page 476</a> )	<baudrate> ::= {2400   9600   19200}
:TRIGger:LIN:SOURce <source> (see <a href="#">page 477</a> )	:TRIGger:LIN:SOURce? (see <a href="#">page 477</a> )	<source> ::= {CHANnel<n>   EXTernal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANDard <std> (see <a href="#">page 478</a> )	:TRIGger:LIN:STANDard ? (see <a href="#">page 478</a> )	<std> ::= {LIN13   LIN20}
:TRIGger:LIN:SYNCbrea k <value> (see <a href="#">page 479</a> )	:TRIGger:LIN:SYNCbrea k? (see <a href="#">page 479</a> )	<value> ::= integer = {11   12   13}
:TRIGger:LIN:TRIGger <condition> (see <a href="#">page 480</a> )	:TRIGger:LIN:TRIGger? (see <a href="#">page 480</a> )	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak   ID} (with Option AMS)

**:TRIGger:LIN:ID**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>  
from 0-63 or 0x00-0x3f

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

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

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

The :TRIGger:LIN:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

**NOTE**

This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

**Query Syntax** :TRIGger:LIN:ID?

The :TRIGger:LIN:ID? query returns the current LIN identifier setting.

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

<value> ::= integer in decimal

**Errors** • "-241, Hardware missing" on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)  
• [":TRIGger:MODE"](#) on [page 411](#)  
• [":TRIGger:LIN:TRIGger"](#) on [page 480](#)  
• [":TRIGger:LIN:SIGNAL:DEFinition"](#) on [page 631](#)  
• [":TRIGger:LIN:SOURce"](#) on [page 477](#)

**:TRIGger:LIN:SAMPlpoint**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:SAMPlpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger:LIN:SAMPlpoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

**NOTE**

The sample point values are not limited by the baud rate.

**Query Syntax** :TRIGger:LIN:SAMPlpoint?

The :TRIGger:LIN:SAMPlpoint? query returns the current LIN sample point setting.

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

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:TRIGger"](#) on page 480

**:TRIGger:LIN:SIGNal:BAUDrate**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {2400 | 9600 | 19200}

The :TRIGger:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal at 2400 b/s, 9600 b/s, or 19200 b/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

**Query Syntax** :TRIGger:LIN:SIGNal:BAUDrate?

The :TRIGger:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

**Return Format** <baudrate><NL>

<baudrate> ::= integer = {2400 | 9600 | 19200}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:TRIGger"](#) on page 480
  - [":TRIGger:LIN:SIGNal:DEFinition"](#) on page 631
  - [":TRIGger:LIN:SOURce"](#) on page 477

**:TRIGger:LIN:SOURce**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:SOURce <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:LIN:SOURce command sets the source for the LIN signal.

**Query Syntax** :TRIGger:LIN:SOURce?

The :TRIGger:LIN:SOURce? query returns the current source for the LIN signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:TRIGger"](#) on page 480
  - [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 631

### :TRIGger:LIN:STANdard

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:STANdard <std>  
<std> ::= {LIN13 | LIN20}

The :TRIGger:LIN:STANdard command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

**Query Syntax** :TRIGger:LIN:STANdard?

The :TRIGger:LIN:STANdard? query returns the current LIN standard setting.

**Return Format** <std><NL>  
<std> ::= {LIN13 | LIN20}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 631
  - [":TRIGger:LIN:SOURce"](#) on page 477

**:TRIGger:LIN:SYNCbreak**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:SYNCbreak <value>  
 <value> ::= integer = {11 | 12 | 13}

The :TRIGger:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11, 12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

**Query Syntax** :TRIGger:LIN:SYNCbreak?

The :TRIGger:LIN:STANDARD? query returns the current LIN sync break setting.

**Return Format** <value><NL>  
 <value> ::= {11 | 12 | 13}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 631
  - [":TRIGger:LIN:SOURce"](#) on page 477

**:TRIGger:LIN:TRIGger**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:TRIGger <condition>  
 <condition> ::= {SYNCbreak | ID}

The :TRIGger:LIN:TRIGger command sets the LIN trigger on condition to be Sync Break (SYNCbreak) or Frame Id (ID).

**NOTE**

The ID option is available when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

**Query Syntax** :TRIGger:LIN:TRIGger?

The :TRIGger:LIN:TRIGger? query returns the current LIN trigger value.

**Return Format** <condition><NL>  
 <condition> ::= {SYNC | ID}

**Errors** • ["-241, Hardware missing"](#) on [page 637](#)

**See Also** • ["Introduction to :TRIGger Commands"](#) on [page 405](#)  
 • [":TRIGger:MODE"](#) on [page 411](#)  
 • [":TRIGger:LIN:SIGNAL:DEFinition"](#) on [page 631](#)  
 • [":TRIGger:LIN:SOURce"](#) on [page 477](#)



## :TRIGger:SEQuence Commands

**Table 76** :TRIGger:SEQuence Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SEQuence:COU Nt <count> (see <a href="#">page 482</a> )	:TRIGger:SEQuence:COU Nt? (see <a href="#">page 482</a> )	<count> ::= integer in NR1 format
:TRIGger:SEQuence:EDG E{1 2} <source>, <slope> (see <a href="#">page 483</a> )	:TRIGger:SEQuence:EDG E{1 2}? (see <a href="#">page 483</a> )	<source> ::= {CHANnel<n>   EXTErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <slope> ::= {POSitive   NEGative} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= query returns "NONE" if edge source is disabled
:TRIGger:SEQuence:FIN D <value> (see <a href="#">page 484</a> )	:TRIGger:SEQuence:FIN D? (see <a href="#">page 484</a> )	<value> ::= {PATTErn1,ENTERed   PATTErn1,EXITed   EDGE1   PATTErn1,AND,EDGE1}
:TRIGger:SEQuence:PAT Tern{1 2} <value>, <mask> (see <a href="#">page 485</a> )	:TRIGger:SEQuence:PAT Tern{1 2}? (see <a href="#">page 485</a> )	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn timer" n ::= {0,...,9   A,...,F}
:TRIGger:SEQuence:RES et <value> (see <a href="#">page 486</a> )	:TRIGger:SEQuence:RES et? (see <a href="#">page 486</a> )	<value> ::= {NONE   PATTErn1,ENTERed   PATTErn1,EXITed   EDGE1   PATTErn1,AND,EDGE1   PATTErn2,ENTERed   PATTErn2,EXITed   EDGE2   TIMer} Values used in find and trigger stages not available. EDGE2 not available if EDGE2,COUNT used in trigger stage.
:TRIGger:SEQuence:TIM er <time_value> (see <a href="#">page 487</a> )	:TRIGger:SEQuence:TIM er? (see <a href="#">page 487</a> )	<time_value> ::= time from 100 ns to 10 seconds in NR3 format
:TRIGger:SEQuence:TRI Gger <value> (see <a href="#">page 488</a> )	:TRIGger:SEQuence:TRI Gger? (see <a href="#">page 488</a> )	<value> ::= {PATTErn2,ENTERed   PATTErn2,EXITed   EDGE2   PATTErn2,AND,EDGE2   EDGE2,COUNT   EDGE2,COUNT,NREFind}

**:TRIGger:SEQuence:COUNT**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:SEQuence:COUNT <count>  
                          <count> ::= integer in NR1 format

The :TRIGger:SEQuence:COUNT command sets the sequencer edge counter resource. The edge counter is used in the trigger stage to determine the number of edges that must be found before the sequencer generates a trigger.

**Query Syntax**       :TRIGger:SEQuence:COUNT?

The :TRIGger:SEQuence:COUNT? query returns the current sequencer edge counter setting.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488
  - [":TRIGger:SEQuence:EDGE"](#) on page 483

**:TRIGger:SEQuence:EDGE**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:SEQuence:EDGE{1 | 2} <source>, <slope>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

<slope> ::= {POSitive | NEGative}

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

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

The :TRIGger:SEQuence:EDGE<n> command defines the specified sequencer edge resource according to the specified <source> and <slope>. To disable an edge resource, set its <source> to NONE. In this case, <slope> has no meaning.

**Query Syntax**    :TRIGger:SEQuence:EDGE{1 | 2}?

The :TRIGger:SEQuence:EDGE<n>? query returns the specified sequencer edge resource setting. If the edge resource is disabled, the returned <source> value is NONE. In this case, the <slope> is undefined.

**Return Format**    <source>, <slope><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:FIND"](#) on page 484
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488
  - [":TRIGger:SEQuence:RESet"](#) on page 486
  - [":TRIGger:SEQuence:COUNt"](#) on page 482

**:TRIGger:SEQuence:FIND**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SEQuence:FIND <value>

<value> ::= {PATTErn1,ENTerEd | PATTErn1,EXITed | EDGE1  
| PATTErn1,AND,EDGE1}

The :TRIGger:SEQuence:FIND command specifies the find stage of a sequence trigger. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example,"EDGE1,NONE,NONE").

PATTErn1 is specified with the":TRIGger:SEQuence:PATTErn command. EDGE1 is specified with the :TRIGger:SEQuence:EDGE command.

**Query Syntax** :TRIGger:SEQuence:FIND?

The :TRIGger:SEQuence:FIND? query returns the find stage specification for a sequence trigger. NONE is returned for unused parameters.

**Return Format** <find\_value><NL>

<find\_value> ::= {PATT1,ENT,NONE | PATT1,EXIT,NONE | EDGE1,NONE,NONE  
| PATT1,AND,EDGE1}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:PATTErn"](#) on page 485
  - [":TRIGger:SEQuence:EDGE"](#) on page 483
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488
  - [":TRIGger:SEQuence:RESet"](#) on page 486

**:TRIGger:SEQuence:PATtern**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SEQuence:PATtern{1 | 2} <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xn timer" where n ::= {0,...,9 | A,...,F}

The :TRIGger:SEQuence:PATtern<n> command defines the specified sequence pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
<b>4 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
<b>2 analog + 16 digital channels (mixed-signal)</b>	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
<b>4 analog channels only</b>	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
<b>2 analog channels only</b>	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

**Query Syntax** :TRIGger:SEQuence:PATtern{1 | 2}?

The :TRIGger:SEQuence:PATtern<n>? query returns the current settings of the specified pattern resource.

**Return Format** <value>,<mask><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:FIND"](#) on page 484
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488
  - [":TRIGger:SEQuence:RESet"](#) on page 486

**:TRIGger:SEQuence:RESet**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SEQuence:RESet <value>

```
<value> ::= {NONE | PATtern1,ENTered | PATtern1,EXITed | EDGE1
            | PATtern1,AND,EDGE1 | PATtern2,ENTered | PATtern2,EXITed
            | EDGE2 | TIMer}
```

Values used in find and trigger stages are not available. EDGE2 is not available if EDGE2,COUNT is used in trigger stage.

The :TRIGger:SEQuence:RESet command specifies the reset stage of a sequence trigger. In multi-level trigger specifications, you may find a pattern, then search for another in sequence, but reset the entire search to the beginning if another condition occurs. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE1,NONE,NONE").

PATtern1 and PATtern2 are specified with the :TRIGger:SEQuence:PATtern command. EDGE1 and EDGE2 are specified with the :TRIGger:SEQuence:EDGE command. TIMer is specified with the :TRIGger:SEQuence:TIMer command.

**Query Syntax** :TRIGger:SEQuence:RESet?

The :TRIGger:SEQuence:RESet? query returns the reset stage specification for a sequence trigger. NONE is returned for unused parameters.

**Return Format** <reset\_value><NL>

```
<reset_value> ::= {NONE,NONE,NONE | PATt1,ENT,NONE | PATt1,EXIT,NONE
                  | EDGE1,NONE,NONE | PATt1,AND,EDGE1 | PATt2,ENT,NONE
                  | PATt2,EXIT,NONE | EDGE2,NONE,NONE | TIM,NONE,NONE}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:PATtern"](#) on page 485
  - [":TRIGger:SEQuence:EDGE"](#) on page 483
  - [":TRIGger:SEQuence:TIMer"](#) on page 487
  - [":TRIGger:SEQuence:FIND"](#) on page 484
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488

**:TRIGger:SEQuence:TIMer**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SEQuence:TIMer <time\_value>

<time\_value> ::= time in seconds in NR1 format

The :TRIGger:SEQuence:TIMer command sets the sequencer timer resource in seconds from 100 ns to 10 s. The timer is used in the reset stage to determine how long to wait for the trigger to occur before restarting.

**Query Syntax** :TRIGger:SEQuence:TIMer?

The :TRIGger:SEQuence:TIMer? query returns current sequencer timer setting.

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

<time\_value> ::= time in seconds in NR1 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:RESet"](#) on page 486

**:TRIGger:SEQuence:TRIGger**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SEQuence:TRIGger <value>

```
<value> ::= {PATTern2,ENTered | PATTern2,EXITed | EDGE2
            | PATTern2,AND,EDGE2 | EDGE2,COUNT | EDGE2,COUNT,NREFind}
```

The :TRIGger:SEQuence:TRIGger command specifies the trigger stage of a sequence trigger. The sequence commands set various search terms. After all of these are found in sequence, the trigger condition itself is searched for. This command accepts three program data parameters; you can use NONE to fill out the parameter list (for example, "EDGE2,NONE,NONE").

PATTern2 is specified with the :TRIGger:SEQuence:PATTern command. EDGE2 is specified with the :TRIGger:SEQuence:EDGE command. COUNT is specified with the :TRIGger:SEQuence:COUNt command.

**Query Syntax** :TRIGger:SEQuence:TRIGger?

The :TRIGger:SEQuence:TRIGger? query returns the trigger stage specification for a sequence trigger. NONE is returned for unused parameters.

**Return Format** <trigger\_value><NL>

```
<trigger_value> ::= {PATT2,ENT,NONE | PATT2,EXIT,NONE
                    | EDGE2,NONE,NONE | PATT2,AND,EDGE2
                    | EDGE2,COUN,NONE | EDGE2,COUN,NREF}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SEQuence:PATTern"](#) on page 485
  - [":TRIGger:SEQuence:EDGE"](#) on page 483
  - [":TRIGger:SEQuence:COUNt"](#) on page 482
  - [":TRIGger:SEQuence:FIND"](#) on page 484
  - [":TRIGger:SEQuence:RESet"](#) on page 486
  - [":TRIGger:SEQuence:RESet"](#) on page 486



## :TRIGger:SPI Commands

Table 77 :TRIGger:SPI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCK:SL OPe <slope> (see page 490)	:TRIGger:SPI:CLOCK:SL OPe? (see page 490)	<slope> ::= {NEGative   POSitive}
:TRIGger:SPI:CLOCK:TI Meout <time_value> (see page 491)	:TRIGger:SPI:CLOCK:TI Meout? (see page 491)	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 492)	:TRIGger:SPI:FRAMing? (see page 492)	<value> ::= {CHIPselect   NOTChipselect   TIMEout}
:TRIGger:SPI:PATtern: DATA <value>, <mask> (see page 493)	:TRIGger:SPI:PATtern: DATA? (see page 493)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn timer" where n ::= {0,...,9   A,...,F}
:TRIGger:SPI:PATtern: WIDTh <width> (see page 494)	:TRIGger:SPI:PATtern: WIDTh? (see page 494)	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:C LOCK <source> (see page 495)	:TRIGger:SPI:SOURce:C LOCK? (see page 495)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:D ATA <source> (see page 496)	:TRIGger:SPI:SOURce:D ATA? (see page 496)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:F RAME <source> (see page 497)	:TRIGger:SPI:SOURce:F RAME? (see page 497)	<value> ::= {CHANnel<n>   EXTErnal} for the DSO models <value> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format

**:TRIGger:SPI:CLOCK:SLOPe**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:CLOCK:SLOPe <slope>  
 <slope> ::= {NEGative | POSitive}

The :TRIGger:SPI:CLOCK:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

**Query Syntax** :TRIGger:SPI:CLOCK:SLOPe?

The :TRIGger:SPI:CLOCK:SLOPe? query returns the current SPI clock source slope.

**Return Format** <slope><NL>  
 <slope> ::= {NEG | POS}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
  - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495

**:TRIGger:SPI:CLOCK:TIMEout**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:CLOCK:TIMEout <time\_value>

<time\_value> ::= time in seconds in NR1 format

The :TRIGger:SPI:CLOCK:TIMEout command sets the SPI signal clock timeout resource in seconds from 500 ns to 10 s when the :TRIGger:SPI:FRAMing command is set to TIMEout. The timer is used to frame a signal by a clock timeout.

**Query Syntax** :TRIGger:SPI:CLOCK:TIMEout?

The :TRIGger:SPI:CLOCK:TIMEout? query returns current SPI clock timeout setting.

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

<time\_value> ::= time in seconds in NR1 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 490
  - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
  - [":TRIGger:SPI:FRAMing"](#) on page 492

**:TRIGger:SPI:FRAMing**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:FRAMing <value>

<value> ::= {CHIPselect | NOTChipselect | TIMEout}

The :TRIGger:SPI:FRAMing command sets the SPI trigger framing value. If TIMEout is selected, the timeout value is set by the :TRIGger:SPI:CLOCK:TIMEout command.

**Query Syntax** :TRIGger:SPI:FRAMing?

The :TRIGger:SPI:FRAMing? query returns the current SPI framing value.

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

<value> ::= {CHIPselect | NOTChipselect | TIMEout}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
  - [":TRIGger:SPI:SOURce:FRAME"](#) on page 497

**:TRIGger:SPI:PAtern:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:PAtern:DATA <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xn timer" where n ::= {0,...,9 | A,...,F}

The :TRIGger:SPI:PAtern:DATA command defines the SPI data pattern resource according to the value and the mask. This pattern, along with the data width, control the data pattern searched for in the data stream.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

**Query Syntax** :TRIGger:SPI:PAtern:DATA?

The :TRIGger:SPI:PAtern:DATA? query returns the current settings of the specified SPI data pattern resource.

**Return Format** <value>,<mask><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:PAtern:WIDTh"](#) on page 494
  - [":TRIGger:SPI:SOURce:DATA"](#) on page 496

## **:TRIGger:SPI:PAThern:WIDTh**

**N** (see [page 676](#))

**Command Syntax**    :TRIGger:SPI:PAThern:WIDTh <width>  
                         <width> ::= integer from 4 to 32 in NR1 format

The :TRIGger:SPI:PAThern:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 32 bits.

**Query Syntax**       :TRIGger:SPI:PAThern:WIDTh?

The :TRIGger:SPI:PAThern:WIDTh? query returns the current SPI data pattern width setting.

**Return Format**       <width><NL>  
                         <width> ::= integer from 4 to 32 in NR1 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:PAThern:DATA"](#) on page 493
  - [":TRIGger:SPI:SOURce:DATA"](#) on page 496

**:TRIGger:SPI:SOURce:CLOCK**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:SOURce:CLOCK <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:SPI:SOURce:CLOCK command sets the source for the SPI serial clock.

**Query Syntax** :TRIGger:SPI:SOURce:CLOCK?

The :TRIGger:SPI:SOURce:CLOCK? query returns the current source for the SPI serial clock.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 490
  - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
  - [":TRIGger:SPI:SOURce:FRAME"](#) on page 497
  - [":TRIGger:SPI:SOURce:DATA"](#) on page 496

**:TRIGger:SPI:SOURce:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:SOURce:DATA <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:SPI:SOURce:DATA command sets the source for the SPI serial data.

**Query Syntax** :TRIGger:SPI:SOURce:DATA?

The :TRIGger:SPI:SOURce:DATA? query returns the current source for the SPI serial data.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
  - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 497
  - [":TRIGger:SPI:PATtern:DATA"](#) on page 493
  - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 494



**:TRIGger:SPI:SOURce:FRAME**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:SPI:SOURce:FRAME <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:SPI:SOURce:FRAME command sets the frame source when :TRIGger:SPI:FRAMing is set to CHIPselect or NOTChipselect.

**Query Syntax** :TRIGger:SPI:SOURce:FRAME?

The :TRIGger:SPI:SOURce:FRAME? query returns the current frame source for the SPI serial frame.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
  - [":TRIGger:SPI:SOURce:DATA"](#) on page 496
  - [":TRIGger:SPI:FRAMing"](#) on page 492

**:TRIGger:TV Commands****Table 78** :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see <a href="#">page 499</a> )	:TRIGger:TV:LINE? (see <a href="#">page 499</a> )	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see <a href="#">page 500</a> )	:TRIGger:TV:MODE? (see <a href="#">page 500</a> )	<tv mode> ::= {FIELD1   FIELD2   AFIELDS   ALINES   LINE   VERTICAL   LFIELD1   LFIELD2   LALTERNATE   LVERTICAL}
:TRIGger:TV:POLarity <polarity> (see <a href="#">page 501</a> )	:TRIGger:TV:POLarity? (see <a href="#">page 501</a> )	<polarity> ::= {POSITIVE   NEGATIVE}
:TRIGger:TV:SOURce <source> (see <a href="#">page 502</a> )	:TRIGger:TV:SOURce? (see <a href="#">page 502</a> )	<source> ::= {CHANNEL<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANDARD <standard> (see <a href="#">page 503</a> )	:TRIGger:TV:STANDARD? (see <a href="#">page 503</a> )	<standard> ::= {GENERIC   NTSC   PALM   PAL   SECAM   {P480L60HZ   P480}   {P720L60HZ   P720}   {P1080L24HZ   P1080}   P1080L25HZ   {I1080L50HZ   I1080}   I1080L60HZ}

**:TRIGger:TV:LINE**

**N** (see [page 676](#))

**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 79** TV Trigger Line Number Limits

TV Standard	Mode				
	LINE	LField1	LField2	LALternate	VERTical
NTSC		1 to 263	1 to 262	1 to 262	
PAL		1 to 313	314 to 625	1 to 312	
PAL-M		1 to 263	264 to 525	1 to 262	
SECAM		1 to 313	314 to 625	1 to 312	
GENERIC		1 to 1024	1 to 1024		1 to 1024
P480L60HZ	1 to 525				
P720L60HZ	1 to 750				
P1080L24HZ	1 to 1125				
P1080L25HZ	1 to 1125				
I1080L50HZ	1 to 1125				
I1080L60HZ	1 to 1125				

**Query Syntax** :TRIGger:TV:LINE?

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

**Return Format** <line\_number><NL>

<line\_number> ::= integer in NR1 format

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 405
  - "[:TRIGger:TV:STANdard](#)" on page 503
  - "[:TRIGger:TV:MODE](#)" on page 500

**:TRIGger:TV:MODE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:TV:MODE <mode>

```
<mode> ::= {FIEld1 | FIEld2 | AFIElds | ALINes | LINE | VERTical
            | LFIeld1 | LFIeld2 | LALTernate | LVERTical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERTical parameter is only available when :TRIGger:TV:STANdard is GENERIC. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENERIC.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIElds	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERTical	LINEVert

**Query Syntax** :TRIGger:TV:MODE?

The :TRIGger:TV:MODE? query returns the TV trigger mode.

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

```
<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
            | LALT | LVER}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:TV:STANdard"](#) on page 503
  - [":TRIGger:MODE"](#) on page 411

**:TRIGger:TV:POLarity**

**N** (see [page 676](#))

**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 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:TV:SOURce"](#) on page 502

## :TRIGger:TV:SOURce

**N** (see [page 676](#))

**Command Syntax** :TRIGger:TV:SOURce <source>

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

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

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

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

**Query Syntax** :TRIGger:TV:SOURce?

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

**Return Format** <source><NL>

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:TV:POLarity"](#) on page 501

**Example Code**

- ["Example Code"](#) on page 442

**:TRIGger:TV:STANdard**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:TV:STANdard <standard>

```
<standard> ::= {GENeric | NTSC | PALM | PAL | SECam
                | {P480L60HZ | P480} | {P720L60HZ | P720}
                | {P1080L24HZ | P1080} | P1080L25HZ
                | {I1080L50HZ | I1080} | I1080L60HZ}
```

The :TRIGger:TV:STANdard command selects the video standard. GENeric mode is non-interlaced.

**Query Syntax** :TRIGger:TV:STANdard?

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

**Return Format** <standard><NL>

```
<standard> ::= {GEN | NTSC | PALM | PAL | SEC | P480L60HZ | P760L60HZ
                | P1080L24HZ | P1080L25HZ | I1080L50HZ | I1080L60HZ}
```

**:TRIGger:UART Commands****Table 80** :TRIGger:UART Commands Summary

Command	Query	Options and Query Returns
:TRIGger:UART:BAUDRate <baudrate> (see <a href="#">page 506</a> )	:TRIGger:UART:BAUDRate? (see <a href="#">page 506</a> )	<baudrate> ::= {1200   1800   2000   2400   3600   4800   7200   9600   14400   15200   19200   28800   38400   56000   57600   76800   115200   128000   230400   460800   921600   1382400   1843200   2764800}
:TRIGger:UART:BITOrder <bitorder> (see <a href="#">page 507</a> )	:TRIGger:UART:BITOrder? (see <a href="#">page 507</a> )	<bitorder> ::= {LSBFirst   MSBFirst}
:TRIGger:UART:BURSt <value> (see <a href="#">page 508</a> )	:TRIGger:UART:BURSt? (see <a href="#">page 508</a> )	<value> ::= {OFF   1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see <a href="#">page 509</a> )	:TRIGger:UART:DATA? (see <a href="#">page 509</a> )	<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,...,9   A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0   1} for binary
:TRIGger:UART:IDLE <time_value> (see <a href="#">page 510</a> )	:TRIGger:UART:IDLE? (see <a href="#">page 510</a> )	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see <a href="#">page 511</a> )	:TRIGger:UART:PARity? (see <a href="#">page 511</a> )	<parity> ::= {EVEN   ODD   NONE}
:TRIGger:UART:POLarity <polarity> (see <a href="#">page 512</a> )	:TRIGger:UART:POLarity? (see <a href="#">page 512</a> )	<polarity> ::= {HIGH   LOW}
:TRIGger:UART:QUALifier <value> (see <a href="#">page 513</a> )	:TRIGger:UART:QUALifier? (see <a href="#">page 513</a> )	<value> ::= {EQUAL   NOTequal   GREaterthan   LESSthan}
:TRIGger:UART:SOURce:RX <source> (see <a href="#">page 514</a> )	:TRIGger:UART:SOURce:RX? (see <a href="#">page 514</a> )	<source> ::= {CHANnel<n>   EXTERNAL} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format



**Table 80** :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:SOURce: TX <source> (see page 515)	:TRIGger:UART:SOURce: TX? (see page 515)	<source> ::= {CHANnel<n>   EXTErnal} for DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see page 516)	:TRIGger:UART:TYPE? (see page 516)	<value> ::= {RSTArt   RSTOp   RDATA   RD1   RD0   RDX   PARityerror   TSTArt   TSTOp   TDATA   TD1   TD0   TDX}
:TRIGger:UART:WIDTh <width> (see page 517)	:TRIGger:UART:WIDTh? (see page 517)	<width> ::= {5   6   7   8   9}

**:TRIGger:UART:BAUDrate**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600  
| 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600  
| 76800 | 115200 | 128000 | 230400 | 460800 | 921600  
| 1382400 | 1843200 | 2764800}

The :TRIGger:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode.

If the baud rate you select does not match the system baud rate, false triggers may occur.

**Query Syntax** :TRIGger:UART:BAUDrate?

The :TRIGger:UART:BAUDrate? query returns the current UART baud rate setting.

**Return Format** <baudrate><NL>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600  
| 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600  
| 76800 | 115200 | 128000 | 230400 | 460800 | 921600  
| 1382400 | 1843200 | 2764800}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516

**:TRIGger:UART:BITorder**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:BITorder <bitorder>  
 <bitorder> ::= {LSBFirst | MSBFirst}

The :TRIGger:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

**Query Syntax** :TRIGger:UART:BITorder?

The :TRIGger:UART:BITorder? query returns the current UART bit order setting.

**Return Format** <bitorder><NL>  
 <bitorder> ::= {LSBF | MSBF}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516
  - [":TRIGger:UART:SOURce:RX"](#) on page 514
  - [":TRIGger:UART:SOURce:TX"](#) on page 515

**:TRIGger:UART:BURSt**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:BURSt <value>  
 <value> ::= {OFF | 1 to 4096 in NR1 format}

The :TRIGger:UART:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

**Query Syntax** :TRIGger:UART:BURSt?

The :TRIGger:UART:BURSt? query returns the current UART trigger burst value.

**Return Format** <value><NL>  
 <value> ::= {OFF | 1 to 4096 in NR1 format}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:IDLE"](#) on page 510
  - [":TRIGger:UART:TYPE"](#) on page 516

**:TRIGger:UART:DATA**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:DATA <value>

<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255  
(0x00-0xff)

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

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

The :TRIGger:UART:DATA command selects the data byte value (0x00 to 0xFF) for the trigger QUALifier when in UART mode. The data value is used when one of the RD or TD trigger types is selected.

**Query Syntax** :TRIGger:UART:DATA?

The :TRIGger:UART:DATA? query returns the current UART trigger data value.

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

<value> ::= 8-bit integer in decimal from 0-255

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516

## :TRIGger:UART:IDLE

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:IDLE <time\_value>

<time\_value> ::= time from 1 us to 10 s in NR3 format

The :TRIGger:UART:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

**Query Syntax** :TRIGger:UART:IDLE?

The :TRIGger:UART:IDLE? query returns the current UART trigger idle period time.

**Return Format** <time\_value><NL>

<time\_value> ::= time from 1 us to 10 s in NR3 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:BURSt"](#) on page 508
  - [":TRIGger:UART:TYPE"](#) on page 516

**:TRIGger:UART:PARity**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:PARity <parity>  
 <parity> ::= {EVEN | ODD | NONE}

The :TRIGger:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

**Query Syntax** :TRIGger:UART:PARity?

The :TRIGger:UART:PARity? query returns the current UART parity setting.

**Return Format** <parity><NL>  
 <parity> ::= {EVEN | ODD | NONE}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516

## :TRIGger:UART:POLarity

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:POLarity <polarity>  
<polarity> ::= {HIGH | LOW}

The :TRIGger:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

**Query Syntax** :TRIGger:UART:POLarity?

The :TRIGger:UART:POLarity? query returns the current UART polarity setting.

**Return Format** <polarity><NL>  
<polarity> ::= {HIGH | LOW}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516



**:TRIGger:UART:QUALifier**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:QUALifier <value>

<value> ::= {EQUAL | NOTequal | GREaterthan | LESSthan}

The :TRIGger:UART:QUALifier command selects the data qualifier when :TYPE is set to RDATA, RD1, RD0, RDX, TDATA, TD1, TD0, or TDX for the trigger when in UART mode.

**Query Syntax** :TRIGger:UART:QUALifier?

The :TRIGger:UART:QUALifier? query returns the current UART trigger qualifier.

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

<value> ::= {EQU | NOT | GRE | LESS}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516

**:TRIGger:UART:SOURce:RX**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:SOURce:RX <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

**Query Syntax** :TRIGger:UART:SOURce:RX?

The :TRIGger:UART:SOURce:RX? query returns the current source for the UART Rx signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516
  - [":TRIGger:UART:BITorder"](#) on page 507

**:TRIGger:UART:SOURce:TX**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:SOURce:TX <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

**Query Syntax** :TRIGger:UART:SOURce:TX?

The :TRIGger:UART:SOURce:TX? query returns the current source for the UART Tx signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516
  - [":TRIGger:UART:BITorder"](#) on page 507

**:TRIGger:UART:TYPE**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:TYPE <value>

<value> ::= {RSTArt | RSTOp | RDATA | RD1 | RD0 | RDX | PARityerror  
| TSTArt | TSTOp | TDATA | TD1 | TD0 | TDX}

The :TRIGger:UART:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :TRIGger:UART:DATA and :TRIGger:UART:QUALifier commands are used to specify the data value and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and alert bit values) are only valid when a 9-bit width has been selected.

**Query Syntax** :TRIGger:UART:TYPE?

The :TRIGger:UART:TYPE? query returns the current UART trigger data value.

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

<value> ::= {RSTA | RSTO | RDATA | RD1 | RD0 | RDX | PAR | TSTA |  
TSTO | TDATA | TD1 | TD0 | TDX}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:DATA"](#) on page 509
  - [":TRIGger:UART:QUALifier"](#) on page 513
  - [":TRIGger:UART:WIDTH"](#) on page 517

**:TRIGger:UART:WIDTH**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:UART:WIDTH <width>  
 <width> ::= {5 | 6 | 7 | 8 | 9}

The :TRIGger:UART:WIDTH command determines the number of bits (5-9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

**Query Syntax** :TRIGger:UART:WIDTH?

The :TRIGger:UART:WIDTH? query returns the current UART width setting.

**Return Format** <width><NL>  
 <width> ::= {5 | 6 | 7 | 8 | 9}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:UART:TYPE"](#) on page 516

**:TRIGger:USB Commands****Table 81** :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:D MINus <source> (see <a href="#">page 519</a> )	:TRIGger:USB:SOURce:D MINus? (see <a href="#">page 519</a> )	<source> ::= {CHANnel<n>   EXtErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SOURce:D PLus <source> (see <a href="#">page 520</a> )	:TRIGger:USB:SOURce:D PLus? (see <a href="#">page 520</a> )	<source> ::= {CHANnel<n>   EXtErnal} for the DSO models <source> ::= {CHANnel<n>   DIGital0,...,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:USB:SPEEd <value> (see <a href="#">page 521</a> )	:TRIGger:USB:SPEEd? (see <a href="#">page 521</a> )	<value> ::= {LOW   FULL}
:TRIGger:USB:TRIGger <value> (see <a href="#">page 522</a> )	:TRIGger:USB:TRIGger? (see <a href="#">page 522</a> )	<value> ::= {SOP   EOP   ENTersuspend   EXITsuspend   RESet}

**:TRIGger:USB:SOURce:DMINus**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:USB:SOURce:DMINus <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:USB:SOURce:DMINus command sets the source for the USB D- signal.

**Query Syntax** :TRIGger:USB:SOURce:DMINus?

The :TRIGger:USB:SOURce:DMINus? query returns the current source for the USB D- signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:USB:SOURce:DPLus"](#) on page 520
  - [":TRIGger:USB:TRIGger"](#) on page 522

**:TRIGger:USB:SOURce:DPLus**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:USB:SOURce:DPLus <source>

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

<source> ::= {CHANnel<n> | DIGital0,...,DIGital15} for the MSO models

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

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

The :TRIGger:USB:SOURce:DPLus command sets the source for the USB D+ signal.

**Query Syntax** :TRIGger:USB:SOURce:DPLus?

The :TRIGger:USB:SOURce:DPLus? query returns the current source for the USB D+ signal.

**Return Format** <source><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:USB:SOURce:DMINus"](#) on page 519
  - [":TRIGger:USB:TRIGger"](#) on page 522



**:TRIGger:USB:SPEEd**

**N** (see [page 676](#))

**Command Syntax** :TRIGger:USB:SPEEd <value>  
 <value> ::= {LOW | FULL}

The :TRIGger:USB:SPEEd command sets the expected USB signal speed to be Low Speed (1.5 Mb/s) or Full Speed (12 Mb/s).

**Query Syntax** :TRIGger:USB:SPEEd?

The :TRIGger:USB:SPEEd? query returns the current speed value for the USB signal.

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

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:USB:SOURce:DMINus"](#) on page 519
  - [":TRIGger:USB:SOURce:DPLus"](#) on page 520
  - [":TRIGger:USB:TRIGger"](#) on page 522

## :TRIGger:USB:TRIGger

**N** (see [page 676](#))

**Command Syntax** :TRIGger:USB:TRIGger <value>

<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}

The :TRIGger:USB:TRIGger command sets where the USB trigger will occur:

- SOP — Start of packet.
- EOP — End of packet.
- ENTersuspend — Enter suspend state.
- EXITsuspend — Exit suspend state.
- RESet — Reset complete.

**Query Syntax** :TRIGger:USB:TRIGger?

The :TRIGger:USB:TRIGger? query returns the current USB trigger value.

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

<value> ::= {SOP | EOP | ENTersuspend | EXITsuspend | RESet}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:USB:SPEed"](#) on page 521

## :WAVEform Commands

Provide access to waveform data. See "[Introduction to :WAVEform Commands](#)" on page 525.

**Table 82** :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see <a href="#">page 531</a> )	:WAVEform:BYTeorder? (see <a href="#">page 531</a> )	<value> ::= {LSBFirst   MSBFirst}
n/a	:WAVEform:COUNt? (see <a href="#">page 532</a> )	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see <a href="#">page 533</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 535</a> )	:WAVEform:FORMat? (see <a href="#">page 535</a> )	<value> ::= {WORD   BYTE   ASCII}
:WAVEform:POINTs <# points> (see <a href="#">page 536</a> )	:WAVEform:POINTs? (see <a href="#">page 536</a> )	<# points> ::= {100   250   500   1000   <points_mode>} if waveform points mode is NORMAl <# points> ::= {100   250   500   1000   2000 ... 8000000 in 1-2-5 sequence   <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl   MAXimum   RAW}
:WAVEform:POINTs:MODE <points_mode> (see <a href="#">page 538</a> )	:WAVEform:POINTs:MODE ? (see <a href="#">page 539</a> )	<points_mode> ::= {NORMAl   MAXimum   RAW}

Table 82 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVEform:PREamble? (see <a href="#">page 540</a> )	<pre>&lt;preamble_block&gt; ::= &lt;format NR1&gt;, &lt;type NR1&gt;,&lt;points NR1&gt;,&lt;count NR1&gt;, &lt;xincrement NR3&gt;, &lt;xorigin NR3&gt;, &lt;xreference NR1&gt;,&lt;yincrement NR3&gt;, &lt;yorigin NR3&gt;, &lt;yreference NR1&gt; &lt;format&gt; ::= an integer in NR1 format:  • 0 for BYTE format • 1 for WORD format • 2 for ASCII format &lt;type&gt; ::= an integer in NR1 format:  • 0 for NORMAl type • 1 for PEAK detect type • 2 for AVERage type • 3 for HRESolution type &lt;count&gt; ::= Average count, or 1 if PEAK detect type or NORMAl; an integer in NR1 format</pre>
n/a	:WAVEform:SEGmented:C OUNT? (see <a href="#">page 543</a> )	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:T TAG? (see <a href="#">page 544</a> )	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see <a href="#">page 545</a> )	:WAVEform:SOURce? (see <a href="#">page 545</a> )	<pre>&lt;source&gt; ::= {CHANnel&lt;n&gt;   FUNCTION   MATH   SBUS} for DSO models &lt;source&gt; ::= {CHANnel&lt;n&gt;   POD{1   2}   BUS{1   2}   FUNCTION   MATH   SBUS} for MSO models &lt;n&gt; ::= 1-2 or 1-4 in NR1 format</pre>
:WAVEform:SOURce:SUBS ource <subsource> (see <a href="#">page 549</a> )	:WAVEform:SOURce:SUBS ource? (see <a href="#">page 549</a> )	<subsource> ::= {{NONE   RX}   TX}
n/a	:WAVEform:TYPE? (see <a href="#">page 550</a> )	<return_mode> ::= {NORM   PEAK   AVER   HRES}
:WAVEform:UNSigned {{0   OFF}   {1   ON}} (see <a href="#">page 551</a> )	:WAVEform:UNSigned? (see <a href="#">page 551</a> )	{0   1}
:WAVEform:VIEW <view> (see <a href="#">page 552</a> )	:WAVEform:VIEW? (see <a href="#">page 552</a> )	<view> ::= {MAIN}

**Table 82** :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:XINCrement? (see <a href="#">page 553</a> )	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see <a href="#">page 554</a> )	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see <a href="#">page 555</a> )	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see <a href="#">page 556</a> )	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see <a href="#">page 557</a> )	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see <a href="#">page 558</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 533](#)) and :WAVeform:PREamble (see [page 540](#)). The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

### Data Acquisition Types

There are three types of waveform acquisitions that can be selected for analog channels with the :ACquire:TYPE command (see [page 185](#)): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always

acquired using NORMal. When the data is acquired using the :DIGitize command (see [page 145](#)) or :RUN command (see [page 165](#)), 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 533](#)) 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 179](#)).

### Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVEform:POINts command (see [page 536](#)). 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 536](#)). 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 176](#)). 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 536](#)). 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 536](#)). 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 185](#)), the value returned by the :WAVEform:XINCrement query (see [page 553](#)) 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 535](#)), 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 185](#)), 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 535](#)). 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 551](#)) 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 535](#)) 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  $9.9\text{e}+37$ . The setting of `:WAVEform:BYTEorder` (see [page 531](#)) and `:WAVEform:UNSIGNED` (see [page 551](#)) have no effect when the format is ASCII.

#### Data Format for Transfer - WORD format

WORD format (see `:WAVEform:FORMat` on [page 535](#)) 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 536](#)). 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 531](#)) 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 535](#)) 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 531](#)) 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 551](#)) 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 535](#)) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see [page 531](#)) 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 187](#)) 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**(see [page 676](#))

**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 525
  - [":WAVeform:DATA"](#) on page 533
  - [":WAVeform:FORMat"](#) on page 535
  - [":WAVeform:PREamble"](#) on page 540

- Example Code**
- ["Example Code"](#) on page 546
  - ["Example Code"](#) on page 541

**:WAVeform:COUNT****C** (see [page 676](#))**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 525
- [":ACQuire:COUNT"](#) on page 176
- [":ACQuire:TYPE"](#) on page 185

**:WAVeform:DATA**

**C** (see [page 676](#))

**Query Syntax** :WAVeform:DATA?

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSIGNED, :WAVeform:BYTEorder, :WAVeform:FORMat, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINts command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 – Hole. Holes are locations where data has not yet been acquired. Holes can be reasonably expected in the equivalent time acquisition mode (especially at slower horizontal sweep speeds when measuring low frequency signals).

Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.

- 0x01 or 0x0001 – Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFFFF – Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

**Return Format** <binary block data><NL>

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":WAVeform:UNSIGNED"](#) on page 551
  - [":WAVeform:BYTEorder"](#) on page 531
  - [":WAVeform:FORMat"](#) on page 535
  - [":WAVeform:POINts"](#) on page 536
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:SOURce"](#) on page 545
  - [":WAVeform:TYPE"](#) on page 550

**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

```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:WAVeform:FORMat**

**C** (see [page 676](#))

**Command Syntax** :WAVeform:FORMat <value>  
 <value> ::= {WORD | BYTE | ASCii}

The :WAVeform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

ASCII formatted data is transferred ASCII text.

- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

When the :WAVeform:SOURce is one of the digital channel buses (BUS1 or BUS2), ASCii and WORD are the only waveform formats allowed.

**Query Syntax** :WAVeform:FORMat?

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

**Return Format** <value><NL>  
 <value> ::= {WORD | BYTE | ASC}

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":WAVeform:BYTeorder"](#) on page 531
  - [":WAVeform:SOURce"](#) on page 545
  - [":WAVeform:DATA"](#) on page 533
  - [":WAVeform:PREamble"](#) on page 540

**Example Code** • ["Example Code"](#) on page 546

**:WAVeform:POINts****C** (see [page 676](#))

**Command Syntax** :WAVeform:POINts <# points>

```
<# points> ::= {100 | 250 | 500 | 1000 | <points mode>}
              if waveform points mode is NORMAl

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000
              in 1-2-5 sequence | <points mode>}
              if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMAl | MAXimum | RAW}
```

**NOTE**

The <points\_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog 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 and may be up to 8,000,000. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog or digital sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

See the :WAVeform:POINts:MODE command (see [page 538](#)) for more information on the <points\_mode> option.

Only data visible on the display will be returned.

The maximum number of points returned when the waveform source is math or function is 1000.

When the :WAVeform:SOURce is the serial decode bus (SBUS), this command is ignored, and all available serial decode bus data is returned.

**Query Syntax** :WAVeform:POINts?

The :WAVeform:POINts query returns the number of waveform points to be transferred when using the :WAVeform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVeform:POINts:MODE command (see [page 538](#)) for more information).



When the :WAVEform:SOURce is the serial decode bus (SBUS), this query returns the number of messages that were decoded.

**Return Format**

```
<# points><NL>

<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
              if waveform points mode is NORMAl

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000
              in 1-2-5 sequence | <maximum # points>}
              if waveform points mode is MAXimum or RAW
```

**NOTE**

If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.

**See Also**

- ["Introduction to :WAVEform Commands"](#) on page 525
- [":ACQuire:POINts"](#) on page 179
- [":WAVEform:DATA"](#) on page 533
- [":WAVEform:SOURce"](#) on page 545
- [":WAVEform:VIEW"](#) on page 552
- [":WAVEform:PREAmble"](#) on page 540
- [":WAVEform:POINts:MODE"](#) on page 538

**Example Code**

```
' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:WAVeform:POINts:MODE**

**N** (see [page 676](#))

**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 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

If the <points\_mode> is NORMal, the measurement record is retrieved.

If the <points\_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points\_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), or if the reconstruction filter (Sin(x)/x interpolation) is in use, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

**Considerations  
for MAXimum or  
RAW data  
retrieval**

- The instrument must be stopped (see the :STOP command (see [page 169](#)) or the :DIGitize command (see [page 145](#)) in the root subsystem) in order to return more than 1000 points.
- :TIMEbase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than 1000 points.
- MAXimum or RAW will allow up to 8,000,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVeform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

**Query Syntax**     `:WAVeform:POINts:MODE?`

The `:WAVeform:POINts:MODE?` query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

**Return Format**     `<points_mode><NL>`

`<points_mode> ::= {NORMal | MAXimum | RAW}`

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":ACQuire:POINts"](#) on page 179
  - [":WAVeform:DATA"](#) on page 533
  - [":WAVeform:VIEW"](#) on page 552
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:POINts"](#) on page 536
  - [":TIMEbase:MODE"](#) on page 395
  - [":ACQuire:TYPE"](#) on page 185
  - [":ACQuire:COUNt"](#) on page 176

**:WAVeform:PREamble**

**C** (see [page 676](#))

**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.

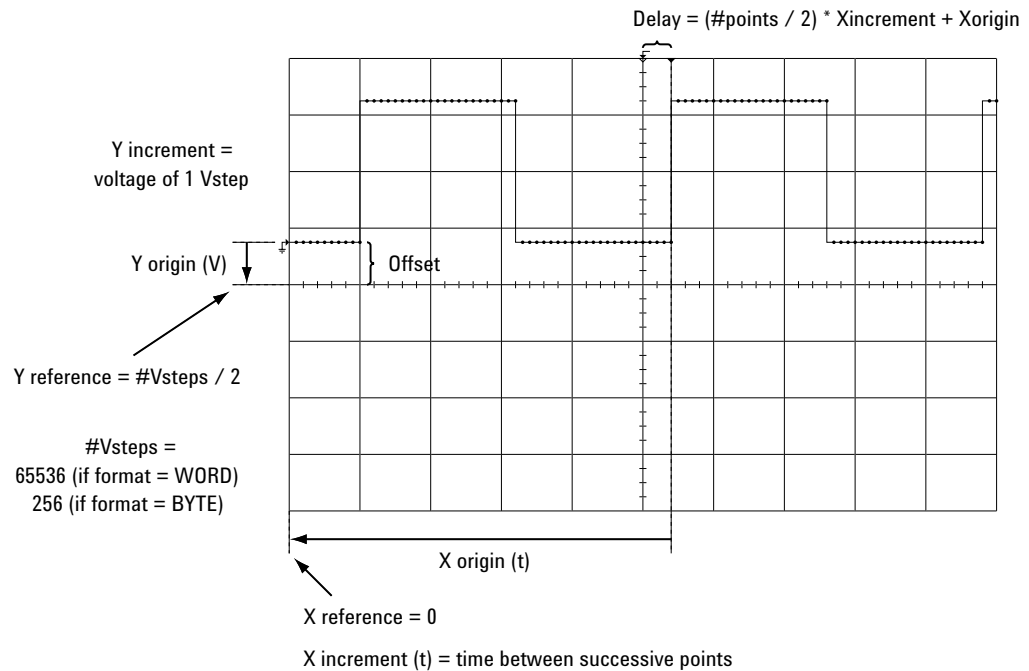
**Return Format** <preamble\_block><NL>

```
<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**
- [":WAVEform Commands"](#) on page 525
  - [":ACquire:COUNT"](#) on page 176
  - [":ACquire:POINTs"](#) on page 179
  - [":ACquire:TYPE"](#) on page 185
  - [":DIGitize"](#) on page 145
  - [":WAVEform:COUNT"](#) on page 532
  - [":WAVEform:DATA"](#) on page 533
  - [":WAVEform:FORMat"](#) on page 535
  - [":WAVEform:POINTs"](#) on page 536
  - [":WAVEform:TYPE"](#) on page 550
  - [":WAVEform:XINCrement"](#) on page 553
  - [":WAVEform:XORigin"](#) on page 554
  - [":WAVEform:XREFerence"](#) on page 555
  - [":WAVEform:YINCrement"](#) on page 556
  - [":WAVEform:YORigin"](#) on page 557
  - [":WAVEform:YREFerence"](#) on page 558

### 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)

```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:WAVeform:SEGMented:COUNT**

**N** (see [page 676](#))

**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.

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 command.

**Return Format** <count> ::= an integer from 2 to 2000 in NR1 format (count set by :ACQuire:SEGMented:COUNT).

- See Also**
- [":ACQuire:MODE"](#) on page 178
  - [":ACQuire:SEGMented:COUNT"](#) on page 181
  - [":DIGitize"](#) on page 145
  - ["Introduction to :WAVeform Commands"](#) on page 525

**Example Code** • ["Example Code"](#) on page 182

**:WAVeform:SEGmented:TTAG****N** (see [page 676](#))**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 182
  - ["Introduction to :WAVeform Commands"](#) on page 525

**Example Code**

- ["Example Code"](#) on page 182



**:WAVeform:SOURce**

**C** (see [page 676](#))

**Command Syntax** :WAVeform:SOURce <source>

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

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

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

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

The :WAVeform:SOURce command selects the analog channel, function, digital pod, digital bus, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply; integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

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 535](#)).

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 | SBUS} for DSO models

<source> ::= {CHAN<n> | POD{1 | 2} | BUS{1 | 2} | FUNC | SBUS}
            for MSO models

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

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

- See Also**
- ["Introduction to :WAVEform Commands" on page 525](#)
  - [":DIGitize" on page 145](#)
  - [":WAVEform:FORMat" on page 535](#)
  - [":WAVEform:BYTeorder" on page 531](#)
  - [":WAVEform:DATA" on page 533](#)
  - [":WAVEform:PREamble" on page 540](#)

### 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, 2 = ASCII.
'   TYPE            : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'   POINTS          : int32 - number of data points transferred.
'   COUNT           : int32 - 1 and is always 1.
'   XINCREMENT      : float64 - time difference between data points.
```

```

'   XORIGIN      : float64 - always the first data point in memory.
'   XREFERENCE   : int32 - specifies the data point associated with
'                   x-origin.
'   YINCREMENT   : float32 - voltage diff between data points.
'   YORIGIN      : float32 - value is the voltage at center screen.
'   YREFERENCE   : int32 - specifies the data point where y-origin
'                   occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'   FormatNumber(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

```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

**:WAVeform:SOURce:SUBSource**(see [page 676](#))**Command Syntax** :WAVeform:SOURce:SUBSource <subsource>

&lt;subsource&gt; ::= {{NONE | RX} | TX}

If the :WAVeform:SOURce is SBUS (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

Currently, only UART serial decode lets you get "TX" data. The default, NONE, specifies "RX" data. (RX is an alias for NONE.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS:MODE is not UART, the only valid subsource is NONE.

**Query Syntax** :WAVeform:SOURce:SUBSource?

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

**Return Format** <subsource><NL>

&lt;subsource&gt; ::= {NONE | TX}

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":WAVeform:SOURce"](#) on page 545

**:WAVeform:TYPE****C** (see [page 676](#))**Query Syntax**    :WAVeform:TYPE?

The :WAVeform:TYPE? query returns the acquisition mode associated with the currently selected waveform. The acquisition mode is set by the :ACQuire:TYPE command.

**Return Format**    <mode><NL>  
                     <mode> ::= {NORM | PEAK | AVER | HRES}

**NOTE**

If the :WAVeform:SOURce is POD1, POD2, or SBUS, the type is always NORM.

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":ACQuire:TYPE"](#) on page 185
  - [":WAVeform:DATA"](#) on page 533
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:SOURce"](#) on page 545

**:WAVeform:UNSigned**(see [page 676](#))

**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 or POD2, 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 525
  - [":WAVeform:SOURce"](#) on page 545

### :WAVeform:VIEW

**C** (see [page 676](#))

**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 525
  - [":WAVeform:POINTS"](#) on page 536



**:WAVeform:XINCrement****C** (see [page 676](#))**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 525
  - [":WAVeform:PREamble"](#) on page 540

- Example Code**
- ["Example Code"](#) on page 541

**:WAVeform:XORigin**

**C** (see [page 676](#))

**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 525
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:XREFerence"](#) on page 555

**Example Code** • ["Example Code"](#) on page 541

**:WAVeform:XREFerence**(see [page 676](#))**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>

&lt;value&gt; ::= x-reference value = 0 in 32-bit NR1 format

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 525
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:XORigin"](#) on page 554

- Example Code**
- ["Example Code"](#) on page 541

## :WAVeform:YINCrement

**C** (see [page 676](#))

**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 525
  - [":WAVeform:PREamble"](#) on page 540

- Example Code**
- ["Example Code"](#) on page 541

**:WAVeform:YORigin**(see [page 676](#))**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 525
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:YREference"](#) on page 558

- Example Code**
- ["Example Code"](#) on page 541

**:WAVeform:YREference****C** (see [page 676](#))**Query Syntax** :WAVeform:YREference?

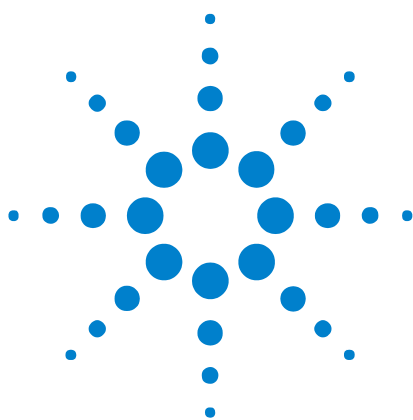
The :WAVeform:YREference? query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCII.

**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 525
  - [":WAVeform:PREamble"](#) on page 540
  - [":WAVeform:YORigin"](#) on page 557

- Example Code**
- ["Example Code"](#) on page 541



## 6 Commands A-Z

A	559
B	560
C	561
D	563
E	565
F	565
G	567
H	567
I	567
L	568
M	569
N	571
O	571
P	571
Q	573
R	573
S	574
T	578
U	583
V	583
W	584
X	585
Y	585

- A**
- AALias, [":ACQuire:AALias"](#) on page 174
  - ACKnowledge, [":TRIGger:CAN:ACKnowledge"](#) on page 629
  - [":ACQuire:AALias"](#) on page 174
  - [":ACQuire:COMplete"](#) on page 175
  - [":ACQuire:COUNt"](#) on page 176
  - [":ACQuire:DAALias"](#) on page 177



- ":ACQuire:MODE" on page 178
- ":ACQuire:POINts" on page 179
- ":ACQuire:RSIGnal" on page 180
- ":ACQuire:SEGmented:COUNt" on page 181
- ":ACQuire:SEGmented:INDex" on page 182
- ":ACQuire:SRATe" on page 184
- ":ACQuire:TYPE" on page 185
- ":ACTivity" on page 137
- ADDRes Commands:
  - ":SBUS:BUSDoctor:ADDRes" on page 360
  - ":TRIGger:IIC:PATtern:ADDRes" on page 465
- ":AER (Arm Event Register)" on page 138
- APrinter, ":HARDcopy:APrinter" on page 270
- AREA Commands:
  - ":HARDcopy:AREA" on page 269
  - ":SAVE:IMAGe:AREA" on page 347
- ASIZe, ":SBUS:IIC:ASIZe" on page 374
- ":AUToscale" on page 139
  - ":AUToscale:AMODE" on page 141
  - ":AUToscale:CHANnels" on page 142
- B**
  - BASE, ":SBUS:UART:BASE" on page 378
  - BAUDrate Commands:
    - ":SBUS:BUSDoctor:BAUDrate" on page 361
    - ":TRIGger:CAN:SIGNal:BAUDrate" on page 423
    - ":TRIGger:LIN:SIGNal:BAUDrate" on page 476
    - ":TRIGger:UART:BAUDrate" on page 506
  - BIT<m>, ":BUS<n>:BIT<m>" on page 189
  - BITorder, ":TRIGger:UART:BITorder" on page 507
  - BITS, ":BUS<n>:BITS" on page 190
  - ":BLANK" on page 143
  - BURSt, ":TRIGger:UART:BURSt" on page 508
  - ":BUS<n>:BIT<m>" on page 189
  - ":BUS<n>:BITS" on page 190
  - ":BUS<n>:CLEar" on page 192
  - ":BUS<n>:DISPlay" on page 193



- [":BUS<n>:LABel"](#) on page 194
  - [":BUS<n>:MASK"](#) on page 195
  - BUSDoctor Commands:
    - [":SBUS:BUSDoctor:ADDResS"](#) on page 360
    - [":SBUS:BUSDoctor:BAUDRate"](#) on page 361
    - [":SBUS:BUSDoctor:CHANnel"](#) on page 362
    - [":SBUS:BUSDoctor:MODE"](#) on page 363
  - BWLimit Commands:
    - [":CHANnel<n>:BWLimit"](#) on page 207
    - [":EXTeRnal:BWLimit"](#) on page 242
  - BYTeorder, [":WAVEform:BYTeorder"](#) on page 531
- C**
- [":CALibrate:DATE"](#) on page 197
  - [":CALibrate:LABel"](#) on page 198
  - [":CALibrate:STARt"](#) on page 199
  - [":CALibrate:STATus"](#) on page 200
  - [":CALibrate:SWITCh"](#) on page 201
  - [":CALibrate:TEMPerature"](#) on page 202
  - [":CALibrate:TIME"](#) on page 203
  - CAN Commands:
    - [":SBUS:CAN:COUNT:ERRor"](#) on page 364
    - [":SBUS:CAN:COUNT:OVERload"](#) on page 365
    - [":SBUS:CAN:COUNT:RESet"](#) on page 366
    - [":SBUS:CAN:COUNT:TOTal"](#) on page 367
    - [":SBUS:CAN:COUNT:UTILization"](#) on page 368
    - [":TRIGger:CAN Commands"](#) on page 416
  - CBASe, [":TRIGger:FLEXray:TIME:CBASe"](#) on page 450
  - CCBASe, [":TRIGger:FLEXray:FRAME:CCBase"](#) on page 446
  - CCRepetition, [":TRIGger:FLEXray:FRAME:CCRepetition"](#) on page 447
  - CRepetition, [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 451
  - [":CDISplay"](#) on page 144
  - CENTer, [":FUNCTION:CENTer"](#) on page 253
  - CHANnel, [":SBUS:BUSDoctor:CHANnel"](#) on page 362
  - [":CHANnel:ACTivity"](#) on page 592
  - [":CHANnel:LABel"](#) on page 593
  - [":CHANnel:THReshold"](#) on page 594

- [":CHANnel2:SKEW"](#) on page 595
- [":CHANnel<n>:BWLimit"](#) on page 207
- [":CHANnel<n>:COUPling"](#) on page 208
- [":CHANnel<n>:DISPlay"](#) on page 209
- [":CHANnel<n>:IMPedance"](#) on page 210
- [":CHANnel<n>:INPut"](#) on page 596
- [":CHANnel<n>:INVert"](#) on page 211
- [":CHANnel<n>:LABel"](#) on page 212
- [":CHANnel<n>:OFFSet"](#) on page 213
- [":CHANnel<n>:PMODE"](#) on page 597
- [":CHANnel<n>:PROBe"](#) on page 214
- [":CHANnel<n>:PROBe:ID"](#) on page 215
- [":CHANnel<n>:PROBe:SKEW"](#) on page 216
- [":CHANnel<n>:PROBe:STYPe"](#) on page 217
- [":CHANnel<n>:PROTection"](#) on page 218
- [":CHANnel<n>:RANGe"](#) on page 219
- [":CHANnel<n>:SCALe"](#) on page 220
- [":CHANnel<n>:UNITs"](#) on page 221
- [":CHANnel<n>:VERNier"](#) on page 222
- CLear Commands:
  - [":BUS<n>:CLEar"](#) on page 192
  - [":DISPlay:CLEar"](#) on page 232
  - [":MEASure:CLEar"](#) on page 295
- CLOck Commands:
  - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 468
  - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 490
  - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
  - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
- ["\\*CLS \(Clear Status\)"](#) on page 113
- COMplete, [":ACQuire:COMplete"](#) on page 175
- CONNect, [":DISPlay:CONNect"](#) on page 598
- COUNT Commands:
  - [":ACQuire:COUNt"](#) on page 176
  - [":ACQuire:SEGMENTed:COUNt"](#) on page 181
  - [":SBUS:CAN:COUNt:ERRor"](#) on page 364

- [":SBUS:CAN:COUNt:OVERload"](#) on page 365
- [":SBUS:CAN:COUNt:RESet"](#) on page 366
- [":SBUS:CAN:COUNt:TOTal"](#) on page 367
- [":SBUS:CAN:COUNt:UTILization"](#) on page 368
- [":SBUS:FLEXray:COUNt:NULL"](#) on page 370
- [":SBUS:FLEXray:COUNt:RESet"](#) on page 371
- [":SBUS:FLEXray:COUNt:SYNC"](#) on page 372
- [":SBUS:FLEXray:COUNt:TOTal"](#) on page 373
- [":SBUS:UART:COUNt:ERRor"](#) on page 379
- [":SBUS:UART:COUNt:RESet"](#) on page 380
- [":SBUS:UART:COUNt:RXFRames"](#) on page 381
- [":SBUS:UART:COUNt:TXFRames"](#) on page 382
- [":TRIGger:EBURst:COUNt"](#) on page 434
- [":WAVEform:COUNt"](#) on page 532
- [":WAVEform:SEGMENTed:COUNt"](#) on page 543
- COUNter, [":MEASure:COUNter"](#) on page 296
- COUPling Commands:
  - [":CHANnel<n>:COUPling"](#) on page 208
  - [":TRIGger\[:EDGE\]:COUPling"](#) on page 438
- D**
  - DAALias, [":ACQuire:DAALias"](#) on page 177
  - DATA Commands:
    - [":DISPlay:DATA"](#) on page 233
    - [":TRIGger:CAN:PATtern:DATA"](#) on page 418
    - [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 419
    - [":TRIGger:IIC:PATtern:DATA"](#) on page 466
    - [":TRIGger:IIC:PATtern:DATA2"](#) on page 467
    - [":TRIGger:IIC:SOURce:DATA"](#) on page 469
    - [":TRIGger:SPI:PATtern:DATA"](#) on page 493
    - [":TRIGger:SPI:SOURce:DATA"](#) on page 496
    - [":TRIGger:UART:DATA"](#) on page 509
    - [":WAVEform:DATA"](#) on page 533
  - DATE Commands:
    - [":CALibrate:DATE"](#) on page 197
    - [":SYSTem:DATE"](#) on page 385
  - DEFine, [":MEASure:DEFine"](#) on page 297

- DEFinition Commands:
  - [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 630
  - [":TRIGger:LIN:SIGNal:DEFinition"](#) on page 631
- DELay Commands:
  - [":MEASure:DELay"](#) on page 300
  - [":TIMEbase:DELay"](#) on page 628
- DESTination, [":HARDcopy:DESTination"](#) on page 605
- DEvice, [":HARDcopy:DEvice"](#) on page 606
- [":DIGital<n>:DISPlay"](#) on page 225
- [":DIGital<n>:LABel"](#) on page 226
- [":DIGital<n>:POSition"](#) on page 227
- [":DIGital<n>:SIZE"](#) on page 228
- [":DIGital<n>:THReshold"](#) on page 229
- [":DIGitize"](#) on page 145
- DISPlay Commands:
  - [":BUS<n>:DISPlay"](#) on page 193
  - [":CHANnel<n>:DISPlay"](#) on page 209
  - [":DIGital<n>:DISPlay"](#) on page 225
  - [":FUNctIon:DISPlay"](#) on page 254
  - [":POD<n>:DISPlay"](#) on page 334
  - [":SBUS:DISPlay"](#) on page 369
- [":DISPlay:CLEar"](#) on page 232
- [":DISPlay:CONNect"](#) on page 598
- [":DISPlay:DATA"](#) on page 233
- [":DISPlay:LABel"](#) on page 235
- [":DISPlay:LABList"](#) on page 236
- [":DISPlay:ORDer"](#) on page 599
- [":DISPlay:PERsistence"](#) on page 237
- [":DISPlay:SOURce"](#) on page 238
- [":DISPlay:VECTors"](#) on page 239
- DMINus, [":TRIGger:USB:SOURce:DMINus"](#) on page 519
- DPLus, [":TRIGger:USB:SOURce:DPLus"](#) on page 520
- DSP, [":SYSTem:DSP"](#) on page 386
- DURation, [":TRIGger:DURation Commands"](#) on page 427
- DUTYcycle, [":MEASure:DUTYcycle"](#) on page 302

- E**
  - EBURst, [":TRIGger:EBURst Commands"](#) on page 433
  - EDGE, [":TRIGger\[:EDGE\] Commands"](#) on page 437
  - [":ERASe"](#) on page 600
  - ERRor Commands:
    - [":SBUS:CAN:COUNt:ERRor"](#) on page 364
    - [":SBUS:UART:COUNt:ERRor"](#) on page 379
    - [":SYSTem:ERRor"](#) on page 387
    - [":TRIGger:FLEXray:ERRor:TYPE"](#) on page 444
  - ["\\*ESE \(Standard Event Status Enable\)"](#) on page 114
  - ["\\*ESR \(Standard Event Status Register\)"](#) on page 116
  - [":EXTeRnal:BWLimit"](#) on page 242
  - [":EXTeRnal:IMPedance"](#) on page 243
  - [":EXTeRnal:INPut"](#) on page 601
  - [":EXTeRnal:PMODE"](#) on page 602
  - [":EXTeRnal:PROBe"](#) on page 244
  - [":EXTeRnal:PROBe:ID"](#) on page 245
  - [":EXTeRnal:PROBe:STYPe"](#) on page 246
  - [":EXTeRnal:PROTection"](#) on page 247
  - [":EXTeRnal:RANGe"](#) on page 248
  - [":EXTeRnal:UNITs"](#) on page 249
- F**
  - FACToRs Commands:
    - [":HARDcopy:FACToRs"](#) on page 271
    - [":SAVE:IMAGe:FACToRs"](#) on page 348
  - FALLtime, [":MEASure:FALLtime"](#) on page 303
  - FFEed, [":HARDcopy:FFEed"](#) on page 272
  - FILeName Commands:
    - [":HARDcopy:FILeName"](#) on page 607
    - [":RECall:FILeName"](#) on page 339
    - [":SAVE:FILeName"](#) on page 345
  - FIND, [":TRIGger:SEQuence:FIND"](#) on page 484
  - FLEXray Commands:
    - [":SBUS:FLEXray:COUNt:NULL"](#) on page 370
    - [":SBUS:FLEXray:COUNt:RESet"](#) on page 371
    - [":SBUS:FLEXray:COUNt:SYNC"](#) on page 372
    - [":SBUS:FLEXray:COUNt:TOTAl"](#) on page 373

- [":TRIGger:FLEXray:ERRor:TYPE"](#) on page 444
- [":TRIGger:FLEXray:FRAMe:CCBase"](#) on page 446
- [":TRIGger:FLEXray:FRAMe:CCRepetition"](#) on page 447
- [":TRIGger:FLEXray:FRAMe:ID"](#) on page 448
- [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 449
- [":TRIGger:FLEXray:TIME:CBASe"](#) on page 450
- [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 451
- [":TRIGger:FLEXray:TIME:SEGMENT"](#) on page 452
- [":TRIGger:FLEXray:TIME:SLOT"](#) on page 453
- [":TRIGger:FLEXray:TRIGger"](#) on page 454
- **FORMat Commands:**
  - [":HARDcopy:FORMat"](#) on page 608
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:WAVEform:FORMat"](#) on page 355
  - [":WAVEform:FORMat"](#) on page 535
- **FRAMe Commands:**
  - [":TRIGger:FLEXray:FRAMe:CCBase"](#) on page 446
  - [":TRIGger:FLEXray:FRAMe:CCRepetition"](#) on page 447
  - [":TRIGger:FLEXray:FRAMe:ID"](#) on page 448
  - [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 449
  - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 497
- **FRAMing Commands:**
  - [":SBUS:UART:FRAMing"](#) on page 383
  - [":TRIGger:SPI:FRAMing"](#) on page 492
- **FREQuency**, [":MEASure:FREQuency"](#) on page 304
- [":FUNCTion:CENTer"](#) on page 253
- [":FUNCTion:DISPlay"](#) on page 254
- [":FUNCTion:GOFT:OPERation"](#) on page 255
- [":FUNCTion:GOFT:SOURce1"](#) on page 256
- [":FUNCTion:GOFT:SOURce2"](#) on page 257
- [":FUNCTion:OFFSet"](#) on page 258
- [":FUNCTion:OPERation"](#) on page 259
- [":FUNCTion:RANGe"](#) on page 260
- [":FUNCTion:REFerence"](#) on page 261
- [":FUNCTion:SCALE"](#) on page 262

- [":FUNCTION:SOURce"](#) on page 603
  - [":FUNCTION:SOURce1"](#) on page 263
  - [":FUNCTION:SOURce2"](#) on page 264
  - [":FUNCTION:SPAN"](#) on page 265
  - [":FUNCTION:VIEW"](#) on page 604
  - [":FUNCTION:WINDow"](#) on page 266
- G**
- GLITCh (Pulse Width), [":TRIGger:GLITCh Commands"](#) on page 455
  - GOFT Commands:
    - [":FUNCTION:GOFT:OPERation"](#) on page 255
    - [":FUNCTION:GOFT:SOURce1"](#) on page 256
    - [":FUNCTION:GOFT:SOURce2"](#) on page 257
  - GRAYscale, [":HARDcopy:GRAYscale"](#) on page 609
  - GREaterthan Commands:
    - [":TRIGger:DURation:GREaterthan"](#) on page 428
    - [":TRIGger:GLITCh:GREaterthan"](#) on page 457
- H**
- [":HARDcopy:AREA"](#) on page 269
  - [":HARDcopy:APRinter"](#) on page 270
  - [":HARDcopy:DESTination"](#) on page 605
  - [":HARDcopy:DEVice"](#) on page 606
  - [":HARDcopy:FACTors"](#) on page 271
  - [":HARDcopy:FFEed"](#) on page 272
  - [":HARDcopy:FILEname"](#) on page 607
  - [":HARDcopy:FORMat"](#) on page 608
  - [":HARDcopy:GRAYscale"](#) on page 609
  - [":HARDcopy:IGColors"](#) on page 610
  - [":HARDcopy:INKSaver"](#) on page 273
  - [":HARDcopy:PALette"](#) on page 274
  - [":HARDcopy:PDRiver"](#) on page 611
  - [":HARDcopy:PRinter:LIST"](#) on page 275
  - [":HARDcopy:START"](#) on page 276
  - HFReject, [":TRIGger:HFReject"](#) on page 409
  - HOLDoff, [":TRIGger:HOLDoff"](#) on page 410
- I**
- ID Commands:
    - [":TRIGger:CAN:PATtern:ID"](#) on page 420

- [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 421
- [":TRIGger:FLEXray:FRAMe:ID"](#) on page 448
- IDLE Commands:
  - [":TRIGger:EBURst:IDLE"](#) on page 435
  - [":TRIGger:UART:IDLE"](#) on page 510
- ["\\*IDN \(Identification Number\)"](#) on page 118
- IIC Commands:
  - [":SBUS:IIC:ASize"](#) on page 374
  - [":TRIGger:IIC Commands"](#) on page 464
- IGColors Commands:
  - [":HARDcopy:IGColors"](#) on page 610
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
- IMAGe Commands:
  - [":RECall:IMAGe\[:STARt\]"](#) on page 340
  - [":SAVE:IMAGe:AREA"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351
  - [":SAVE:IMAGe\[:STARt\]"](#) on page 346
- IMPedance Commands:
  - [":CHANnel<n>:IMPedance"](#) on page 210
  - [":EXTernal:IMPedance"](#) on page 243
- INDeX, [":ACQuire:SEGMENTed:INDeX"](#) on page 182
- INKSaver, [":HARDcopy:INKSaver"](#) on page 273
- INVert, [":CHANnel<n>:INVert"](#) on page 211
- L**
  - LABEL Commands:
    - [":BUS<n>:LABel"](#) on page 194
    - [":CALibrate:LABel"](#) on page 198
    - [":CHANnel:LABel"](#) on page 593
    - [":CHANnel<n>:LABel"](#) on page 212
    - [":DIGital<n>:LABel"](#) on page 226
    - [":DISPlay:LABel"](#) on page 235
  - LABList, [":DISPlay:LABList"](#) on page 236
  - LENGth Commands:



- [":SAVE:WAVEform:LENGth"](#) on page 356
  - [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 419
  - LESSthan Commands:
    - [":TRIGger:DURation:LESSthan"](#) on page 429
    - [":TRIGger:GLITCh:LESSthan"](#) on page 458
  - LEVel Commands:
    - [":TRIGger\[:EDGE\]:LEVel"](#) on page 439
    - [":TRIGger:GLITCh:LEVel"](#) on page 459
  - LIN Commands:
    - [":SBUS:LIN:PARity"](#) on page 375
    - [":TRIGger:LIN Commands"](#) on page 473
  - LINE, [":TRIGger:TV:LINE"](#) on page 499
  - LIST, [":HARDcopy:PRinter:LIST"](#) on page 275
  - LOCK Commands:
    - [":SYSTem:LOCK"](#) on page 388
    - [":SYSTem:PROTection:LOCK"](#) on page 389
  - LOWer, [":MEASure:LOWer"](#) on page 612
  - ["\\*LRN \(Learn Device Setup\)"](#) on page 119
- M**
- [":MARKer:MODE"](#) on page 279
  - [":MARKer:X1Position"](#) on page 280
  - [":MARKer:X1Y1source"](#) on page 281
  - [":MARKer:X2Position"](#) on page 282
  - [":MARKer:X2Y2source"](#) on page 283
  - [":MARKer:XDELta"](#) on page 284
  - [":MARKer:Y1Position"](#) on page 285
  - [":MARKer:Y2Position"](#) on page 286
  - [":MARKer:YDELta"](#) on page 287
  - MASK, [":BUS<n>:MASK"](#) on page 195
  - [":MEASure:CLEar"](#) on page 295
  - [":MEASure:COUNter"](#) on page 296
  - [":MEASure:DEFine"](#) on page 297
  - [":MEASure:DELay"](#) on page 300
  - [":MEASure:DUTYcycle"](#) on page 302
  - [":MEASure:FALLtime"](#) on page 303
  - [":MEASure:FREQuency"](#) on page 304

- [":MEASure:LOWer"](#) on page 612
- [":MEASure:NWIDth"](#) on page 305
- [":MEASure:OVERshoot"](#) on page 306
- [":MEASure:PERiod"](#) on page 308
- [":MEASure:PHASe"](#) on page 309
- [":MEASure:PREShoot"](#) on page 310
- [":MEASure:PWIDth"](#) on page 311
- [":MEASure:RISetime"](#) on page 312
- [":MEASure:SCRatch"](#) on page 613
- [":MEASure:SDEViation"](#) on page 313
- [":MEASure:SHOW"](#) on page 314
- [":MEASure:SOURce"](#) on page 315
- [":MEASure:TDELta"](#) on page 614
- [":MEASure:TEDGE"](#) on page 317
- [":MEASure:THResholds"](#) on page 615
- [":MEASure:TMAX"](#) on page 616
- [":MEASure:TMIN"](#) on page 617
- [":MEASure:TSTArt"](#) on page 618
- [":MEASure:TSTOp"](#) on page 619
- [":MEASure:TVALue"](#) on page 319
- [":MEASure:TVOLt"](#) on page 620
- [":MEASure:UPPer"](#) on page 622
- [":MEASure:VAMPLitude"](#) on page 321
- [":MEASure:VAVerage"](#) on page 322
- [":MEASure:VBASe"](#) on page 323
- [":MEASure:VDELta"](#) on page 623
- [":MEASure:VMAX"](#) on page 324
- [":MEASure:VMIN"](#) on page 325
- [":MEASure:VPP"](#) on page 326
- [":MEASure:VRATio"](#) on page 327
- [":MEASure:VRMS"](#) on page 328
- [":MEASure:VSTArt"](#) on page 624
- [":MEASure:VSTOp"](#) on page 625
- [":MEASure:VTIME"](#) on page 329
- [":MEASure:VTOP"](#) on page 330

- [":MEASure:XMAX"](#) on page 331
- [":MEASure:XMIN"](#) on page 332
- [":MERGe"](#) on page 153
- MODE Commands:
  - [":ACQuire:MODE"](#) on page 178
  - [":MARKer:MODE"](#) on page 279
  - [":SBUS:BUSDoctor:MODE"](#) on page 363
  - [":SBUS:MODE"](#) on page 376
  - [":TIMEbase:MODE"](#) on page 395
  - [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 421
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:TV:MODE"](#) on page 500
  - [":WAVeform:POINts:MODE"](#) on page 538
- N**
  - NREJect, [":TRIGger:NREJect"](#) on page 412
  - NULL, [":SBUS:FLEXray:COUNt:NULL"](#) on page 370
  - NWIDth, [":MEASure:NWIDth"](#) on page 305
- O**
  - OFFSet Commands:
    - [":CHANnel<n>:OFFSet"](#) on page 213
    - [":FUNctio:n:OFFSet"](#) on page 258
  - ["\\*OPC \(Operation Complete\)"](#) on page 120
  - [":OPEE \(Operation Status Enable Register\)"](#) on page 154
  - OPERation Commands:
    - [":FUNctio:n:GOFT:OPERation"](#) on page 255
    - [":FUNctio:n:OPERation"](#) on page 259
  - [":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 156
  - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 158
  - ["\\*OPT \(Option Identification\)"](#) on page 121
  - ORDer, [":DISPlay:ORDer"](#) on page 599
  - OVERload, [":SBUS:CAN:COUNt:OVERload"](#) on page 365
  - OVERshoot, [":MEASure:OVERshoot"](#) on page 306
  - [":OVLenable \(Overload Event Enable Register\)"](#) on page 160
  - [":OVLRegister \(Overload Event Register\)"](#) on page 162
- P**
  - PALette Commands:

- [":HARDcopy:PALETTE"](#) on page 274
  - [":SAVE:IMAGE:PALETTE"](#) on page 351
- PARity Commands:
  - [":SBUS:LIN:PARity"](#) on page 375
  - [":TRIGger:UART:PARity"](#) on page 511
- PATtern Commands:
  - [":TRIGger:CAN:PATtern:DATA"](#) on page 418
  - [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 419
  - [":TRIGger:CAN:PATtern:ID"](#) on page 420
  - [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 421
  - [":TRIGger:DURation:PATtern"](#) on page 430
  - [":TRIGger:IIC:PATtern:ADDReSS"](#) on page 465
  - [":TRIGger:IIC:PATtern:DATA"](#) on page 466
  - [":TRIGger:IIC:PATtern:DATA2"](#) on page 467
  - [":TRIGger:PATtern"](#) on page 413
  - [":TRIGger:SEQuence:PATtern"](#) on page 485
  - [":TRIGger:SPI:PATtern:DATA"](#) on page 493
  - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 494
- PDRiver, [":HARDcopy:PDRiver"](#) on page 611
- PERiod, [":MEASure:PERiod"](#) on page 308
- PERSistence, [":DISPlay:PERSistence"](#) on page 237
- PHASe, [":MEASure:PHASe"](#) on page 309
- PMODE, [":CHANnel<n>:PMODE"](#) on page 597
- [":POD<n>:DISPlay"](#) on page 334
- [":POD<n>:SIZE"](#) on page 335
- [":POD<n>:THReShold"](#) on page 336
- POINts Commands:
  - [":ACQuire:POINts"](#) on page 179
  - [":WAVeform:POINts"](#) on page 536
  - [":WAVeform:POINts:MODE"](#) on page 538
- POLarity Commands:
  - [":TRIGger:GLITCh:POLarity"](#) on page 460
  - [":TRIGger:TV:POLarity"](#) on page 501
  - [":TRIGger:UART:POLarity"](#) on page 512
- POSition Commands:

- ":DIGital<n>:POSition" on page 227
- ":TIMebase:POSition" on page 396
- ":TIMebase:WINDow:POSition" on page 402
- PREamble, ":WAVEform:PREamble" on page 540
- PREShoot, ":MEASure:PREShoot" on page 310
- ":PRINT" on page 164
- ":PRINT?" on page 626
- PRINter, ":HARDcopy:PRinter:LIST" on page 275
- PROBe Commands:
  - ":CHANnel<n>:PROBe" on page 214
  - ":EXTernal:PROBe" on page 244
- PROTection Commands:
  - ":CHANnel<n>:PROTection" on page 218
  - ":EXTernal:PROTection" on page 247
  - ":SYSTem:PROTection:LOCK" on page 389
- Pulse Width (GLITch), ":TRIGger:GLITch Commands" on page 455
- PWD Commands:
  - ":RECall:PWD" on page 341
  - ":SAVE:PWD" on page 352
- PWIDth, ":MEASure:PWIDth" on page 311
- Q** • QUALifier Commands:
  - ":TRIGger:DURation:QUALifier" on page 431
  - ":TRIGger:GLITch:QUALifier" on page 461
  - ":TRIGger:IIC:TRIGger:QUALifier" on page 470
  - ":TRIGger:UART:QUALifier" on page 513
- R** • RANGE Commands:
  - ":CHANnel<n>:RANGe" on page 219
  - ":EXTernal:RANGe" on page 248
  - ":FUNCTion:RANGe" on page 260
  - ":TIMebase:RANGe" on page 397
  - ":TIMebase:WINDow:RANGe" on page 403
  - ":TRIGger:DURation:RANGe" on page 432
  - ":TRIGger:GLITch:RANGe" on page 462
- "\*RCL (Recall)" on page 122

- [":RECall:FILEname"](#) on page 339
- [":RECall:IMAGe\[:STArT\]"](#) on page 340
- [":RECall:PWD"](#) on page 341
- [":RECall:SETup\[:STArT\]"](#) on page 342
- REFClock, [":TIMEbase:REFClock"](#) on page 398
- REFerence Commands:
  - [":FUNcTion:REFerence"](#) on page 261
  - [":TIMEbase:REFerence"](#) on page 399
- REJect, [":TRIGger\[:EDGE\]:REJect"](#) on page 440
- RESet Commands:
  - [":SBUS:CAN:COUNt:RESet"](#) on page 366
  - [":SBUS:FLEXray:COUNt:RESet"](#) on page 371
  - [":SBUS:UART:COUNt:RESet"](#) on page 380
  - [":TRIGger:SEQuence:RESet"](#) on page 486
- RISetime, [":MEASure:RISetime"](#) on page 312
- ["Root \(:\)"](#) Commands on page 134
- RSIGnal, [":ACQuire:RSIGnal"](#) on page 180
- ["\\*RST \(Reset\)"](#) on page 123
- [":RUN"](#) on page 165
- RX, [":TRIGger:UART:SOURce:RX"](#) on page 514
- RXFRames, [":SBUS:UART:COUNt:RXFRames"](#) on page 381
- S**
  - SAMPlpoint Commands:
    - [":TRIGger:CAN:SAMPlpoint"](#) on page 422
    - [":TRIGger:LIN:SAMPlpoint"](#) on page 475
  - ["\\*SAV \(Save\)"](#) on page 126
  - [":SAVE:FILEname"](#) on page 345
  - [":SAVE:IMAGe:AREA"](#) on page 347
  - [":SAVE:IMAGe:FACTors"](#) on page 348
  - [":SAVE:IMAGe:FORMat"](#) on page 349
  - [":SAVE:IMAGe:INKSaver"](#) on page 350
  - [":SAVE:IMAGe:PALette"](#) on page 351
  - [":SAVE:IMAGe\[:STArT\]"](#) on page 346
  - [":SAVE:PWD"](#) on page 352
  - [":SAVE:SETup\[:STArT\]"](#) on page 353
  - [":SAVE:WAVEform:FORMat"](#) on page 355

- ":SAVE:WAVEform:LENGth" on page 356
- ":SAVE:WAVEform[:STARt]" on page 354
- ":SBUS:BUSDoctor:ADDReSS" on page 360
- ":SBUS:BUSDoctor:BAUDrate" on page 361
- ":SBUS:BUSDoctor:CHANnel" on page 362
- ":SBUS:BUSDoctor:MODE" on page 363
- ":SBUS:CAN:COUNt:ERRor" on page 364
- ":SBUS:CAN:COUNt:OVERload" on page 365
- ":SBUS:CAN:COUNt:RESet" on page 366
- ":SBUS:CAN:COUNt:TOTal" on page 367
- ":SBUS:CAN:COUNt:UTILization" on page 368
- ":SBUS:DISPlay" on page 369
- ":SBUS:FLEXray:COUNt:NULL" on page 370
- ":SBUS:FLEXray:COUNt:RESet" on page 371
- ":SBUS:FLEXray:COUNt:SYNC" on page 372
- ":SBUS:FLEXray:COUNt:TOTal" on page 373
- ":SBUS:IIC:ASIZe" on page 374
- ":SBUS:LIN:PARity" on page 375
- ":SBUS:MODE" on page 376
- ":SBUS:SPI:WIDTh" on page 377
- ":SBUS:UART:BASE" on page 378
- ":SBUS:UART:COUNt:ERRor" on page 379
- ":SBUS:UART:COUNt:RESet" on page 380
- ":SBUS:UART:COUNt:RXFRames" on page 381
- ":SBUS:UART:COUNt:TXFRames" on page 382
- ":SBUS:UART:FRAMing" on page 383
- SCALE Commands:
  - ":CHANnel<n>:SCALE" on page 220
  - ":FUNCTion:SCALE" on page 262
  - ":TIMEbase:SCALE" on page 400
  - ":TIMEbase:WINDow:SCALE" on page 404
- SCRatch, ":MEASure:SCRatch" on page 613
- SDEViation, ":MEASure:SDEViation" on page 313
- ":SERial" on page 166
- SEGMENT, ":TRIGger:FLEXray:TIME:SEGMENT" on page 452

- SEGmented Commands:
  - [":ACQuire:SEGmented:COUNT"](#) on page 181
  - [":ACQuire:SEGmented:INDEX"](#) on page 182
  - [":WAVEform:SEGmented:COUNT"](#) on page 543
  - [":WAVEform:SEGmented:TTAG"](#) on page 544
- SETup Commands:
  - [":RECall:SETup\[:START\]"](#) on page 342
  - [":SAVE:SETup\[:START\]"](#) on page 353
  - [":SYSTem:SETup"](#) on page 390
- SEQUENCE, [":TRIGger:SEQuence Commands"](#) on page 481
- SHOW, [":MEASure:SHOW"](#) on page 314
- SLOT, [":TRIGger:FLEXray:TIME:SLOT"](#) on page 453
- SIGNAL Commands:
  - [":TRIGger:CAN:SIGNAL:BAUDrate"](#) on page 423
  - [":TRIGger:CAN:SIGNAL:DEFinition"](#) on page 630
  - [":TRIGger:LIN:SIGNAL:BAUDrate"](#) on page 476
  - [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 631
- [":SINGLE"](#) on page 167
- SIZE Commands:
  - [":DIGital<n>:SIZE"](#) on page 228
  - [":POD<n>:SIZE"](#) on page 335
- SKEW, [":CHANnel<n>:PROBe:SKEW"](#) on page 216
- SLOPe Commands:
  - [":TRIGger:EBURst:SLOPe"](#) on page 436
  - [":TRIGger\[:EDGE\]:SLOPe"](#) on page 441
  - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 490
- SOURce Commands:
  - [":DISPlay:SOURce"](#) on page 238
  - [":FUNCTion:SOURce"](#) on page 603
  - [":MEASure:SOURce"](#) on page 315
  - [":TRIGger:CAN:SOURce"](#) on page 424
  - [":TRIGger:GLITCh:SOURce"](#) on page 463
  - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 468
  - [":TRIGger:IIC:SOURce:DATA"](#) on page 469
  - [":TRIGger:LIN:SOURce"](#) on page 477



- [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
- [":TRIGger:SPI:SOURce:DATA"](#) on page 496
- [":TRIGger:SPI:SOURce:FRAME"](#) on page 497
- [":TRIGger:TV:SOURce"](#) on page 502
- [":TRIGger:UART:SOURce:RX"](#) on page 514
- [":TRIGger:UART:SOURce:TX"](#) on page 515
- [":TRIGger:USB:SOURce:DMinus"](#) on page 519
- [":TRIGger:USB:SOURce:DPlus"](#) on page 520
- [":WAVEform:SOURce"](#) on page 545
- [":WAVEform:SOURce:SUBSource"](#) on page 549
- SOURce1 Commands:
  - [":FUNCTION:GOFt:SOURce1"](#) on page 256
  - [":FUNCTION:SOURce1"](#) on page 263
- SOURce2 Commands:
  - [":FUNCTION:GOFt:SOURce2"](#) on page 257
  - [":FUNCTION:SOURce2"](#) on page 264
- SPAN, [":FUNCTION:SPAN"](#) on page 265
- SPEEd, [":TRIGger:USB:SPEEd"](#) on page 521
- SPI Commands:
  - [":SBUS:SPI:WIDTh"](#) on page 377
  - [":TRIGger:SPI Commands"](#) on page 489
- SRATe, [":ACQuire:SRATe"](#) on page 184
- ["\\*SRE \(Service Request Enable\)"](#) on page 127
- STANdard Commands:
  - [":TRIGger:LIN:STANdard"](#) on page 478
  - [":TRIGger:TV:STANdard"](#) on page 503
- STARt Commands:
  - [":CALibrate:STARt"](#) on page 199
  - [":HARDcopy:STARt"](#) on page 276
  - [":RECall:IMAGe\[:STARt\]"](#) on page 340
  - [":RECall:SETup\[:STARt\]"](#) on page 342
  - [":SAVE:IMAGe\[:STARt\]"](#) on page 346
  - [":SAVE:SETup\[:STARt\]"](#) on page 353
  - [":SAVE:WAVEform\[:STARt\]"](#) on page 354
- STATus Commands:

- [":CALibrate:STATus"](#) on page 200
  - [":STATus"](#) on page 168
  - ["\\*STB \(Read Status Byte\)"](#) on page 129
  - [":STOP"](#) on page 169
  - SUBSource, [":WAVEform:SOURce:SUBSource"](#) on page 549
  - SWEep, [":TRIGger:SWEep"](#) on page 415
  - SWITCh, [":CALibrate:SWITCh"](#) on page 201
  - SYNC, [":SBUS:FLEXray:COUNt:SYNC"](#) on page 372
  - SYNCbreak, [":TRIGger:LIN:SYNCbreak"](#) on page 479
  - [":SYSTem:DATE"](#) on page 385
  - [":SYSTem:DSP"](#) on page 386
  - [":SYSTem:ERRor"](#) on page 387
  - [":SYSTem:LOCK"](#) on page 388
  - [":SYSTem:SETup"](#) on page 390
  - [":SYSTem:TIME"](#) on page 392
- T**
- TDELta, [":MEASure:TDELta"](#) on page 614
  - TEDGe, [":MEASure:TEDGe"](#) on page 317
  - TEMPerature, [":CALibrate:TEMPerature"](#) on page 202
  - [":TER \(Trigger Event Register\)"](#) on page 170
  - THReshold Commands:
    - [":CHANnel:THReshold"](#) on page 594
    - [":DIGital<n>:THReshold"](#) on page 229
    - [":MEASure:THResholds"](#) on page 615
    - [":POD<n>:THReshold"](#) on page 336
    - [":TRIGger:THReshold"](#) on page 632
  - THResholds, [":MEASure:THResholds"](#) on page 615
  - TIME Commands:
    - [":CALibrate:TIME"](#) on page 203
    - [":SYSTem:TIME"](#) on page 392
    - [":TRIGger:FLEXray:TIME:CBASe"](#) on page 450
    - [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 451
    - [":TRIGger:FLEXray:TIME:SEGMENT"](#) on page 452
    - [":TRIGger:FLEXray:TIME:SLOT"](#) on page 453
  - [":TIMEbase:DELay"](#) on page 628
  - [":TIMEbase:MODE"](#) on page 395

- [":TIMEbase:POSition"](#) on page 396
- [":TIMEbase:RANGe"](#) on page 397
- [":TIMEbase:REFClock"](#) on page 398
- [":TIMEbase:REFerence"](#) on page 399
- [":TIMEbase:SCALe"](#) on page 400
- [":TIMEbase:VERNier"](#) on page 401
- [":TIMEbase:WINDow:POSition"](#) on page 402
- [":TIMEbase:WINDow:RANGe"](#) on page 403
- [":TIMEbase:WINDow:SCALe"](#) on page 404
- [TIMEout, ":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
- [TIMER, ":TRIGger:SEQuence:TIMER"](#) on page 487
- [TMAX, ":MEASure:TMAX"](#) on page 616
- [TMIN, ":MEASure:TMIN"](#) on page 617
- [TOTal Commands:](#)
  - [":SBUS:CAN:COUNt:TOTal"](#) on page 367
  - [":SBUS:FLEXray:COUNt:TOTal"](#) on page 373
- ["\\*TRG \(Trigger\)"](#) on page 131
- [TRIGger Commands:](#)
  - [":TRIGger:CAN:TRIGger"](#) on page 425
  - [":TRIGger:FLEXray:TRIGger"](#) on page 454
  - [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 470
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471
  - [":TRIGger:LIN:TRIGger"](#) on page 480
  - [":TRIGger:SEQuence:TRIGger"](#) on page 488
  - [":TRIGger:USB:TRIGger"](#) on page 522
- [":TRIGger:HFReject"](#) on page 409
- [":TRIGger:HOLDoff"](#) on page 410
- [":TRIGger:MODE"](#) on page 411
- [":TRIGger:NREJect"](#) on page 412
- [":TRIGger:PATtern"](#) on page 413
- [":TRIGger:SWEep"](#) on page 415
- [":TRIGger:THReshold"](#) on page 632
- [":TRIGger:CAN:ACKnowledge"](#) on page 629
- [":TRIGger:CAN:PATtern:DATA"](#) on page 418
- [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 419

- [":TRIGger:CAN:PATtern:ID"](#) on page 420
- [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 421
- [":TRIGger:CAN:SAMPlEpoint"](#) on page 422
- [":TRIGger:CAN:SIGNal:BAUDrate"](#) on page 423
- [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 630
- [":TRIGger:CAN:SOURce"](#) on page 424
- [":TRIGger:CAN:TRIGger"](#) on page 425
- [":TRIGger:DURation:GREaterthan"](#) on page 428
- [":TRIGger:DURation:LESSthan"](#) on page 429
- [":TRIGger:DURation:PATtern"](#) on page 430
- [":TRIGger:DURation:QUALifier"](#) on page 431
- [":TRIGger:DURation:RANGe"](#) on page 432
- [":TRIGger\[:EDGE\]:COUPling"](#) on page 438
- [":TRIGger\[:EDGE\]:LEVel"](#) on page 439
- [":TRIGger\[:EDGE\]:REJect"](#) on page 440
- [":TRIGger\[:EDGE\]:SLOPe"](#) on page 441
- [":TRIGger\[:EDGE\]:SOURce"](#) on page 442
- [":TRIGger:FLEXray:ERRor:TYPE"](#) on page 444
- [":TRIGger:FLEXray:FRAMe:CCBase"](#) on page 446
- [":TRIGger:FLEXray:FRAMe:CCRepetition"](#) on page 447
- [":TRIGger:FLEXray:FRAMe:ID"](#) on page 448
- [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 449
- [":TRIGger:FLEXray:TIME:CBASe"](#) on page 450
- [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 451
- [":TRIGger:FLEXray:TIME:SEGMENT"](#) on page 452
- [":TRIGger:FLEXray:TIME:SLOT"](#) on page 453
- [":TRIGger:FLEXray:TRIGger"](#) on page 454
- [":TRIGger:GLITch:GREaterthan"](#) on page 457
- [":TRIGger:GLITch:LESSthan"](#) on page 458
- [":TRIGger:GLITch:LEVel"](#) on page 459
- [":TRIGger:GLITch:POLarity"](#) on page 460
- [":TRIGger:GLITch:QUALifier"](#) on page 461
- [":TRIGger:GLITch:RANGe"](#) on page 462
- [":TRIGger:GLITch:SOURce"](#) on page 463
- [":TRIGger:HFReject"](#) on page 409

- [":TRIGger:HOLDoff"](#) on page 410
- [":TRIGger:IIC:PATtern:ADDReSS"](#) on page 465
- [":TRIGger:IIC:PATtern:DATA"](#) on page 466
- [":TRIGger:IIC:PATtern:DATA2"](#) on page 467
- [":TRIGger:IIC:SOURce:CLOCK"](#) on page 468
- [":TRIGger:IIC:SOURce:DATA"](#) on page 469
- [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 470
- [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471
- [":TRIGger:LIN:SIGNAL:BAUDrate"](#) on page 476
- [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 631
- [":TRIGger:LIN:SOURce"](#) on page 477
- [":TRIGger:LIN:TRIGger"](#) on page 480
- [":TRIGger:MODE"](#) on page 411
- [":TRIGger:NREject"](#) on page 412
- [":TRIGger:PATtern"](#) on page 413
- [":TRIGger:SEQuence:COUNt"](#) on page 482
- [":TRIGger:SEQuence:EDGE"](#) on page 483
- [":TRIGger:SEQuence:FIND"](#) on page 484
- [":TRIGger:SEQuence:PATtern"](#) on page 485
- [":TRIGger:SEQuence:RESet"](#) on page 486
- [":TRIGger:SEQuence:TIMer"](#) on page 487
- [":TRIGger:SEQuence:TRIGger"](#) on page 488
- [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 490
- [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 491
- [":TRIGger:SPI:FRAMing"](#) on page 492
- [":TRIGger:SPI:PATtern:DATA"](#) on page 493
- [":TRIGger:SPI:PATtern:WIDTh"](#) on page 494
- [":TRIGger:SPI:SOURce:CLOCK"](#) on page 495
- [":TRIGger:SPI:SOURce:DATA"](#) on page 496
- [":TRIGger:SPI:SOURce:FRAMe"](#) on page 497
- [":TRIGger:SWEep"](#) on page 415
- [":TRIGger:THReshold"](#) on page 632
- [":TRIGger:TV:LINE"](#) on page 499
- [":TRIGger:TV:MODE"](#) on page 500
- [":TRIGger:TV:POLarity"](#) on page 501

- [":TRIGger:TV:SOURce"](#) on page 502
- [":TRIGger:TV:STANdard"](#) on page 503
- [":TRIGger:TV:TVMODE"](#) on page 633
- [":TRIGger:UART:BAUDrate"](#) on page 506
- [":TRIGger:UART:BITorder"](#) on page 507
- [":TRIGger:UART:BURSt"](#) on page 508
- [":TRIGger:UART:DATA"](#) on page 509
- [":TRIGger:UART:IDLE"](#) on page 510
- [":TRIGger:UART:PARity"](#) on page 511
- [":TRIGger:UART:POLarity"](#) on page 512
- [":TRIGger:UART:QUALifier"](#) on page 513
- [":TRIGger:UART:SOURce:RX"](#) on page 514
- [":TRIGger:UART:SOURce:TX"](#) on page 515
- [":TRIGger:UART:TYPE"](#) on page 516
- [":TRIGger:UART:WIDTh"](#) on page 517
- [":TRIGger:USB:SOURce:DMINus"](#) on page 519
- [":TRIGger:USB:SOURce:DPLus"](#) on page 520
- [":TRIGger:USB:SPEed"](#) on page 521
- [":TRIGger:USB:TRIGger"](#) on page 522
- ["\\*TST \(Self Test\)"](#) on page 132
- TSTArt, [":MEASure:TSTArt"](#) on page 618
- TSTOp, [":MEASure:TSTOp"](#) on page 619
- TTAG, [":WAVEform:SEGmented:TTAG"](#) on page 544
- TV, [":TRIGger:TV Commands"](#) on page 498
- TVALue, [":MEASure:TVALue"](#) on page 319
- TVOLt, [":MEASure:TVOLt"](#) on page 620
- TX, [":TRIGger:UART:SOURce:TX"](#) on page 515
- TXFRames, [":SBUS:UART:COUNt:TXFRames"](#) on page 382
- TYPE Commands:
  - [":ACQuire:TYPE"](#) on page 185
  - [":WAVEform:TYPE"](#) on page 550
  - [":TRIGger:FLEXray:ERRor:TYPE"](#) on page 444
  - [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 449
  - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 471
  - [":TRIGger:UART:TYPE"](#) on page 516

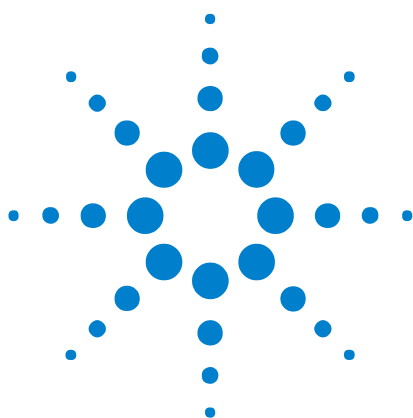
- U**
  - UART Commands:
    - [":SBUS:UART:BASE"](#) on page 378
    - [":SBUS:UART:COUNt:ERRor"](#) on page 379
    - [":SBUS:UART:COUNt:RESet"](#) on page 380
    - [":SBUS:UART:COUNt:RXFRames"](#) on page 381
    - [":SBUS:UART:COUNt:TXFRames"](#) on page 382
    - [":SBUS:UART:FRAMing"](#) on page 383
    - [":TRIGger:UART:BAUDrate"](#) on page 506
    - [":TRIGger:UART:BITorder"](#) on page 507
    - [":TRIGger:UART:BURSt"](#) on page 508
    - [":TRIGger:UART:DATA"](#) on page 509
    - [":TRIGger:UART:IDLE"](#) on page 510
    - [":TRIGger:UART:PARity"](#) on page 511
    - [":TRIGger:UART:POLarity"](#) on page 512
    - [":TRIGger:UART:QUALifier"](#) on page 513
    - [":TRIGger:UART:SOURce:RX"](#) on page 514
    - [":TRIGger:UART:SOURce:TX"](#) on page 515
    - [":TRIGger:UART:TYPE"](#) on page 516
    - [":TRIGger:UART:WIDTh"](#) on page 517
  - UNITs Commands:
    - [":CHANnel<n>:UNITs"](#) on page 221
    - [":EXTernal:UNITs"](#) on page 249
  - UNSigned, [":WAVEform:UNSigned"](#) on page 551
  - UPPer, [":MEASure:UPPer"](#) on page 622
  - USB, [":TRIGger:USB Commands"](#) on page 518
  - UTILization, [":SBUS:CAN:COUNt:UTILization"](#) on page 368
- V**
  - VAMplitude, [":MEASure:VAMplitude"](#) on page 321
  - VAVerage, [":MEASure:VAVerage"](#) on page 322
  - VBASE, [":MEASure:VBASe"](#) on page 323
  - VDELta, [":MEASure:VDELta"](#) on page 623
  - VECTors, [":DISPlay:VECTors"](#) on page 239
  - VERNier, [":CHANnel<n>:VERNier"](#) on page 222
  - [":VIEW"](#) on page 171
  - VMAX, [":MEASure:VMAX"](#) on page 324
  - VMIN, [":MEASure:VMIN"](#) on page 325

- VPP, [":MEASure:VPP"](#) on page 326
  - VRATio, [":MEASure:VRATio"](#) on page 327
  - VRMS, [":MEASure:VRMS"](#) on page 328
  - VSTArt, [":MEASure:VSTArt"](#) on page 624
  - VSTOp, [":MEASure:VSTOp"](#) on page 625
  - VTIMe, [":MEASure:VTIMe"](#) on page 329
  - VTOP, [":MEASure:VTOp"](#) on page 330
- W**
- ["\\*WAI \(Wait To Continue\)"](#) on page 133
  - WAVEform Commands:
    - [":SAVE:WAVEform:FORMat"](#) on page 355
    - [":SAVE:WAVEform:LENGth"](#) on page 356
    - [":SAVE:WAVEform\[:STARt\]"](#) on page 354
    - [":WAVEform:BYTeorder"](#) on page 531
    - [":WAVEform:COUNt"](#) on page 532
    - [":WAVEform:DATA"](#) on page 533
    - [":WAVEform:FORMat"](#) on page 535
    - [":WAVEform:POINts"](#) on page 536
    - [":WAVEform:POINts:MODE"](#) on page 538
    - [":WAVEform:PREamble"](#) on page 540
    - [":WAVEform:SEGmented:COUNt"](#) on page 543
    - [":WAVEform:SEGmented:TTAG"](#) on page 544
    - [":WAVEform:SOURce"](#) on page 545
    - [":WAVEform:SOURce:SUBSource"](#) on page 549
    - [":WAVEform:TYPE"](#) on page 550
    - [":WAVEform:UNSigned"](#) on page 551
    - [":WAVEform:VIEW"](#) on page 552
    - [":WAVEform:XINCrement"](#) on page 553
    - [":WAVEform:XORigin"](#) on page 554
    - [":WAVEform:XREFerence"](#) on page 555
    - [":WAVEform:YINCrement"](#) on page 556
    - [":WAVEform:YORigin"](#) on page 557
    - [":WAVEform:YREFerence"](#) on page 558
  - WIDTh Commands:
    - [":SBUS:SPI:WIDTh"](#) on page 377
    - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 494



- [":TRIGger:UART:WIDTh"](#) on page 517
- WINDow, [":FUNCTion:WINDow"](#) on page 266
- X**
  - X1Position, [":MARKer:X1Position"](#) on page 280
  - X1Y1source, [":MARKer:X1Y1source"](#) on page 281
  - X2Position, [":MARKer:X2Position"](#) on page 282
  - X2Y2source, [":MARKer:X2Y2source"](#) on page 283
  - XDELta, [":MARKer:XDELta"](#) on page 284
  - XINCrement, [":WAVEform:XINCrement"](#) on page 553
  - XMAX, [":MEASure:XMAX"](#) on page 331
  - XMIN, [":MEASure:XMIN"](#) on page 332
  - XORigin, [":WAVEform:XORigin"](#) on page 554
  - XREFerence, [":WAVEform:XREFerence"](#) on page 555
- Y**
  - Y1Position, [":MARKer:Y1Position"](#) on page 285
  - Y2Position, [":MARKer:Y2Position"](#) on page 286
  - YDELta, [":MARKer:YDELta"](#) on page 287
  - YINCrement, [":WAVEform:YINCrement"](#) on page 556
  - YORigin, [":WAVEform:YORigin"](#) on page 557
  - YREFerence, [":WAVEform:YREFerence"](#) on page 558





## 7 Obsolete and Discontinued Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see "[Obsolete Commands](#)" on page 676).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see <a href="#">page 207</a> )	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see <a href="#">page 208</a> )	
ANALog<n>:INVert	:CHANnel<n>:INVert (see <a href="#">page 211</a> )	
ANALog<n>:LABel	:CHANnel<n>:LABel (see <a href="#">page 212</a> )	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see <a href="#">page 213</a> )	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see <a href="#">page 214</a> )	
ANALog<n>:PMODE	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see <a href="#">page 219</a> )	
:CHANnel:ACTivity (see <a href="#">page 592</a> )	:ACTivity (see <a href="#">page 137</a> )	
:CHANnel:LABel (see <a href="#">page 593</a> )	:CHANnel<n>:LABel (see <a href="#">page 212</a> ) or :DIGital<n>:LABel (see <a href="#">page 226</a> )	use CHANnel<n>:LABel for analog channels and use DIGital<n>:LABel for digital channels
:CHANnel:THReshold (see <a href="#">page 594</a> )	:POD<n>:THReshold (see <a href="#">page 336</a> ) or :DIGital<n>:THReshold (see <a href="#">page 229</a> )	
:CHANnel2:SKEW (see <a href="#">page 595</a> )	:CHANnel<n>:PROBe:SKEW (see <a href="#">page 216</a> )	



## 7 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel<n>:INPut (see <a href="#">page 596</a> )	:CHANnel<n>:IMPedance (see <a href="#">page 210</a> )	
:CHANnel<n>:PMODE (see <a href="#">page 597</a> )	none	
:DISPlay:CONNect (see <a href="#">page 598</a> )	:DISPlay:VECTors (see <a href="#">page 239</a> )	
:DISPlay:ORDer (see <a href="#">page 599</a> )	none	
:ERASe (see <a href="#">page 600</a> )	:CDISplay (see <a href="#">page 144</a> )	
:EXTernal:INPut (see <a href="#">page 601</a> )	:EXTernal:IMPedance (see <a href="#">page 243</a> )	
:EXTernal:PMODE (see <a href="#">page 602</a> )	none	
FUNcTion1, FUNcTion2	:FUNcTion Commands (see <a href="#">page 250</a> )	ADD not included
:FUNcTion:SOURce (see <a href="#">page 603</a> )	:FUNcTion:SOURce1 (see <a href="#">page 263</a> )	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNcTion:VIEW (see <a href="#">page 604</a> )	:FUNcTion:DISPlay (see <a href="#">page 254</a> )	
:HARDcopy:DESTination (see <a href="#">page 605</a> )	:HARDcopy:FILeName (see <a href="#">page 607</a> )	
:HARDcopy:DEVIce (see <a href="#">page 606</a> )	:HARDcopy:FORMat (see <a href="#">page 608</a> )	PLOTter, THINkjet not supported; TIF, BMP, CSV, SEIko added
:HARDcopy:FILeName (see <a href="#">page 607</a> )	:RECall:FILeName (see <a href="#">page 339</a> ) :SAVE:FILeName (see <a href="#">page 339</a> )	
:HARDcopy:FORMat (see <a href="#">page 608</a> )	:HARDcopy:APRinter (see <a href="#">page 270</a> ) :SAVE:IMAGe:FORMat (see <a href="#">page 349</a> ) :SAVE:WAVEform:FORMat (see <a href="#">page 355</a> )	
:HARDcopy:GRAYscale (see <a href="#">page 609</a> )	:HARDcopy:PALette (see <a href="#">page 274</a> )	
:HARDcopy:IGColors (see <a href="#">page 610</a> )	:HARDcopy:INKSaver (see <a href="#">page 273</a> )	

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:PDRiver (see <a href="#">page 611</a> )	:HARDcopy:APRinter (see <a href="#">page 270</a> )	
:MEASure:LOWer (see <a href="#">page 612</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 297</a> )	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see <a href="#">page 613</a> )	:MEASure:CLEar (see <a href="#">page 295</a> )	
:MEASure:TDELta (see <a href="#">page 614</a> )	:MARKer:XDELta (see <a href="#">page 284</a> )	
:MEASure:THResholds (see <a href="#">page 615</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 297</a> )	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see <a href="#">page 616</a> )	:MEASure:XMAX (see <a href="#">page 331</a> )	
:MEASure:TMIN (see <a href="#">page 617</a> )	:MEASure:XMIN (see <a href="#">page 332</a> )	
:MEASure:TSTArt (see <a href="#">page 618</a> )	:MARKer:X1Position (see <a href="#">page 280</a> )	
:MEASure:TSTOp (see <a href="#">page 619</a> )	:MARKer:X2Position (see <a href="#">page 282</a> )	
:MEASure:TVOLt (see <a href="#">page 620</a> )	:MEASure:TVALue (see <a href="#">page 319</a> )	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see <a href="#">page 622</a> )	:MEASure:DEFine:THResholds (see <a href="#">page 297</a> )	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see <a href="#">page 623</a> )	:MARKer:YDELta (see <a href="#">page 287</a> )	
:MEASure:VSTArt (see <a href="#">page 624</a> )	:MARKer:Y1Position (see <a href="#">page 285</a> )	
:MEASure:VSTOp (see <a href="#">page 625</a> )	:MARKer:Y2Position (see <a href="#">page 286</a> )	
:PRINt? (see <a href="#">page 626</a> )	:DISPlay:DATA? (see <a href="#">page 233</a> )	
:TIMebase:DELay (see <a href="#">page 628</a> )	:TIMebase:POSition (see <a href="#">page 396</a> ) or :TIMebase:WINDow:POSition (see <a href="#">page 402</a> )	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of delayed time base window.
:TRIGger:CAN:ACKNowledge (see <a href="#">page 629</a> )	none	

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:CAN:SIGNal:DEFinitio on (see <a href="#">page 630</a> )	none	
:TRIGger:LIN:SIGNal:DEFinitio n (see <a href="#">page 631</a> )	none	
:TRIGger:THReshold (see <a href="#">page 632</a> )	:POD<n>:THReshold (see <a href="#">page 336</a> ) or :DIGital<n>:THReshold (see <a href="#">page 229</a> )	
:TRIGger:TV:TVMode (see <a href="#">page 633</a> )	:TRIGger:TV:MODE (see <a href="#">page 500</a> )	

### Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 6000 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 237</a> )	
CHANnel:MATH	:FUNCTion:OPERation (see <a href="#">page 259</a> )	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTectio (see <a href="#">page 218</a> )	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMal.
DISPlay:INVerse	none	
DISPlay:COLumn	none	
DISPlay:GRID	none	
DISPlay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCTion:MOVE	none	
FUNCTion:PEAKs	none	

Discontinued Command	Current Command Equivalent	Comments
HARDcopy:ADDRes	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 132</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 500</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 6000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

**:CHANnel:ACTivity**

**O** (see [page 676](#))

**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 137](#)) 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 676](#))

**Command Syntax** :CHANnel:LABel <source\_text><string>

<source\_text> ::= {CHANnel1 | CHANnel2 | DIGital0,...,DIGital15}

<string> ::= quoted ASCII string

The :CHANnel:LABel command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

### NOTE

The :CHANnel:LABel command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABel command (see [page 212](#)) or :DIGital<n>:LABel command (see [page 226](#)) for the InfiniiVision 6000 Series oscilloscopes.

**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 676](#))

**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 336](#)) or :DIGital<n>:THReshold command (see [page 229](#)) for the InfiniiVision 6000 Series oscilloscopes.

**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 676](#))

**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 216](#)) 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 205

**:CHANnel<n>:INPut**

**O** (see [page 676](#))

**Command Syntax**    :CHANnel<n>:INPut <impedance>  
                          <impedance> ::= {ONEMeg | FIFTy}  
                          <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
                          <n> ::= {1 | 2} for the two channel oscilloscope models

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

**NOTE**

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see [page 210](#)) 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 676](#))

**Command Syntax** :CHANnel<n>:PMODE <pmode value>  
 <pmode value> ::= {AUTO | MANual}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODE sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

### NOTE

The :CHANnel<n>:PMODE command is an obsolete command provided for compatibility to previous oscilloscopes.

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

The :CHANnel<n>:PMODE? query returns AUT if an autosense probe is attached and MAN otherwise.

**Return Format** <pmode value><NL>  
 <pmode value> ::= {AUT | MAN}

**:DISPlay:CONNect**

**O** (see [page 676](#))

**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 239](#)) 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 239](#)

## :DISPlay:ORDer

**O** (see [page 676](#))

**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<n>:POSition"](#) on page 227


### 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"
```

Example program from the start: ["VISA COM Example in Visual Basic"](#) on page 764

## :ERASe

 (see [page 676](#))

**Command Syntax** :ERASe

The :ERASe command erases the screen.

### NOTE

The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CDISplay command (see [page 144](#)) instead.

---



## :EXternal:INPut

**O** (see [page 676](#))

**Command Syntax**    :EXternal:INPut <impedance>  
                          <impedance> ::= {ONEMeg | FIFTy}

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

### NOTE

The :EXternal:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :EXternal:IMPedance command (see [page 243](#)) instead.

**Query Syntax**    :EXternal:INPut?

The :EXternal:INPut? query returns the current input impedance setting for the external trigger.

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

- See Also**
- ["Introduction to :EXternal Trigger Commands"](#) on page 240
  - ["Introduction to :TRIGger Commands"](#) on page 405
  - [":CHANnel<n>:IMPedance"](#) on page 210

**:EXternal:PMODE**

**O** (see [page 676](#))

**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 676](#))

**Command Syntax** :FUNCTION:SOURce <value>  
 <value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

The :FUNCTION:SOURce command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCTION:OPERation command for more information about selecting an operation). The :FUNCTION:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCTION:OPERation command.

### NOTE

The :FUNCTION:SOURce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :FUNCTION:SOURce1 command (see [page 263](#)) instead.

**Query Syntax** :FUNCTION:SOURce?

The :FUNCTION:SOURce? query returns the current source for function operations.

**Return Format** <value><NL>  
 <value> ::= {CHAN<n> | ADD | SUBT | MULT}  
 <n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models  
 <n> ::= {1 | 2} for the two channel oscilloscope models

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 252
  - [":FUNCTION:OPERation"](#) on page 259

**:FUNction:VIEW**(see [page 676](#))

**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 254](#)) 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 676](#))

**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 607](#)) 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 267
  - [":HARDcopy:FORMat"](#) on page 608

**:HARDcopy:DEVIce**

**O** (see [page 676](#))

**Command Syntax** :HARDcopy:DEVIce <device>

<device> ::= {TIFF | GIF | BMP | LASerjet | EPSON | DESKjet  
| BWDeskJet | SEIKO}

The HARDcopy:DEVIce command sets the hardcopy device type.

**NOTE**

BWDeskJet option refers to the monochrome Deskjet printer.

**NOTE**

The :HARDcopy:DEVIce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FORMat command (see [page 608](#)) instead.

**Query Syntax** :HARDcopy:DEVIce?

The :HARDcopy:DEVIce? query returns the selected hardcopy device type.

**Return Format** <device><NL>

<device> ::= {TIFF | GIF | BMP | LAS | EPS | DESK | BWD | SEIK}

## :HARDcopy:FiLename

**O** (see [page 676](#))

**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 345](#)) and :RECall:FiLename command (see [page 339](#)) 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 267
  - "[:HARDcopy:FORMat](#)" on page 608

**:HARDcopy:FORMat**

**O** (see [page 676](#))

**Command Syntax** :HARDcopy:FORMat <format>

```
<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCiixy | BINary
              | PRINter0 | PRINter1}
```

The HARDcopy:FORMat command sets the hardcopy format type.

PRINter0 and PRINter1 are only valid when printers are connected to the oscilloscope's USB ports. (The first printer connected/identified is PRINter0 and the second is PRINter1.)

**NOTE**

The :HARDcopy:FORMat command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:IMAGe:FORMat (see [page 349](#)), :SAVE:WAVEform:FORMat (see [page 355](#)), and :HARDcopy:APRinter (see [page 270](#)) commands instead.

**Query Syntax** :HARDcopy:FORMat?

The :HARDcopy:FORMat? query returns the selected hardcopy format type.

**Return Format** <format><NL>

```
<format> ::= {BMP | BMP8 | PNG | CSV | ASC | BIN | PRIN0 | PRIN1}
```

**See Also** • ["Introduction to :HARDcopy Commands"](#) on [page 267](#)



## :HARDcopy:GRAYscale

**O** (see [page 676](#))

**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 274](#)) 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 267

**:HARDcopy:IGColors**

**O** (see [page 676](#))

**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 273](#)) 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 267

## :HARDcopy:PDRiver

**O** (see [page 676](#))

**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 270](#)) 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 267
  - ":HARDcopy:FORMat" on page 608

**:MEASure:LOWer**(see [page 676](#))**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 297](#)) 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 293
  - "[:MEASure:THResholds](#)" on page 615
  - "[:MEASure:UPPer](#)" on page 622

## :MEASure:SCRatch

**O** (see [page 676](#))

**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 295](#)) instead.

**:MEASure:TDELta**(see [page 676](#))**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 284](#)) instead.

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

<value> ::= time difference between start and stop markers in NR3 format

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - ["Introduction to :MEASure Commands"](#) on page 293
  - [":MARKer:X1Position"](#) on page 280
  - [":MARKer:X2Position"](#) on page 282
  - [":MARKer:XDELta"](#) on page 284
  - [":MEASure:TSTArt"](#) on page 618
  - [":MEASure:TSTOp"](#) on page 619

## :MEASure:THResholds

 (see [page 676](#))

**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 297](#)) 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 293
  - [":MEASure:LOWer"](#) on page 612
  - [":MEASure:UPPer"](#) on page 622

**:MEASure:TMAX**

**O** (see [page 676](#))

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

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:TMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

**NOTE**

The :MEASure:TMAX command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMAX command (see [page 331](#)) instead.

**Query Syntax** :MEASure:TMAX? [<source>]

The :MEASure:TMAX? query returns the horizontal axis value at which the maximum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

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

<value> ::= time at maximum in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 293
  - [":MEASure:TMIN"](#) on page 617
  - [":MEASure:XMAX"](#) on page 331
  - [":MEASure:XMIN"](#) on page 332



## :MEASure:TMIN

**O** (see [page 676](#))

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

The :MEASure:TMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

### NOTE

The :MEASure:TMIN command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMIN command (see [page 332](#)) instead.

**Query Syntax** :MEASure:TMIN? [<source>]

The :MEASure:TMIN? query returns the horizontal axis value at which the minimum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

**Return Format** <value><NL>  
 <value> ::= time at minimum in NR3 format

- See Also**
- "[Introduction to :MEASure Commands](#)" on page 293
  - "[:MEASure:TMAX](#)" on page 616
  - "[:MEASure:XMAX](#)" on page 331
  - "[:MEASure:XMIN](#)" on page 332

**:MEASure:TSTArt**

**O** (see [page 676](#))

**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 678](#)). 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 280](#)) 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 278
  - "[Introduction to :MEASure Commands](#)" on page 293
  - "[:MARKer:X1Position](#)" on page 280
  - "[:MARKer:X2Position](#)" on page 282
  - "[:MARKer:XDELta](#)" on page 284
  - "[:MEASure:TDELta](#)" on page 614
  - "[:MEASure:TSTOp](#)" on page 619

## :MEASure:TSTOp

**O** (see [page 676](#))

**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 678](#)). 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 282](#)) 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 278
  - "[Introduction to :MEASure Commands](#)" on page 293
  - "[:MARKer:X1Position](#)" on page 280
  - "[:MARKer:X2Position](#)" on page 282
  - "[:MARKer:XDELta](#)" on page 284
  - "[:MEASure:TDELta](#)" on page 614
  - "[:MEASure:TSTArt](#)" on page 618

**:MEASure:TVOLt**

**O** (see [page 676](#))

**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> ::= {DIGital0,...,DIGital15} for the MSO models

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

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

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](#)) for the InfiniiVision 6000 Series oscilloscopes.

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

```
<value> ::= time in seconds of the specified voltage crossing
           in NR3 format
```

**:MEASure:UPPer**

**O** (see [page 676](#))

**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 297](#)) 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 293
  - "[:MEASure:LOWer](#)" on page 612
  - "[:MEASure:THResholds](#)" on page 615

## :MEASure:VDELta

**O** (see [page 676](#))

**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.

$VDELta = \text{value at marker 2} - \text{value at marker 1}$

### NOTE

The :MEASure:VDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:YDELta command (see [page 287](#)) instead.

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

<value> ::= delta V value in NR1 format

- See Also**
- ["Introduction to :MARKer Commands"](#) on page 278
  - ["Introduction to :MEASure Commands"](#) on page 293
  - [":MARKer:Y1Position"](#) on page 285
  - [":MARKer:Y2Position"](#) on page 286
  - [":MARKer:YDELta"](#) on page 287
  - [":MEASure:TDELta"](#) on page 614
  - [":MEASure:TSTArt"](#) on page 618

**:MEASure:VSTArt**

**O** (see [page 676](#))

**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 678](#)). 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 285](#)) 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 278
- "[Introduction to :MEASure Commands](#)" on page 293
- "[:MARKer:Y1Position](#)" on page 285
- "[:MARKer:Y2Position](#)" on page 286
- "[:MARKer:YDELta](#)" on page 287
- "[:MARKer:X1Y1source](#)" on page 281
- "[:MEASure:SOURce](#)" on page 315
- "[:MEASure:TDELta](#)" on page 614
- "[:MEASure:TSTArt](#)" on page 618



## :MEASure:VSTOp

**O** (see [page 676](#))

**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 678](#)). 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 286](#)) 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 278
  - "[Introduction to :MEASure Commands](#)" on page 293
  - "[:MARKer:Y1Position](#)" on page 285
  - "[:MARKer:Y2Position](#)" on page 286
  - "[:MARKer:YDELta](#)" on page 287
  - "[:MARKer:X2Y2source](#)" on page 283
  - "[:MEASure:SOURce](#)" on page 315
  - "[:MEASure:TDELta](#)" on page 614
  - "[:MEASure:TSTArt](#)" on page 618

**:PRINt?**

**O** (see [page 676](#))

**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 233](#)) 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 136](#)
  - ["Introduction to :HARDcopy Commands" on page 267](#)
  - [":HARDcopy:FORMat" on page 608](#)
  - [":HARDcopy:FACTors" on page 271](#)
  - [":HARDcopy:GRAYscale" on page 609](#)
  - [":DISPlay:DATA" on page 233](#)

**:TIMEbase:DElay**

**O** (see [page 676](#))

**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 399](#)).

**NOTE**

The :TIMEbase:DElay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POsition command (see [page 396](#)) instead.

**Query Syntax** :TIMEbase:DElay?

The :TIMEbase:DElay query returns the current delay value.

**Return Format** <delay\_value><NL>

<delay\_value> ::= time from trigger to display reference in seconds in NR3 format.


**Example Code**

```
' TIMEBASE_DELAY - Sets the time base delay. This delay
' is the internal time between the trigger event and the
' onscreen delay reference point.

' Set time base delay to 0.0.
myScope.WriteString ":TIMEBASE:DELAY 0.0"
```

Example program from the start: "[VISA COM Example in Visual Basic](#)" on [page 764](#)

## :TRIGger:CAN:ACKnowledge

 (see [page 676](#))

**Command Syntax** :TRIGger:CAN:ACKnowledge <value>

<value> ::= {0 | OFF}

This command was used with the N2758A CAN trigger module for 54620/54640 Series mixed-signal oscilloscopes. The InfiniiVision 6000 Series oscilloscopes do not support the N2758A CAN trigger module.

**Query Syntax** :TRIGger:CAN:ACKnowledge?

The :TRIGger:CAN:ACKnowledge? query returns the current CAN acknowledge setting.

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

<value> ::= 0

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:CAN:TRIGger"](#) on page 425

**:TRIGger:CAN:SIGNAL:DEFinition**

**O** (see [page 676](#))

**Command Syntax** :TRIGger:CAN:SIGNAL:DEFinition <value>

<value> ::= {CANH | CANL | RX | TX | DIFFerential}

The :TRIGger:CAN:SIGNAL:DEFinition command sets the CAN signal type when :TRIGger:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signal:

- CANH – the actual CAN\_H differential bus signal.

Dominant low signals:

- CANL – the actual CAN\_L differential bus signal.
- RX – the Receive signal from the CAN bus transceiver.
- TX – the Transmit signal to the CAN bus transceiver.
- DIFFerential – the CAN differential bus signal connected to an analog source channel using a differential probe.

**NOTE**

With InfiniiVision 6000 Series oscilloscope software version 3.50 or greater, this command is available, but the only legal value is DIFF.

**Query Syntax** :TRIGger:CAN:SIGNAL:DEFinition?

The :TRIGger:CAN:SIGNAL:DEFinition? query returns the current CAN signal type.

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

<value> ::= DIFF

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:CAN:SIGNAL:BAUDrate"](#) on page 423
  - [":TRIGger:CAN:SOURce"](#) on page 424
  - [":TRIGger:CAN:TRIGger"](#) on page 425

## :TRIGger:LIN:SIGNal:DEFinition

**O** (see [page 676](#))

**Command Syntax** :TRIGger:LIN:SIGNal:DEFinition <value>

<value> ::= {LIN | RX | TX}

The :TRIGger:LIN:SIGNal:DEFinition command sets the LIN signal type. These signals can be set to:

Dominant low signals:

- LIN — the actual LIN single-end bus signal line.
- RX — the Receive signal from the LIN bus transceiver.
- TX — the Transmit signal to the LIN bus transceiver.

### NOTE

With InfiniiVision 6000 Series oscilloscope software version 3.50 or greater, this command is available, but the only legal value is LIN.

**Query Syntax** :TRIGger:LIN:SIGNal:DEFinition?

The :TRIGger:LIN:SIGNal:DEFinition? query returns the current LIN signal type.

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

<value> ::= LIN

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 405
  - [":TRIGger:MODE"](#) on page 411
  - [":TRIGger:LIN:SIGNal:BAUDrate"](#) on page 476
  - [":TRIGger:LIN:SOURce"](#) on page 477

**:TRIGger:THReshold**

**O** (see [page 676](#))

**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 336](#)), :DIGital<n>:THReshold command (see [page 229](#)), or :TRIGger[:EDGE]:LEVel command (see [page 439](#)) for the InfiniiVision 6000 Series oscilloscopes.

**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 676](#))

**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 503](#)).

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 500](#)) 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}





## 8 Error Messages

**-440, Query UNTERMINATED after indefinite response**

**-430, Query DEADLOCKED**

**-420, Query UNTERMINATED**

**-410, Query INTERRUPTED**

**-400, Query error**

**-340, Calibration failed**

**-330, Self-test failed**

**-321, Out of memory**

**-320, Storage fault**

**-315, Configuration memory lost**



**-314, Save/recall memory lost**

**-313, Calibration memory lost**

**-311, Memory error**

**-310, System error**

**-300, Device specific error**

**-278, Macro header not found**

**-277, Macro redefinition not allowed**

**-276, Macro recursion error**

**-273, Illegal macro label**

**-272, Macro execution error**

**-258, Media protected**

**-257, File name error**

**-256, File name not found**

**-255, Directory full**

**-254, Media full**

**-253, Corrupt media**

**-252, Missing media**

**-251, Missing mass storage**

**-250, Mass storage error**

**-241, Hardware missing**

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

**-240, Hardware error**

**-231, Data questionable**

**-230, Data corrupt or stale**

**-224, Illegal parameter value**

**-223, Too much data**

**-222, Data out of range**

**-221, Settings conflict**

**-220, Parameter error**

**-200, Execution error**

**-183, Invalid inside macro definition**

**-181, Invalid outside macro definition**

**-178, Expression data not allowed**

**-171, Invalid expression**

**-170, Expression error**

**-168, Block data not allowed**

**-161, Invalid block data**

**-158, String data not allowed**

- 151, Invalid string data
- 150, String data error
- 148, Character data not allowed
- 138, Suffix not allowed
- 134, Suffix too long
- 131, Invalid suffix
- 128, Numeric data not allowed
- 124, Too many digits
- 123, Exponent too large
- 121, Invalid character in number
- 120, Numeric data error
- 114, Header suffix out of range
- 113, Undefined header

**-112, Program mnemonic too long**

**-109, Missing parameter**

**-108, Parameter not allowed**

**-105, GET not allowed**

**-104, Data type error**

**-103, Invalid separator**

**-102, Syntax error**

**-101, Invalid character**

**-100, Command error**

**+10, Software Fault Occurred**

**+100, File Exists**

**+101, End-Of-File Found**

**+102, Read Error**



**+103, Write Error**

**+104, Illegal Operation**

**+105, Print Canceled**

**+106, Print Initialization Failed**

**+107, Invalid Trace File**

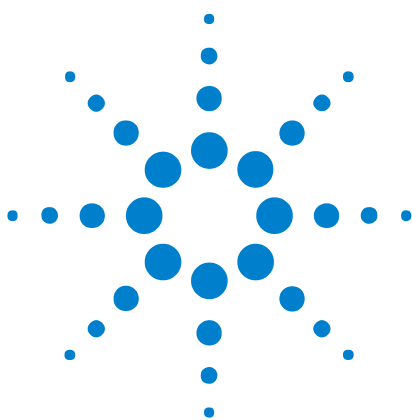
**+108, Compression Error**

**+109, No Data For Operation**

**+112, Unknown File Type**

**+113, Directory Not Supported**





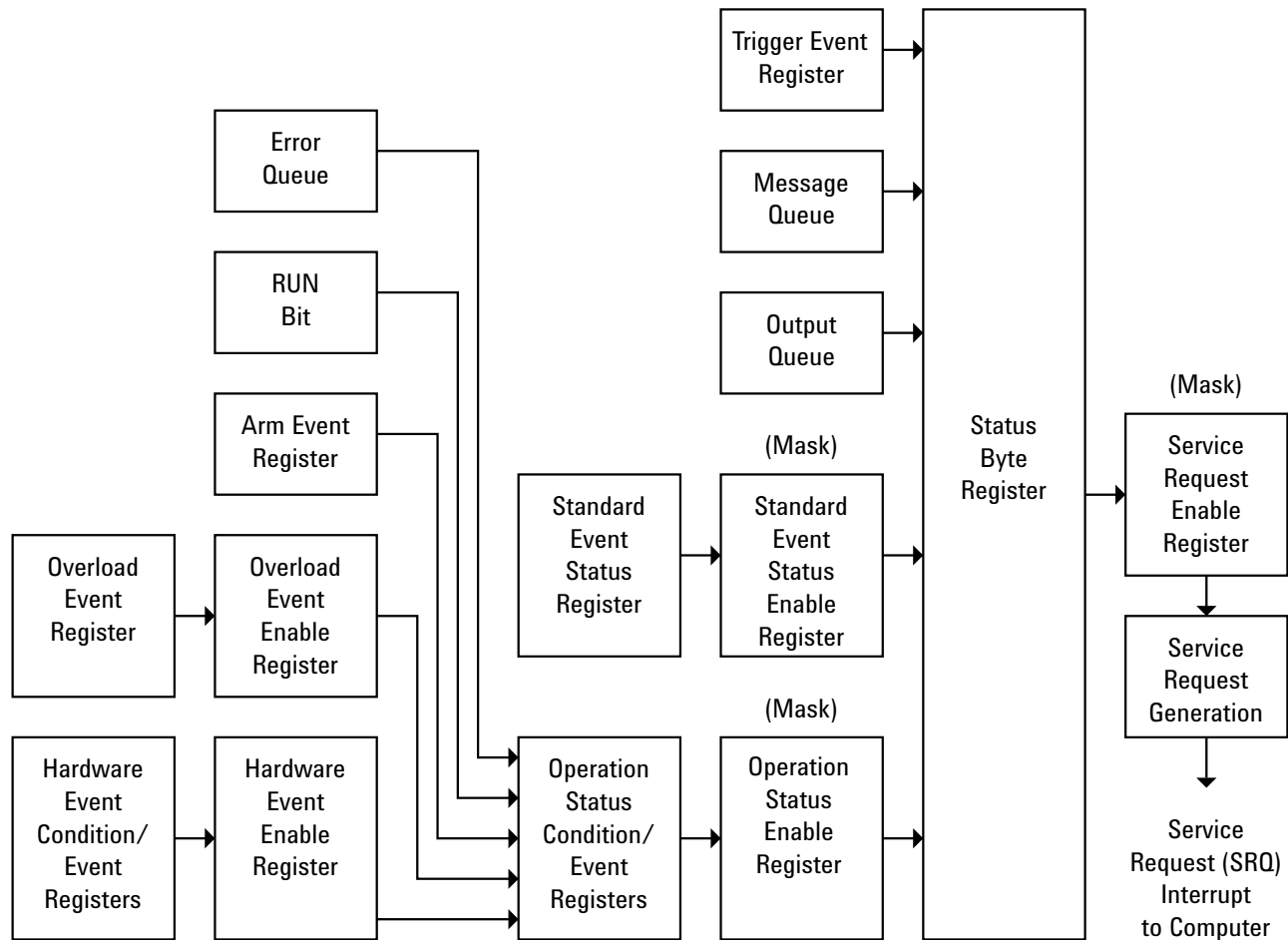
## 9 Status Reporting

Status Reporting Data Structures	645
Status Byte Register (STB)	648
Service Request Enable Register (SRE)	650
Trigger Event Register (TER)	651
Output Queue	652
Message Queue	653
(Standard) Event Status Register (ESR)	654
(Standard) Event Status Enable Register (ESE)	655
Error Queue	656
Operation Status Event Register (:OPERRegister[:EVENTt])	657
Operation Status Condition Register (:OPERRegister:CONDition)	658
Arm Event Register (AER)	659
Hardware Event Event Register (:HWERRegister[:EVENTt])	660
Hardware Event Condition Register (:HWERRegister:CONDition)	661
Clearing Registers and Queues	662
Status Reporting Decision Chart	663

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).

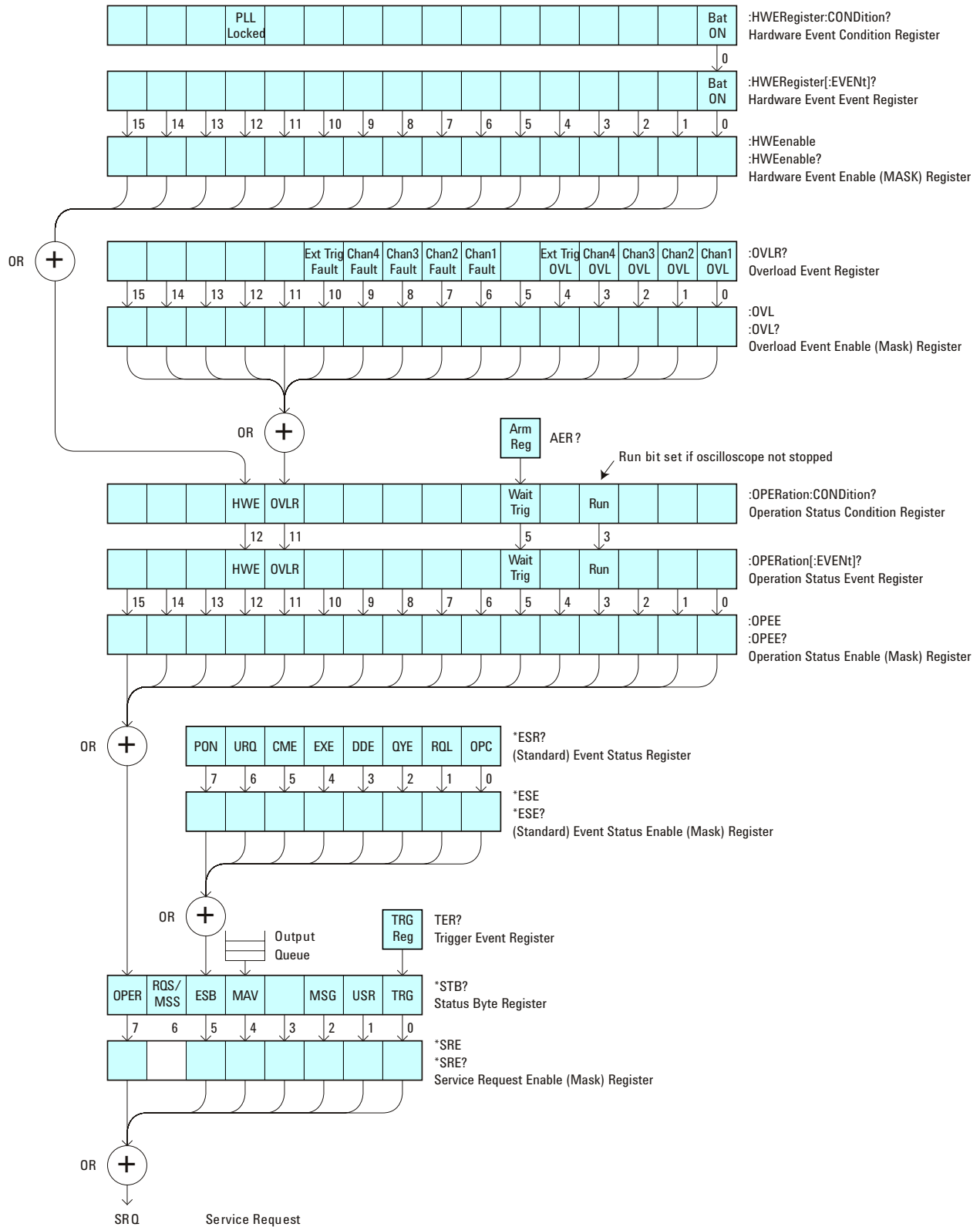
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

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.

## 9 Status Reporting



The status register bits are described in more detail in the following tables:

- "Status Byte Register (STB)" on page 129
- "Standard Event Status Register (ESR)" on page 116
- "Operation Status Condition Register" on page 156
- "Operation Status Event Register" on page 158
- "Overload Event Register (OVLRL)" on page 162
- "Hardware Event Condition Register" on page 149
- "Hardware Event Event Register" on page 151

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[:EVENTt])

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), and the OVLRL bit (bit 11).

- The RUN bit is set whenever the instrument goes from a stop state to a single or running state.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLRL bit 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[:EVENTt]? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

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

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), the OVLRL bit (bit 11), and the HWE bit (bit 12).

- The :OPERRegister:CONDition? query returns the value of the Operation Status Condition Register.
- The HWE bit (bit 12) comes from the Hardware Event Registers.
- The RUN bit is set whenever the instrument is not stopped.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLRL bit is set whenever a 50 $\Omega$  input overload occurs.

## 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.

## **Hardware Event Event Register (:HWERegister[:EVENTt])**

This register hosts the Bat On bit (bit 0).

- The Bat On bit is set whenever the instrument is operating on battery power.

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

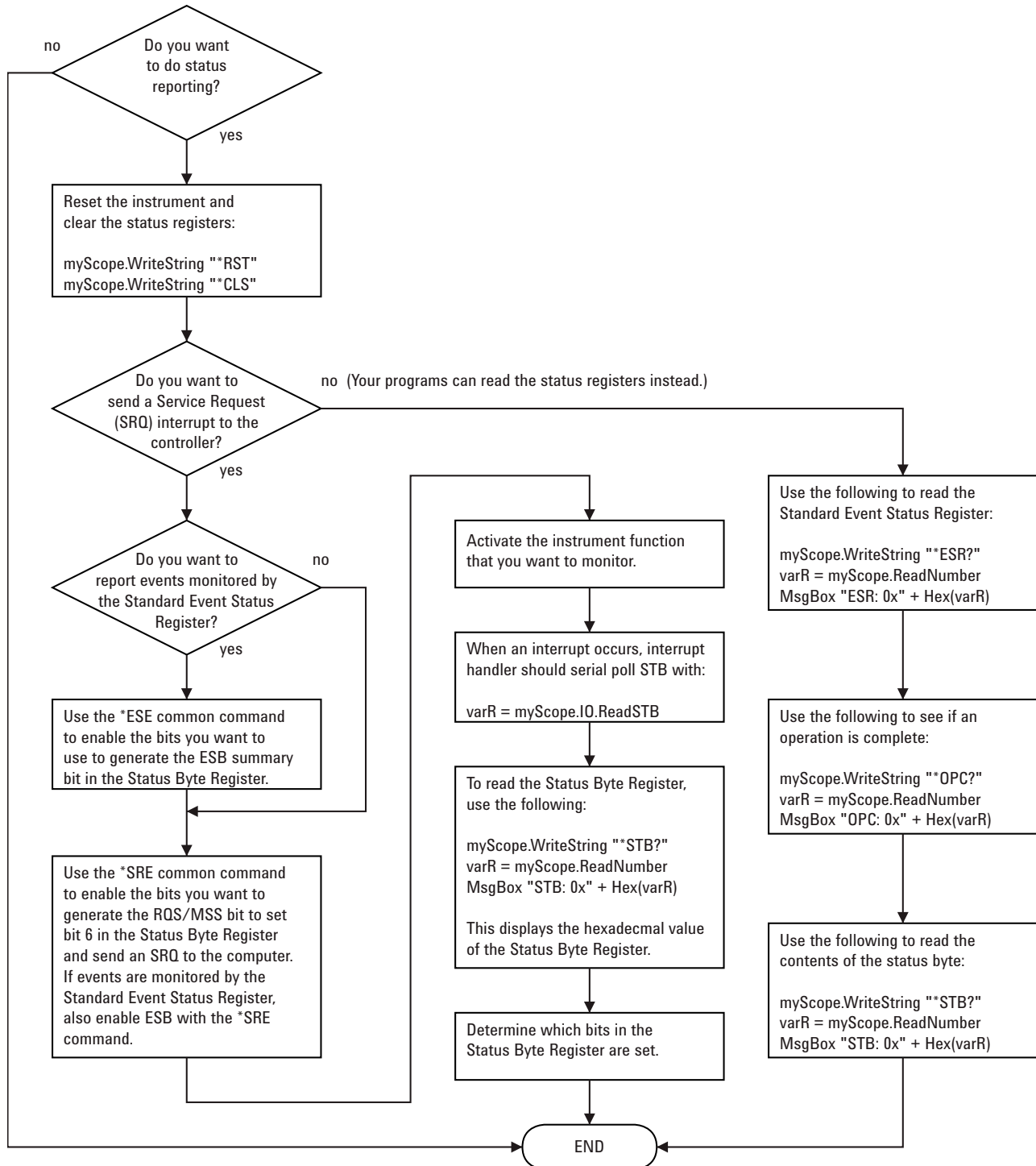
This register hosts the Bat On bit (bit 0) and the PLL LOCKED bit (bit 12).

- The :HWERegister:CONDition? query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.
- The Bat On bit is set whenever the instrument is operating on battery power.

## **Clearing Registers and Queues**

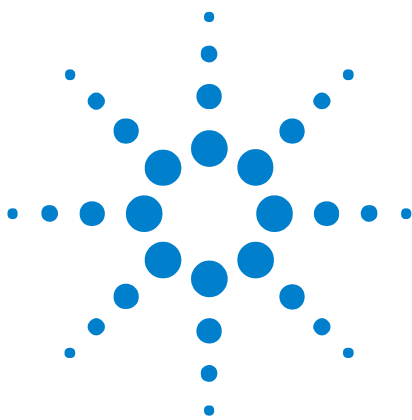
The \*CLS common command clears all event registers and all queues except the output queue. If \*CLS is sent immediately after a program message terminator, the output queue is also cleared.

## Status Reporting Decision Chart









## 10 Synchronizing Acquisitions

Synchronization in the Programming Flow	666
Blocking Synchronization	667
Polling Synchronization With Timeout	668
Synchronizing with a Single-Shot Device Under Test (DUT)	670
Synchronization with an Averaging Acquisition	672

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 666](#)).
- 2 Acquire a waveform (see [page 666](#)).
- 3 Retrieve results (see [page 666](#)).

### 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 667.	See " <a href="#">Polling Synchronization With Timeout</a> " on page 668.

### 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 667 and ["Polling Synchronization With Timeout"](#) on page 668 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 668 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"

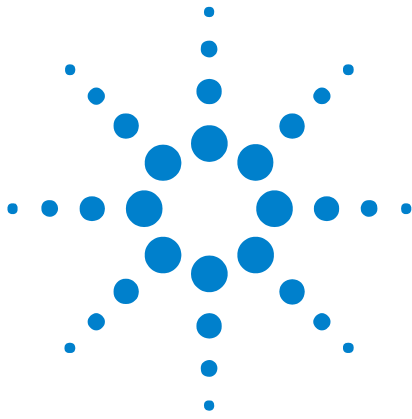
Exit Sub

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

End Sub

```





# 11

## More About Oscilloscope Commands

Command Classifications [676](#)  
Valid Command/Query Strings [677](#)  
Query Return Values [696](#)  
All Oscilloscope Commands Are Sequential [697](#)



## Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of oscilloscopes, commands are classified by the following categories:

- ["Core Commands"](#) on page 676
- ["Non-Core Commands"](#) on page 676
- ["Obsolete Commands"](#) on page 676

### **C** Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent 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 oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

### **O** Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

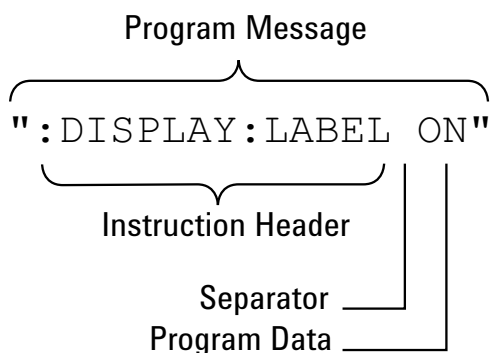
- ["Obsolete and Discontinued Commands"](#) on page 587
- As well as: ["Commands A-Z"](#) on page 559

## Valid Command/Query Strings

- ["Program Message Syntax"](#) on page 677
- ["Command Tree"](#) on page 681
- ["Duplicate Mnemonics"](#) on page 693
- ["Tree Traversal Rules and Multiple Commands"](#) on page 693

### 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 678), 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. The ["Command Tree"](#) on page 681 illustrates how all the mnemonics can be joined together to form a complete header.

":DISPlay:LABel ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, ":DISPlay:LABel?". Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- ["Simple Command Headers"](#) on page 679
- ["Compound Command Headers"](#) on page 679
- ["Common Command Headers"](#) on page 680

**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 680 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, 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 delayed.

The available mnemonics for character program data are always included with the commands's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

#### Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMEbase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

```
28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.
```



When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

## Command Tree

The command tree shows all of the commands and the relationships of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed-ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the root of the command tree.

- : (root)
  - :ACQuire (see [page 172](#))
  - :AALias (see [page 174](#))
  - :COMPlEte (see [page 175](#))
  - :COUNT (see [page 176](#))
  - :DAALias (see [page 177](#))
  - :MODE (see [page 178](#))
  - :POINTs (see [page 179](#))
  - :RSIGnal (see [page 180](#))
  - :SEGMENTed
    - :COUNT (see [page 181](#))
    - :INDEX (see [page 182](#))
  - :SRATe (see [page 184](#))
  - :TYPE (see [page 185](#))
  - :ACTivity (see [page 137](#))
  - :AER (Arm Event Register) (see [page 138](#))
  - :AUToscale (see [page 139](#))
    - :AMODE (see [page 141](#))
    - :CHANnels (see [page 142](#))
  - :BLANK (see [page 143](#))
  - :BUS<n> (see [page 187](#))
    - :BIT<m> (see [page 189](#))

- :BITS (see [page 190](#))
- :CLear (see [page 192](#))
- :DISPlay (see [page 193](#))
- :LABel (see [page 194](#))
- :MASK (see [page 195](#))
- :CALibrate (see [page 196](#))
  - :DATE (see [page 197](#))
  - :LABel (see [page 198](#))
  - :STARt (see [page 199](#))
  - :STATus (see [page 200](#))
  - :SWITch (see [page 201](#))
  - :TEMPerature (see [page 202](#))
  - :TIME (see [page 203](#))
- :CDISplay (see [page 144](#))
- :CHANnel<n> (see [page 204](#))
  - :BWLimit (see [page 207](#))
  - :COUPling (see [page 208](#))
  - :DISPlay (see [page 209](#))
  - :IMPedance (see [page 210](#))
  - :INVert (see [page 211](#))
  - :LABel (see [page 212](#))
  - :OFFSet (see [page 213](#))
  - :PROBe (see [page 214](#))
    - :ID (see [page 215](#))
    - :SKEW (see [page 216](#))
    - :STYPe (see [page 217](#))
  - :PROTection (see [page 218](#))
  - :RANGe (see [page 219](#))
  - :SCALe (see [page 220](#))
  - :UNITs (see [page 221](#))
  - :VERNier (see [page 222](#))
- :DIGital<n> (see [page 223](#))
  - :DISPlay (see [page 225](#))
  - :LABel (see [page 226](#))
  - :POSition (see [page 227](#))

- :SIZE (see [page 228](#))
- :THReshold (see [page 229](#))
- :DIGitize (see [page 145](#))
- :DISPlay (see [page 230](#))
  - :CLEar (see [page 232](#))
  - :DATA (see [page 233](#))
  - :LABel (see [page 235](#))
  - :LABList (see [page 236](#))
  - :PERsistence (see [page 237](#))
  - :SOURce (see [page 238](#))
  - :VECTors (see [page 239](#))
- :EXTernal (see [page 240](#))
  - :BWLimit (see [page 242](#))
  - :IMPedance (see [page 243](#))
  - :PROBe (see [page 244](#))
    - :ID (see [page 245](#))
    - :STYPe (see [page 246](#))
  - :PROTection (see [page 247](#))
  - :RANGe (see [page 248](#))
  - :UNITs (see [page 249](#))
- :FUNCTion (see [page 250](#))
  - :CENTer (see [page 253](#))
  - :DISPlay (see [page 254](#))
  - :GOFT
    - :OPERation (see [page 255](#))
    - :SOURce1 (see [page 256](#))
    - :SOURce2 (see [page 257](#))
  - :OFFSet (see [page 258](#))
  - :OPERation (see [page 259](#))
  - :RANGe (see [page 260](#))
  - :REFerence (see [page 261](#))
  - :SCALe (see [page 262](#))
  - :SOURce1 (see [page 263](#))
  - :SOURce2 (see [page 264](#))
  - :SPAN (see [page 265](#))

- :WINDow (see [page 266](#))
- :HARDcopy (see [page 267](#))
  - :AREA (see [page 269](#))
  - :APRinter (see [page 270](#))
  - :FACTors (see [page 271](#))
  - :FFEed (see [page 272](#))
  - :INKSaver (see [page 273](#))
  - :PALette (see [page 274](#))
  - [:PRINter]
    - :LIST (see [page 275](#))
  - [:STARt] (see [page 276](#))
- :HWEenable (Hardware Event Enable Register) (see [page 147](#))
- :HWERegister
  - :CONDition (Hardware Event Condition Register) (see [page 149](#))
  - [:EVENT] (Hardware Event Event Register) (see [page 151](#))
- :MARKer (see [page 277](#))
  - :MODE (see [page 279](#))
  - :X1Position (see [page 280](#))
  - :X1Y1source (see [page 281](#))
  - :X2Position (see [page 282](#))
  - :X2Y2source (see [page 283](#))
  - :XDELta (see [page 284](#))
  - :Y1Position (see [page 285](#))
  - :Y2Position (see [page 286](#))
  - :YDELta (see [page 287](#))
- :MEASure (see [page 288](#))
  - :CLEar (see [page 295](#))
  - :COUNter (see [page 296](#))
  - :DEFine (see [page 297](#))
  - :DELay (see [page 300](#))
  - :DUTYcycle (see [page 302](#))
  - :FALLtime (see [page 303](#))
  - :FREQuency (see [page 304](#))
  - :NWIDth (see [page 305](#))
  - :OVERshoot (see [page 306](#))

- :PERiod (see [page 308](#))
- :PHASe (see [page 309](#))
- :PREShoot (see [page 310](#))
- :PWIDth (see [page 311](#))
- :RISetime (see [page 312](#))
- :SDEViation (see [page 313](#))
- :SHOW (see [page 314](#))
- :SOURce (see [page 315](#))
- :TEDGe (see [page 317](#))
- :TVALue (see [page 319](#))
- :VAMPLitude (see [page 321](#))
- :VAVerage (see [page 322](#))
- :VBASe (see [page 323](#))
- :VMAX (see [page 324](#))
- :VMIN (see [page 325](#))
- :VPP (see [page 326](#))
- :VRATio (see [page 327](#))
- :VRMS (see [page 328](#))
- :VTIMe (see [page 329](#))
- :VTOP (see [page 330](#))
- :XMAX (see [page 331](#))
- :XMIN (see [page 332](#))
- :MERGe (see [page 153](#))
- :OPEE (Operation Status Enable Register) (see [page 154](#))
- :OPERegister
  - :CONDition (Operation Status Condition Register) (see [page 156](#))
  - [:EVENT] (Operation Status Event Register) (see [page 158](#))
- :OVLenable (Overload Event Enable Register) (see [page 160](#))
- :OVLRegister (Overload Event Register) (see [page 162](#))
- :POD<n> (see [page 333](#))
  - :DISPlay (see [page 334](#))
  - :SIZE (see [page 335](#))
  - :THReshold (see [page 336](#))
- :RECall
  - :FILename (see [page 339](#))

- :IMAGe (see [page 340](#))
  - [:START] (see [page 340](#))
- :PWD (see [page 341](#))
- :SETup (see [page 342](#))
  - [:START] (see [page 342](#))
- :RUN (see [page 165](#))
- :SAVE
  - :FILename (see [page 345](#))
  - :IMAGe (see [page 346](#))
    - [:START] (see [page 346](#))
    - :AREA (see [page 347](#))
    - :FACTors (see [page 348](#))
    - :FORMat (see [page 349](#))
    - :IGColors (see [page 350](#))
    - :PALette (see [page 351](#))
  - :PWD (see [page 352](#))
  - :SETup (see [page 353](#))
    - [:START] (see [page 353](#))
  - :WAVeform (see [page 354](#))
    - [:START] (see [page 354](#))
    - :FORMat (see [page 355](#))
    - :LENGth (see [page 356](#))
- :SBUS (see [page 357](#))
  - :BUSDoctor
    - :ADDRess (see [page 360](#))
    - :BAUDrate (see [page 361](#))
    - :CHANnel (see [page 362](#))
    - :MODE (see [page 363](#))
  - :CAN
    - :COUNT
      - :ERRor (see [page 364](#))
      - :OVERload (see [page 365](#))
      - :RESet (see [page 366](#))
      - :TOTal (see [page 367](#))
      - :UTILization (see [page 368](#))

- :DISPlay (see [page 369](#))
- :FLEXray
  - :COUNT
    - :NULL? (see [page 370](#))
    - :RESet (see [page 371](#))
    - :SYNC? (see [page 372](#))
    - :TOTal? (see [page 373](#))
- :IIC
  - :WIDTh (see [page 377](#))
- :LIN
  - :PARity (see [page 375](#))
- :MODE (see [page 376](#))
- :SPI
  - :ASIZe (see [page 374](#))
- :UART
  - :BASE (see [page 378](#))
  - :COUNT
    - :ERRor (see [page 379](#))
    - :RESet (see [page 380](#))
    - :RXFRames (see [page 381](#))
    - :TXFRames (see [page 382](#))
  - :FRAMing (see [page 383](#))
- :SERial (see [page 166](#))
- :SINGle (see [page 167](#))
- :STATus (see [page 168](#))
- :STOP (see [page 169](#))
- :SYSTem (see [page 384](#))
  - :DATE (see [page 385](#))
  - :DSP (see [page 386](#))
  - :ERRor (see [page 387](#))
  - :LOCK (see [page 388](#))
  - :PROTection
    - :LOCK (see [page 374](#))
  - :SETup (see [page 390](#))
  - :TIME (see [page 392](#))

- :TER (Trigger Event Register) (see [page 170](#))
- :TIMEbase (see [page 393](#))
  - :MODE (see [page 395](#))
  - :POSition (see [page 396](#))
  - :RANGe (see [page 397](#))
  - :REFClock (see [page 398](#))
  - :REFerence (see [page 399](#))
  - :SCALE (see [page 400](#))
  - :VERNier (see [page 401](#))
  - :WINDow
    - :POSition (see [page 402](#))
    - :RANGe (see [page 403](#))
    - :SCALE (see [page 404](#))
- :TRIGger (see [page 405](#))
  - :HFReject (see [page 409](#))
  - :HOLDoff (see [page 410](#))
  - :MODE (see [page 411](#))
  - :NREJect (see [page 412](#))
  - :PATtern (see [page 413](#))
  - :SWEep (see [page 415](#))
  - :CAN (see [page 416](#))
    - :ACKnowledge (see [page 629](#))
    - :PATtern
      - :DATA (see [page 418](#))
        - :LENGth (see [page 419](#))
      - :ID (see [page 420](#))
        - :MODE (see [page 421](#))
    - :SAMPlepoint (see [page 422](#))
    - :SIGNal
      - :BAUDrate (see [page 423](#))
      - :DEFinition (see [page 630](#))
    - :SOURce (see [page 424](#))
    - :TRIGger (see [page 425](#))
  - :DURation (see [page 427](#))
    - :GREaterthan (see [page 428](#))



- :LESSthan (see [page 429](#))
- :PATtern (see [page 430](#))
- :QUALifier (see [page 431](#))
- :RANGe (see [page 432](#))
- :EBURst (see [page 433](#))
  - :COUNT (see [page 434](#))
  - :IDLE (see [page 435](#))
  - :SLOPe (see [page 436](#))
- [:EDGE] (see [page 437](#))
  - :COUpling (see [page 438](#))
  - :LEVel (see [page 439](#))
  - :REJect (see [page 440](#))
  - :SLOPe (see [page 441](#))
  - :SOURce (see [page 442](#))
- :FLEXray (see [page 443](#))
  - :ERRor
    - :TYPE (see [page 444](#))
  - :FRAMe
    - :CCBase (see [page 446](#))
    - :CCRepetition (see [page 447](#))
    - :ID (see [page 448](#))
    - :TYPE (see [page 449](#))
  - :TIME
    - :CBASe (see [page 450](#))
    - :CREPetition (see [page 451](#))
    - :SEGMENT (see [page 452](#))
    - :SLOT (see [page 453](#))
  - :TRIGger (see [page 454](#))
- :GLITCh (see [page 455](#))
  - :GREaterthan (see [page 457](#))
  - :LESSthan (see [page 458](#))
  - :LEVel (see [page 459](#))
  - :POLarity (see [page 460](#))
  - :QUALifier (see [page 461](#))
  - :RANGe (see [page 462](#))

- :SOURce (see [page 463](#))
- :HFReject (see [page 409](#))
- :HOLDoff (see [page 410](#))
- :IIC (see [page 464](#))
  - :PATtern
    - :ADDress (see [page 465](#))
    - :DATA (see [page 466](#))
    - :DATa2 (see [page 467](#))
  - :SOURce
    - :CLOCK (see [page 468](#))
    - :DATA (see [page 469](#))
  - :TRIGger
    - :QUALifier (see [page 470](#))
    - [:TYPE] (see [page 471](#))
- :LIN (see [page 473](#))
  - :ID (see [page 474](#))
  - :SAMPlepoint (see [page 475](#))
  - :SIGNal
    - :BAUDrate (see [page 476](#))
    - :DEFinition (see [page 631](#))
  - :SOURce (see [page 477](#))
  - :STANdard (see [page 478](#))
  - :SYNCbreak (see [page 479](#))
  - :TRIGger (see [page 480](#))
- :MODE (see [page 411](#))
- :NREJect (see [page 412](#))
- :PATtern (see [page 413](#))
- :SEQuence (see [page 481](#))
  - :COUNT (see [page 482](#))
  - :EDGE (see [page 483](#))
  - :FIND (see [page 484](#))
  - :PATtern (see [page 485](#))
  - :RESet (see [page 486](#))
  - :TIMer (see [page 487](#))
  - :TRIGger (see [page 488](#))

- :SPI (see [page 489](#))
  - :CLOCK
    - :SLOPe (see [page 490](#))
    - :TImeout (see [page 491](#))
  - :FRAMing (see [page 492](#))
  - :PATtern
    - :DATA (see [page 493](#))
    - :WIDTh (see [page 494](#))
  - :SOURce
    - :CLOCK (see [page 495](#))
    - :DATA (see [page 496](#))
    - :FRAMe (see [page 497](#))
- :SWEep (see [page 415](#))
- :TV (see [page 498](#))
  - :LINE (see [page 499](#))
  - :MODE (see [page 500](#))
  - :POLarity (see [page 501](#))
  - :SOURce (see [page 502](#))
  - :STANdard (see [page 503](#))
  - :TVMode (see [page 633](#))
- :UART (see [page 504](#))
  - :BAUDrate (see [page 506](#))
  - :BITorder (see [page 507](#))
  - :BURSt (see [page 508](#))
  - :DATA (see [page 509](#))
  - :IDLE (see [page 510](#))
  - :PARity (see [page 511](#))
  - :QUALifier (see [page 513](#))
  - :POLarity (see [page 512](#))
  - :SOURce
    - :RX (see [page 514](#))
    - :TX (see [page 515](#))
  - :TYPE (see [page 516](#))
  - :WIDTh (see [page 517](#))
- :USB (see [page 518](#))

- :SOURce
  - :DMINus (see [page 519](#))
  - :DPLus (see [page 520](#))
  - :SPEed (see [page 521](#))
  - :TRIGger (see [page 522](#))
- :VIEW (see [page 171](#))
- :WAVEform (see [page 523](#))
  - :BYTeorder (see [page 531](#))
  - :COUNT (see [page 532](#))
  - :DATA (see [page 533](#))
  - :FORMat (see [page 535](#))
  - :POINTs (see [page 536](#))
    - :MODE (see [page 538](#))
  - :PREamble (see [page 540](#))
  - :SEGmented
    - :COUNT (see [page 543](#))
    - :TTAG (see [page 544](#))
  - :SOURce (see [page 545](#))
    - :SUBSource (see [page 549](#))
  - :TYPE (see [page 550](#))
  - :UNSigned (see [page 551](#))
  - :VIEW (see [page 552](#))
  - :XINCrement (see [page 553](#))
  - :XORigin (see [page 554](#))
  - :XREFerence (see [page 555](#))
  - :YINCrement (see [page 556](#))
  - :YORigin (see [page 557](#))
  - :YREFerence (see [page 558](#))

### Common Commands (IEEE 488.2)

- \*CLS (see [page 113](#))
- \*ESE (see [page 114](#))
- \*ESR (see [page 116](#))
- \*IDN (see [page 118](#))
- \*LRN (see [page 119](#))
- \*OPC (see [page 120](#))
- \*OPT (see [page 121](#))

- \*RCL (see [page 122](#))
- \*RST (see [page 123](#))
- \*SAV (see [page 126](#))
- \*SRE (see [page 127](#))
- \*STB (see [page 129](#))
- \*TRG (see [page 131](#))
- \*TST (see [page 132](#))
- \*WAI (see [page 133](#))

## Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGE may be used to change the vertical range or to change the horizontal range:

```
:CHANnel1:RANGe .4
```

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMebase:RANGe 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMebase are subsystem selectors and determine which range is being modified.

## Tree Traversal Rules and Multiple Commands

Command headers are created by traversing down the Command Tree (see [page 681](#)). 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.



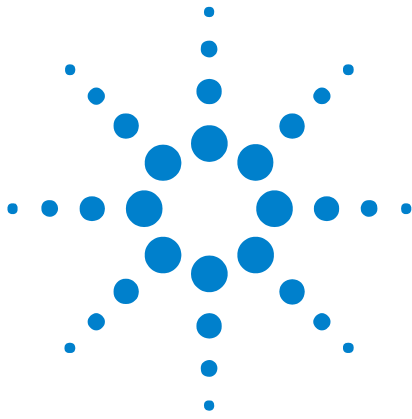
## All Oscilloscope Commands Are Sequential

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.





## 12 Programming Examples

SICL Examples [700](#)

VISA Examples [718](#)

VISA COM Examples [764](#)

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.



## SICL Examples

- "SICL Example in C" on page 700
- "SICL Example in Visual Basic" on page 709

### SICL Example in C

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual C++, Win32, Win32 Console Application project.
- 3 In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
- 4 Cut-and-paste the code that follows into a file named "example.c" in the project directory.
- 5 In Visual Studio 2005, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
- 6 Edit the program to use the SICL address of your oscilloscope.
- 7 Choose **Project > Properties....** In the Property Pages dialog, update these project settings:
  - a Under Configuration Properties, Linker, Input, add "sicl32.lib" to the Additional Dependencies field.
  - b Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
  - c Click **OK** to close the Property Pages dialog.
- 8 Add the include files and library files search paths:
  - a Choose **Tools > Options....**
  - b In the Options dialog, select **VC++ Directories** under Projects and Solutions.
  - c Show directories for **Include files**, and add the include directory (for example, Program Files\Agilent\IO Libraries Suite\include).
  - d Show directories for **Library files**, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
  - e Click **OK** to close the Options dialog.
- 9 Build and run the program.

```
/*
 * Agilent SICL Example in C
 * -----
 * This program illustrates most of the commonly-used programming
 * features of your Agilent oscilloscope.
 * This program is to be built as a WIN32 console application.
```

```

    * Edit the DEVICE_ADDRESS line to specify the address of the
    * applicable device.
    */

#include <stdio.h>           /* For printf(). */
#include "sicl.h"           /* SICL routines. */

/* #define DEVICE_ADDRESS "gpib0,7" */           /* GPIB */
/* #define DEVICE_ADDRESS "lan[a-mso6102-90541]:inst0" */ /* LAN */
#define DEVICE_ADDRESS "usb0[2391::5970::30D3090541::0]" /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT 5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE 300000

/* Function prototypes */
void initialize(void);      /* Initialize the oscilloscope. */
void extra(void);          /* Miscellaneous commands not executed,
                           shown for reference purposes. */
void capture(void);        /* Digitize data from oscilloscope. */
void analyze(void);        /* Make some measurements. */
void get_waveform(void);   /* Download waveform data from
                           oscilloscope. */
void save_waveform(void);  /* Save waveform data to a file. */
void retrieve_waveform(void); /* Load waveform data from a file. */

/* Global variables */
INST id;                  /* Device session ID. */
char buf[256] = { 0 };    /* Buffer for IDN string. */

/* Array for waveform data. */
unsigned char waveform_data[WAVE_DATA_SIZE];
double preamble[10];      /* Array for preamble. */

void main(void)
{
    /* Install a default SICL error handler that logs an error message
    * and exits. On Windows 98SE or Windows Me, view messages with
    * the SICL Message Viewer. For Windows 2000 or XP, use the Event
    * Viewer.
    */
    ionerror(I_ERROR_EXIT);

    /* Open a device session using the DEVICE_ADDRESS */
    id = iopen(DEVICE_ADDRESS);

    if (id == 0)
    {
        printf ("Oscilloscope iopen failed!\n");
    }
    else
    {
        printf ("Oscilloscope session initialized!\n");

        /* Set the I/O timeout value for this session to 5 seconds. */
        itimeout(id, TIMEOUT);
    }
}

```

```

        /* Clear the interface. */
        iclear(id);
        iremote(id);
    }

    initialize();

    /* The extras function contains miscellaneous commands that do not
     * need to be executed for the proper operation of this example.
     * The commands in the extras function are shown for reference
     * purposes only.
     */
    /* extra(); */    /* <-- Uncomment to execute the extra function */

    capture();

    analyze();

    /* Close the device session to the instrument. */
    iclose(id);
    printf ("Program execution is complete...\n");

    /* For WIN16 programs, call _siclcleanup before exiting to release
     * resources allocated by SICL for this application.  This call is
     * a no-op for WIN32 programs.
     */
    _siclcleanup();
}

/*
 * initialize
 * -----
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
    /* RESET - This command puts the oscilloscope in a known state.
     * Without this command, the oscilloscope settings are unknown.
     * This command is very important for program control.
     */
    /* Many of the following initialization commands are initialized
     * by this command.  It is not necessary to reinitialize them
     * unless you want to change the default setting.
     */
    iprintf(id, "RST\n");

    /* Write the *IDN? string and send an EOI indicator, then read
     * the response into buf.
     */
    ipromptf(id, "*IDN?\n", "%t", buf);
    printf("%s\n", buf);

    /* AUTOSCALE - This command evaluates all the input signals and
     * sets the correct conditions to display all of the active signals.

```

```

    */
    iprintf(id, ":AUTOSCALE\n");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from
     * 0.1 to 1000.
     */
    iprintf(id, ":CHAN1:PROBE 10\n");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
     * The range value is eight times the volts per division.
     */
    iprintf(id, ":CHANNEL1:RANGE 8\n");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
     * The range value is ten times the time per division.
     */
    iprintf(id, ":TIM:RANG 2e-3\n");

    /* TIME_REFERENCE - Possible values are LEFT and CENTER:
     * - LEFT sets the display reference one time division from the
     *   left.
     * - CENTER sets the display reference to the center of the screen.
     */
    iprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

    /* TRIGGER_SOURCE - Selects the channel that actually produces the
     * TV trigger. Any channel can be selected.
     */
    iprintf(id, ":TRIGGER:TV:SOURCE CHANNEL1\n");

    /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCh, PATtern,
     * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
     */
    iprintf(id, ":TRIGGER:MODE EDGE\n");

    /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
     * to either POSITIVE or NEGATIVE.
     */
    iprintf(id, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * -----
 * The commands in this function are not executed and are shown for
 * reference purposes only. To execute these commands, call this
 * function from main.
 */

void extra (void)
{
    /* RUN_STOP (not executed in this example):
     * - RUN starts the acquisition of data for the active waveform
     *   display.
     * - STOP stops the data acquisition and turns off AUTOSTORE.
     */
}

```

## 12 Programming Examples

```
iprintf(id, ":RUN\n");
iprintf(id, ":STOP\n");

/* VIEW_BLANK (not executed in this example):
 * - VIEW turns on (starts displaying) an active channel or pixel
 *   memory.
 * - BLANK turns off (stops displaying) a specified channel or
 *   pixel memory.
 */
iprintf(id, ":BLANK CHANNEL1\n");
iprintf(id, ":VIEW CHANNEL1\n");

/* TIME_MODE (not executed in this example) - Set the time base
 * mode to MAIN, DELAYED, XY or ROLL.
 */
iprintf(id, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * -----
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    iprintf(id, ":ACQUIRE:TYPE NORMAL\n");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition. The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    iprintf(id, ":ACQUIRE:COMPLETE 100\n");

    /* DIGITIZE - Used to acquire the waveform data for transfer over
     * the interface. Sending this command causes an acquisition to
     * take place with the resulting data being placed in the buffer.
     */

    /* NOTE! The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement. Keep in mind when the oscilloscope is running,
     * communication with the computer interrupts data acquisition.
     * Setting up the oscilloscope over the bus causes the data
     * buffers to be cleared and internal hardware to be reconfigured.
     * If a measurement is immediately requested there may not have
     * been enough time for the data acquisition process to collect
     * data and the results may not be accurate. An error value of
     * 9.9E+37 may be returned over the bus in this situation.
     */
    iprintf(id, ":DIGITIZE CHAN1\n");
}
```



```

/*
 * analyze
 * -----
 * In this example we will do the following:
 * - Save the system setup to a file for restoration at a later time.
 * - Save the oscilloscope display to a file which can be printed.
 * - Make single channel measurements.
 */

void analyze (void)
{
    double frequency, vpp;          /* Measurements. */
    double vdiv, off, sdiv, delay;  /* Calculated from preamble data. */
    int i;                          /* Loop counter. */
    /* Array for setup string. */
    unsigned char setup_string[SETUP_STR_SIZE];
    int setup_size;
    FILE *fp;
    unsigned char image_data[IMG_SIZE]; /* Array for image data. */
    int img_size;

    /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
     * message that contains the current state of the instrument. Its
     * format is a definite-length binary block, for example,
     * #800002204<setup string><NL>
     * where the setup string is 2204 bytes in length.
     */
    setup_size = SETUP_STR_SIZE;
    /* Query and read setup string. */
    ipromptf(id, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
    printf("Read setup string query (%d bytes).\n", setup_size);
    /* Write setup string to file. */
    fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
    setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
        fp);
    fclose (fp);
    printf("Wrote setup string (%d bytes) to file.\n", setup_size);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
     * to the oscilloscope.
     */
    /* Read setup string from file. */
    fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
    setup_size = fread (setup_string, sizeof(unsigned char),
        SETUP_STR_SIZE, fp);
    fclose (fp);
    printf("Read setup string (%d bytes) from file.\n", setup_size);
    /* Restore setup string. */
    iprintf(id, ":SYSTEM:SETUP #8%08d", setup_size);
    ifwrite(id, setup_string, setup_size, 1, &setup_size);
    printf("Restored setup string (%d bytes).\n", setup_size);

    /* IMAGE_TRANSFER - In this example we will query for the image
     * data with ":DISPLAY:DATA?" to read the data and save the data
     * to the file "image.dat" which you can then send to a printer.
     */
}

```

```

        itimeout(id, 30000);
        printf("Transferring image to c:\\scope\\data\\screen.bmp\\n");
        img_size = IMG_SIZE;
        ipromptf(id, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\\n", "%#b\\n",
            &img_size, image_data);
        printf("Read display data query (%d bytes).\\n", img_size);
        /* Write image data to file. */
        fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
        img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
        fclose (fp);
        printf("Wrote image data (%d bytes) to file.\\n", img_size);
        itimeout(id, 5000);

        /* MEASURE - The commands in the MEASURE subsystem are used to
         * make measurements on displayed waveforms.
         */

        /* Set source to measure. */
        iprintf(id, ":MEASURE:SOURCE CHANNEL1\\n");

        /* Query for frequency. */
        ipromptf(id, ":MEASURE:FREQUENCY?\\n", "%lf", &frequency);
        printf("The frequency is: %.4f kHz\\n", frequency / 1000);

        /* Query for peak to peak voltage. */
        ipromptf(id, ":MEASURE:VPP?\\n", "%lf", &vpp);
        printf("The peak to peak voltage is: %.2f V\\n", vpp);

        /* WAVEFORM_DATA - Get waveform data from oscilloscope.
         */
        get_waveform();

        /* Make some calculations from the preamble data. */
        vdiv = 32 * preamble [7];
        off = preamble [8];
        sdiv = preamble [2] * preamble [4] / 10;
        delay = (preamble [2] / 2) * preamble [4] + preamble [5];

        /* Print them out... */
        printf ("Scope Settings for Channel 1:\\n");
        printf ("Volts per Division = %f\\n", vdiv);
        printf ("Offset = %f\\n", off);
        printf ("Seconds per Division = %f\\n", sdiv);
        printf ("Delay = %f\\n", delay);

        /* print out the waveform voltage at selected points */
        for (i = 0; i < 1000; i = i + 50)
            printf ("Data Point %4d = %6.2f Volts at %10f Seconds\\n", i,
                ((float)waveform_data[i] - preamble[9]) * preamble[7] +
                preamble[8],
                ((float)i - preamble[6]) * preamble[4] + preamble[5]);

        save_waveform();          /* Save waveform data to disk. */
        retrieve_waveform();       /* Load waveform data from disk. */
    }

    /*

```

```

* get_waveform
* -----
* This function transfers the data displayed on the oscilloscope to
* the computer for storage, plotting, or further analysis.
*/

void get_waveform (void)
{
    int waveform_size;

    /* WAVEFORM_DATA - To obtain waveform data, you must specify the
     * WAVEFORM parameters for the waveform data prior to sending the
     * ":WAVEFORM:DATA?" query.
     *
     * Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
     * query provides information concerning the vertical and horizontal
     * scaling of the waveform data.
     *
     * With the preamble information you can then use the
     * ":WAVEFORM:DATA?" query and read the data block in the
     * correct format.
     */

    /* WAVE_FORMAT - Sets the data transmission mode for waveform data
     * output. This command controls how the data is formatted when
     * sent from the oscilloscope and can be set to WORD or BYTE format.
     */

    /* Set waveform format to BYTE. */
    fprintf(id, ":WAVEFORM:FORMAT BYTE\n");

    /* WAVE_POINTS - Sets the number of points to be transferred.
     * The number of time points available is returned by the
     * "ACQUIRE:POINTS?" query. This can be set to any binary
     * fraction of the total time points available.
     */
    fprintf(id, ":WAVEFORM:POINTS 1000\n");

    /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
     * settings returned in the form <preamble block><NL> where the
     * <preamble block> is:
     *
     * FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
     * TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
     * POINTS      : int32 - number of data points transferred.
     * COUNT       : int32 - 1 and is always 1.
     * XINCREMENT  : float64 - time difference between data points.
     * XORIGIN     : float64 - always the first data point in memory.
     * XREFERENCE  : int32 - specifies the data point associated
     *                   with the x-origin.
     * YINCREMENT  : float32 - voltage difference between data points.
     * YORIGIN     : float32 - value of the voltage at center screen.
     * YREFERENCE  : int32 - data point where y-origin occurs.
     */
    printf("Reading preamble\n");
    ipromptf(id, ":WAVEFORM:PREAMBLE?\n", "%10lf\n", preamble);
    /*
    printf("Preamble FORMAT: %e\n", preamble[0]);

```

```

printf("Preamble TYPE: %e\n", preamble[1]);
printf("Preamble POINTS: %e\n", preamble[2]);
printf("Preamble COUNT: %e\n", preamble[3]);
printf("Preamble XINCREMENT: %e\n", preamble[4]);
printf("Preamble XORIGIN: %e\n", preamble[5]);
printf("Preamble XREFERENCE: %e\n", preamble[6]);
printf("Preamble YINCREMENT: %e\n", preamble[7]);
printf("Preamble YORIGIN: %e\n", preamble[8]);
printf("Preamble YREFERENCE: %e\n", preamble[9]);
*/

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEFORM:SOURCE" command.
 */
iprintf(id, ":WAVEFORM:DATA?\n"); /* Query waveform data. */

/* READ_WAVE_DATA - The wave data consists of two parts: the header,
 * and the actual waveform data followed by an New Line (NL)
 * character. The query data has the following format:
 *
 *      <header><waveform data block><NL>
 *
 * Where:
 *
 *      <header> = #800002048      (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the ":WAVEFORM:POINTS"
 * command. You may then read that number of bytes from the
 * oscilloscope; then, read the following NL character to
 * terminate the query.
 */
waveform_size = WAVE_DATA_SIZE;
/* Read waveform data. */
iscanf(id, "%#b\n", &waveform_size, waveform_data);
if ( waveform_size == WAVE_DATA_SIZE )
{
    printf("Waveform data buffer full: ");
    printf("May not have received all points.\n");
}
else
{
    printf("Reading waveform data... size = %d\n", waveform_size);
}
}

/*
 * save_waveform
 * -----
 * This function saves the waveform data from the get_waveform
 * function to disk. The data is saved to a file called "wave.dat".
 */

void save_waveform(void)

```

```

{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "wb");
    /* Write preamble. */
    fwrite(preamble, sizeof(preamble[0]), 10, fp);
    /* Write actually waveform data. */
    fwrite(waveform_data, sizeof(waveform_data[0]),
           (int)preamble[2], fp);
    fclose (fp);
}

/*
 * retrieve_waveform
 * -----
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "rb");
    /* Read preamble. */
    fread (preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread (waveform_data, sizeof(waveform_data[0]),
           (int)preamble[2], fp);
    fclose (fp);
}

```

## SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Add the sicl32.bas file to your project:
  - a Choose **File>Import File....**
  - b Navigate to the header file, sicl32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the SICL address of your oscilloscope, and save the changes.
- 7 Run the program.

## 12 Programming Examples

```
'
' Agilent SICL Example in Visual Basic
' -----
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' -----

Option Explicit

Public id As Integer    ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

'
' Main Program
' -----

Sub Main()

    On Error GoTo ErrorHandler

    ' Open a device session using the SICL_ADDRESS.
    id = iopen("lan[130.29.69.12]:inst0")
    Call itimeout(id, 5000)

    ' Initialize - start from a known state.
    Initialize

    ' Capture data.
    Capture

    ' Analyze the captured waveform.
    Analyze

    ' Close the vi session and the resource manager session.
    Call iclose(id)

    Exit Sub

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Sub

'
' Initialize the oscilloscope to a known state.
```

```

' -----

Private Sub Initialize()

    On Error GoTo ErrorHandler

    ' Clear the interface.
    Call iclear(id)

    ' Get and display the device's *IDN? string.
    strQueryResult = DoQueryString("*IDN?")
    MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

    ' Clear status and load the default setup.
    DoCommand "*CLS"
    DoCommand "*RST"

    Exit Sub

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
    End

End Sub

'
' Capture the waveform.
' -----

Private Sub Capture()

    On Error GoTo ErrorHandler

    ' Use auto-scale to automatically configure oscilloscope.
    ' -----
    DoCommand ":AUToscale"

    ' Save oscilloscope configuration.
    ' -----

    Dim lngSetupStringSize As Long
    lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
    Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

    ' Output setup string to a file:
    Dim strPath As String
    strPath = "c:\scope\config\setup.dat"

    ' Open file for output.
    Dim hFile As Long
    hFile = FreeFile
    Open strPath For Binary Access Write Lock Write As hFile
    Dim lngI As Long
    For lngI = 0 To lngSetupStringSize - 1
        Put hFile, , byteArray(lngI) ' Write data.
    Next lngI
    Close hFile ' Close file.

```

```

' Or, configure the settings with individual commands:
' -----

' Set trigger mode and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
    DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
    DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
    DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
    DoQueryString(":TRIGger:EDGE:SLOPe?")

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.5"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet 1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and offset.
DoCommand ":TIMEbase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMEbase:SCALe?")

DoCommand ":TIMEbase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMEbase:POSition?")

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' -----
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile) ' Length of file.
Get hFile, , byteArray ' Read data.
Close hFile ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)

```



```

Debug.Print "Setup bytes restored: " + CStr(lngRestored)

' Acquire data.
' -----
DoCommand ":DIGitize"

Exit Sub

ErrorHandler:

MsgBox "**** Error : " + Error, vbExclamation
End

End Sub

'
' Analyze the captured waveform.
' -----

Private Sub Analyze()

On Error GoTo ErrorHandler

' Make a couple of measurements.
' -----
DoCommand ":MEASure:SOURce CHANnel1"
Debug.Print "Measure source: " + _
    DoQueryString(":MEASure:SOURce?")

DoCommand ":MEASure:VAMPlitude"
dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
MsgBox "Vertial amplitude:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

DoCommand ":MEASure:FREQuency"
dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(dblQueryResult / 1000, 4) + " kHz"

' Download the screen image.
' -----
' Get screen image.
Dim lngBlockSize As Long
lngBlockSize = _
    DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, SCReen, COLOr")
Debug.Print "Image IEEEBlock bytes: " + CStr(lngBlockSize)

' Save screen image to a file:
Dim strPath As String
strPath = "c:\scope\data\screen.png"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 10 To lngBlockSize - 1 ' Skip past 10-byte header.
    Put hFile, , byteArray(lngI) ' Write data.
Next lngI

```

```

Close hFile    ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----
Dim lngPoints As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double
Dim dblYReference As Double

' Set the waveform source.
DoCommand ":WAVEform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVEform:SOURce?")

' Get the number of waveform points:
' How do you get max depth like when saving CSV from front panel?
dblQueryResult = DoQueryNumber(":WAVEform:POINts?")
lngPoints = dblQueryResult
Debug.Print "Waveform points, channel 1: " + _
    CStr(lngPoints)

' Display the waveform settings:
dblXIncrement = DoQueryNumber(":WAVEform:XINCrement?")
Debug.Print "Waveform X increment, channel 1: " + _
    Format(dblXIncrement, "Scientific")
dblXOrigin = DoQueryNumber(":WAVEform:XORigin?")
Debug.Print "Waveform X origin, channel 1: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVEform:YINCrement?")
Debug.Print "Waveform Y increment, channel 1: " + _
    Format(dblYIncrement, "Scientific")
dblYOrigin = DoQueryNumber(":WAVEform:YORigin?")
Debug.Print "Waveform Y origin, channel 1: " + _
    Format(dblYOrigin, "Scientific")
dblYReference = DoQueryNumber(":WAVEform:YREFerence?")
Debug.Print "Waveform Y reference, channel 1: " + _
    Format(dblYReference, "Scientific")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVEform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVEform:FORMat?")
' Data in range 0 to 255.
Dim lngVSteps As Long
Dim intBytesPerData As Integer
lngVSteps = 256
intBytesPerData = 1

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVEform:DATA?")
Debug.Print "Waveform data IEEEBlock bytes: " + CStr(lngNumBytes)

```

```

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long

For lngI = 10 To lngNumBytes - 2 ' Skip past 10-byte header.
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        Format(dblXOrigin + lngI * dblXIncrement, "Scientific") + _
        ", " + _
        FormatNumber((lngDataValue - dblyReference) * dblyIncrement + _
            dblyOrigin)

Next lngI

' Close output file.
Close hFile ' Close file.
MsgBox "Waveform format BYTE data written to " + _
    "c:\scope\data\waveform_data.csv."

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Sub DoCommand(command As String)

    On Error GoTo ErrorHandler

    Call ivprintf(id, command + vbLf)
    CheckForInstrumentErrors command

Exit Sub

ErrorHandler:

MsgBox "*** Error : " + Error, vbExclamation
End

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

    On Error GoTo ErrorHandler

    ' Send command part.

```

```

    Call ivprintf(id, command + " ")
    ' Write definite-length block bytes.
    Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)
    ' retCount is now actual number of bytes written.
    CheckForInstrumentErrors command
    DoCommandIEEEBlock = retCount

Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryString(query As String) As String

    Dim actual As Long

    On Error GoTo ErrorHandler

    Dim ret_val As Integer
    Dim strResult As String * 200

    Call ivprintf(id, query + vbLf)
    Call ivscanf(id, "%200t", strResult)
    CheckForInstrumentErrors query
    DoQueryString = strResult

Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Function DoQueryNumber(query As String) As Double

    On Error GoTo ErrorHandler

    Dim dblResult As Double

    Call ivprintf(id, query + vbLf)
    Call ivscanf(id, "%lf" + vbLf, dblResult)
    CheckForInstrumentErrors query
    DoQueryNumber = dblResult

Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

```

```

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

    On Error GoTo ErrorHandler

    ' Send query.
    Call ivprintf(id, query + vbCrLf)
    ' Read definite-length block bytes.
    Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)
    ' retCount is now actual number of bytes returned by read.
    CheckForInstrumentErrors query
    DoQueryIEEEBlock_Bytes = retCount

Exit Function

ErrorHandler:

    MsgBox "*** Error : " + Error, vbExclamation
End

End Function

Private Sub CheckForInstrumentErrors(strCmdOrQuery As String)

    On Error GoTo ErrorHandler

    Dim strErrVal As String * 200
    Dim strOut As String

    Do
        Call ivprintf(id, "SYSTem:ERRor?" + vbCrLf) ' Request error message.
        Call ivscanf(id, "%200t", strErrVal) ' Read: Errno,"Error String".
        If Val(strErrVal) <> 0 Then
            strOut = strOut + "INST Error: " + RTrim(strErrVal) + vbCrLf
        End If
    Loop While Val(strErrVal) <> 0 ' End if find: 0,"No Error".

    If Not strOut = "" Then
        MsgBox strOut, vbExclamation, "INST Error Messages, " + _
            strCmdOrQuery
        Call iflush(id, I_BUF_DISCARD_READ Or I_BUF_DISCARD_WRITE)
    End If

Exit Sub

ErrorHandler:

    MsgBox "**** Error: " + Error, vbExclamation

End Sub

```

## VISA Examples

- "VISA Example in C" on page 718
- "VISA Example in Visual Basic" on page 727
- "VISA Example in C#" on page 737
- "VISA Example in Visual Basic .NET" on page 751

### VISA Example in C

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual C++, Win32, Win32 Console Application project.
- 3 In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
- 4 Cut-and-paste the code that follows into a file named "example.c" in the project directory.
- 5 In Visual Studio 2005, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
- 6 Edit the program to use the VISA address of your oscilloscope.
- 7 Choose **Project > Properties....** In the Property Pages dialog, update these project settings:
  - a Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.
  - b Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
  - c Click **OK** to close the Property Pages dialog.
- 8 Add the include files and library files search paths:
  - a Choose **Tools > Options....**
  - b In the Options dialog, select **VC++ Directories** under Projects and Solutions.
  - c Show directories for **Include files**, and add the include directory (for example, Program Files\VISA\winnt\include).
  - d Show directories for **Library files**, and add the library files directory (for example, Program Files\VISA\winnt\lib\msc).
  - e Click **OK** to close the Options dialog.
- 9 Build and run the program.

```
/*
 * Agilent VISA Example in C
 * -----
```

```

* This program illustrates most of the commonly-used programming
* features of your Agilent oscilloscope.
* This program is to be built as a WIN32 console application.
* Edit the RESOURCE line to specify the address of the
* applicable device.
*/

#include <stdio.h>           /* For printf(). */
#include <visa.h>            /* Agilent VISA routines. */

/* GPIB */
/* #define RESOURCE "GPIB0::7::INSTR" */

/* LAN */
/* #define RESOURCE "TCPIP0::a-mso6102-90541::inst0::INSTR" */

/* USB */
#define RESOURCE "USB0::2391::5970::30D3090541::0::INSTR"

#define WAVE_DATA_SIZE 5000
#define TIMEOUT        5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE        300000

/* Function prototypes */
void initialize(void);      /* Initialize the oscilloscope. */
void extra(void);          /* Miscellaneous commands not executed,
                           shown for reference purposes. */
void capture(void);        /* Digitize data from oscilloscope. */
void analyze(void);        /* Make some measurements. */
void get_waveform(void);   /* Download waveform data from
                           oscilloscope. */
void save_waveform(void);  /* Save waveform data to a file. */
void retrieve_waveform(void); /* Load waveform data from a file. */

/* Global variables */
ViSession defaultRM, vi;   /* Device session ID. */
char buf[256] = { 0 };    /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE]; /* Array for waveform
                                                data. */
double preamble[10];      /* Array for preamble. */

void main(void)
{
    /* Open session. */
    viOpenDefaultRM(&defaultRM);
    viOpen(defaultRM, RESOURCE, VI_NULL, VI_NULL, &vi);
    printf ("Oscilloscope session initialized!\n");

    /* Clear the interface. */
    viClear(vi);

    initialize();

    /* The extras function contains miscellaneous commands that do not
     * need to be executed for the proper operation of this example.

```

```

    * The commands in the extras function are shown for reference
    * purposes only.
    */
    /* extra(); */    /* <-- Uncomment to execute the extra function */

    capture();

    analyze();

    /* Close session */
    viClose(vi);
    viClose(defaultRM);
    printf ("Program execution is complete...\n");
}

/*
 * initialize
 * -----
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
    /* RESET - This command puts the oscilloscope in a known state.
     * Without this command, the oscilloscope settings are unknown.
     * This command is very important for program control.
     */
    /* Many of the following initialization commands are initialized
     * by this command. It is not necessary to reinitialize them
     * unless you want to change the default setting.
     */
    viPrintf(vi, "*RST\n");

    /* Write the *IDN? string and send an EOI indicator, then read
     * the response into buf.
     */
    viQueryf(vi, "*IDN?\n", "%t", buf);
    printf("%s\n", buf);
    /*

    /* AUTOSCALE - This command evaluates all the input signals and
     * sets the correct conditions to display all of the active signals.
     */
    viPrintf(vi, ":AUTOSCALE\n");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from
     * 0.1 to 1000.
     */
    viPrintf(vi, ":CHAN1:PROBE 10\n");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
     * The range value is eight times the volts per division.
     */
    viPrintf(vi, ":CHANNEL1:RANGE 8\n");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.

```



```

    * The range value is ten times the time per division.
    */
viPrintf(vi, ":TIM:RANG 2e-3\n");

/* TIME_REFERENCE - Possible values are LEFT and CENTER:
 * - LEFT sets the display reference one time division from the
 *   left.
 * - CENTER sets the display reference to the center of the screen.
 */
viPrintf(vi, ":TIMEBASE:REFERENCE CENTER\n");

/* TRIGGER_SOURCE - Selects the channel that actually produces the
 * TV trigger. Any channel can be selected.
 */
viPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1\n");

/* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCH, PATTERN,
 * CAN, DURATION, IIC, LIN, SEQUENCE, SPI, TV, or USB.
 */
viPrintf(vi, ":TRIGGER:MODE EDGE\n");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
 * to either POSITIVE or NEGATIVE.
 */
viPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * -----
 * The commands in this function are not executed and are shown for
 * reference purposes only. To execute these commands, call this
 * function from main.
 */

void extra (void)
{
    /* RUN_STOP (not executed in this example):
     * - RUN starts the acquisition of data for the active waveform
     *   display.
     * - STOP stops the data acquisition and turns off AUTOSTORE.
     */
    viPrintf(vi, ":RUN\n");
    viPrintf(vi, ":STOP\n");

    /* VIEW_BLANK (not executed in this example):
     * - VIEW turns on (starts displaying) an active channel or pixel
     *   memory.
     * - BLANK turns off (stops displaying) a specified channel or
     *   pixel memory.
     */
    viPrintf(vi, ":BLANK CHANNEL1\n");
    viPrintf(vi, ":VIEW CHANNEL1\n");

    /* TIME_MODE (not executed in this example) - Set the time base
     * mode to MAIN, DELAYED, XY or ROLL.
     */

```

```

        viPrintf(vi, ":TIMEBASE:MODE MAIN\n");
    }

    /*
    * capture
    * -----
    * This function prepares the scope for data acquisition and then
    * uses the DIGITIZE MACRO to capture some data.
    */

void capture (void)
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
    viPrintf(vi, ":ACQUIRE:TYPE NORMAL\n");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition. The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
    viPrintf(vi, ":ACQUIRE:COMPLETE 100\n");

    /* DIGITIZE - Used to acquire the waveform data for transfer over
    * the interface. Sending this command causes an acquisition to
    * take place with the resulting data being placed in the buffer.
    */

    /* NOTE! The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement. Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate. An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
    viPrintf(vi, ":DIGITIZE CHAN1\n");
}

/*
* analyze
* -----
* In this example we will do the following:
* - Save the system setup to a file for restoration at a later time.
* - Save the oscilloscope display to a file which can be printed.
* - Make single channel measurements.
*/

void analyze (void)
{
    double frequency, vpp;          /* Measurements. */
    double vdiv, off, sdiv, delay; /* Values calculated from preamble
                                     data. */

```

```

int i;                                /* Loop counter. */
unsigned char setup_string[SETUP_STR_SIZE]; /* Array for setup
                                           string. */

int setup_size;
FILE *fp;
unsigned char image_data[IMG_SIZE]; /* Array for image data. */
int img_size;

/* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
 * message that contains the current state of the instrument. Its
 * format is a definite-length binary block, for example,
 * #800002204<setup string><NL>
 * where the setup string is 2204 bytes in length.
 */
setup_size = SETUP_STR_SIZE;
/* Query and read setup string. */
viQueryf(vi, ":SYSTEM:SETUP?\n", "%b\n", &setup_size, setup_string);
printf("Read setup string query (%d bytes).\n", setup_size);
/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
                    fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to file.\n", setup_size);

/* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
 * to the oscilloscope.
 */
/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
setup_size = fread (setup_string, sizeof(unsigned char),
                    SETUP_STR_SIZE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file.\n", setup_size);
/* Restore setup string. */
viPrintf(vi, ":SYSTEM:SETUP #8%08d", setup_size);
viBufWrite(vi, setup_string, setup_size, &setup_size);
viPrintf(vi, "\n");
printf("Restored setup string (%d bytes).\n", setup_size);

/* IMAGE_TRANSFER - In this example we will query for the image
 * data with ":DISPLAY:DATA?" to read the data and save the data
 * to the file "image.dat" which you can then send to a printer.
 */
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 30000);
printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
img_size = IMG_SIZE;
viQueryf(vi, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%b\n",
        &img_size, image_data);
printf("Read display data query (%d bytes).\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 5000);

```

```

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

/* Set source to measure. */
viPrintf(vi, ":MEASURE:SOURCE CHANNEL1\n");

/* Query for frequency. */
viQueryf(vi, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
printf("The frequency is: %.4f kHz\n", frequency / 1000);

/* Query for peak to peak voltage. */
viQueryf(vi, ":MEASURE:VPP?\n", "%lf", &vpp);
printf("The peak to peak voltage is: %.2f V\n", vpp);

/* WAVEFORM_DATA - Get waveform data from oscilloscope.
 */
get_waveform();

/* Make some calculations from the preamble data. */
vdiv = 32 * preamble [7];
off = preamble [8];
sdiv = preamble [2] * preamble [4] / 10;
delay = (preamble [2] / 2) * preamble [4] + preamble [5];

/* Print them out... */
printf ("Scope Settings for Channel 1:\n");
printf ("Volts per Division = %f\n", vdiv);
printf ("Offset = %f\n", off);
printf ("Seconds per Division = %f\n", sdiv);
printf ("Delay = %f\n", delay);

/* print out the waveform voltage at selected points */
for (i = 0; i < 1000; i = i + 50)
    printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
        ((float)waveform_data[i] - preamble[9]) * preamble[7] +
        preamble[8],
        ((float)i - preamble[6]) * preamble[4] + preamble[5]);

save_waveform();          /* Save waveform data to disk. */
retrieve_waveform();      /* Load waveform data from disk. */
}

/*
 * get_waveform
 * -----
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
    int waveform_size;

    /* WAVEFORM_DATA - To obtain waveform data, you must specify the
     * WAVEFORM parameters for the waveform data prior to sending the
     * ":WAVEFORM:DATA?" query.

```

```

*
* Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
* query provides information concerning the vertical and horizontal
* scaling of the waveform data.
*
* With the preamble information you can then use the
* ":WAVEFORM:DATA?" query and read the data block in the
* correct format.
*/

/* WAVE_FORMAT - Sets the data transmission mode for waveform data
* output. This command controls how the data is formatted when
* sent from the oscilloscope and can be set to WORD or BYTE format.
*/

/* Set waveform format to BYTE. */
viPrintf(vi, ":WAVEFORM:FORMAT BYTE\n");

/* WAVE_POINTS - Sets the number of points to be transferred.
* The number of time points available is returned by the
* "ACQUIRE:POINTS?" query. This can be set to any binary
* fraction of the total time points available.
*/
viPrintf(vi, ":WAVEFORM:POINTS 1000\n");

/* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
* settings returned in the form <preamble block><NL> where the
* <preamble block> is:
*   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
*   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
*   POINTS      : int32 - number of data points transferred.
*   COUNT       : int32 - 1 and is always 1.
*   XINCREMENT  : float64 - time difference between data points.
*   XORIGIN     : float64 - always the first data point in memory.
*   XREFERENCE  : int32 - specifies the data point associated
*                     with the x-origin.
*   YINCREMENT  : float32 - voltage difference between data points.
*   YORIGIN     : float32 - value of the voltage at center screen.
*   YREFERENCE  : int32 - data point where y-origin occurs.
*/
printf("Reading preamble\n");
viQueryf(vi, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
/*
printf("Preamble FORMAT: %e\n", preamble[0]);
printf("Preamble TYPE: %e\n", preamble[1]);
printf("Preamble POINTS: %e\n", preamble[2]);
printf("Preamble COUNT: %e\n", preamble[3]);
printf("Preamble XINCREMENT: %e\n", preamble[4]);
printf("Preamble XORIGIN: %e\n", preamble[5]);
printf("Preamble XREFERENCE: %e\n", preamble[6]);
printf("Preamble YINCREMENT: %e\n", preamble[7]);
printf("Preamble YORIGIN: %e\n", preamble[8]);
printf("Preamble YREFERENCE: %e\n", preamble[9]);
*/

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
* over the interface that is stored in a buffer previously

```

```

    * specified with the ":WAVEFORM:SOURCE" command.
    */
viPrintf(vi, ":WAVEFORM:DATA?\n");    /* Query waveform data. */

/* READ_WAVE_DATA - The wave data consists of two parts: the header,
 * and the actual waveform data followed by an New Line (NL)
 * character. The query data has the following format:
 *
 *      <header><waveform data block><NL>
 *
 * Where:
 *
 *      <header> = #800002048      (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the ":WAVEFORM:POINTS"
 * command. You may then read that number of bytes from the
 * oscilloscope; then, read the following NL character to
 * terminate the query.
 */
waveform_size = WAVE_DATA_SIZE;
/* Read waveform data. */
viScanf(vi, "%#b\n", &waveform_size, waveform_data);
if ( waveform_size == WAVE_DATA_SIZE )
{
    printf("Waveform data buffer full: ");
    printf("May not have received all points.\n");
}
else
{
    printf("Reading waveform data... size = %d\n", waveform_size);
}
}

/*
 * save_waveform
 * -----
 * This function saves the waveform data from the get_waveform
 * function to disk. The data is saved to a file called "wave.dat".
 */

void save_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "wb");
    /* Write preamble. */
    fwrite(preamble, sizeof(preamble[0]), 10, fp);
    /* Write actually waveform data. */
    fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
           fp);
    fclose(fp);
}

/*

```

```

* retrieve_waveform
* -----
* This function retrieves previously saved waveform data from a
* file called "wave.dat".
*/

void retrieve_waveform(void)
{
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "rb");
    /* Read preamble. */
    fread(preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
        fp);
    fclose(fp);
}

```

## VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Add the visa32.bas file to your project:
  - a Choose **File>Import File...**
  - b Navigate to the header file, visa32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, and click **Open**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```

'
' Agilent VISA Example in Visual Basic
' -----
' This program illustrates most of the commonly-used programming
' features of your Agilent oscilloscope.
' -----

Option Explicit

Public err As Long    ' Error returned by VISA function calls.
Public drm As Long    ' Session to Default Resource Manager.
Public vi As Long     ' Session to instrument.

```

```

' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const DblArraySize = 20
Public Const ByteArraySize = 5000000
Public retCount As Long
Public dblArray(DblArraySize) As Double
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

'
' MAIN PROGRAM
' -----
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form. Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -----

Sub Main()

' Open the default resource manager session.
err = viOpenDefaultRM(drm)

' Open the session to the resource.
' The "GPIB0" parameter is the VISA Interface name to
' an GPIB instrument as defined in:
' Start->Programs->Agilent IO Libraries->IO Config
' Change this name to whatever you have defined for your
' VISA Interface.
' "GPIB0::7::INSTR" is the address string for the device -
' this address will be the same as seen in:
' Start->Programs->Agilent IO Libraries->VISA Assistant
' (after the VISA Interface Name is defined in IO Config).

' err = viOpen(drm, "GPIB0::7::INSTR", 0, 0, vi)
' err = viOpen(drm, "TCPIP0::a-mso6102-90541::inst0::INSTR", 0, 0, vi)
err = viOpen(drm, _
    "USB0::2391::5970::30D3090541::0::INSTR", 0, 60000, vi)

' Initialize - Initialization will start the program with the
' oscilloscope in a known state.
Initialize

' Capture - After initialization, you must make waveform data
' available to analyze. To do this, capture the data using the

```



```

' DIGITIZE command.
Capture

' Analyze - Once the waveform has been captured, it can be analyzed.
' There are many parts of a waveform to analyze. This example shows
' some of the possible ways to analyze various parts of a waveform.
Analyze

' Close the vi session and the resource manager session.
err = viClose(vi)
err = viClose(drm)

End Sub

'
' Initialize
' -----
' Initialize will start the program with the oscilloscope in a known
' state. This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
' - Oscilloscope
' - Channel 1 range
' - Display Grid
' - Timebase reference, range, and delay
' - Trigger mode and type
'
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -----

Private Sub Initialize()

' Clear the interface.
err = viClear(vi)

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

' Reset the oscilloscope to the defaults.
err = viVPrintf(vi, "*RST" + vbLf, 0)

' IDN - Ask for the device's *IDN string.
err = viVPrintf(vi, "*IDN?" + vbLf, 0)
err = viVScanf(vi, "%t", strQueryResult) ' Read the results as a
                                         ' string.

' Display results.
MsgBox "Result is: " + strQueryResult, vbOKOnly, "*IDN? Result"

' AUTOSCALE - This command evaluates all the input signals and sets
' the correct conditions to display all of the active signals.
err = viVPrintf(vi, ":AUTOSCALE" + vbLf, 0) ' Same as pressing
                                           ' the Autoscale key.

```

```
' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
' channel. The probe attenuation factor may be set from 0.1 to 1000.

' Set Probe to 10:1.
err = viVPrintf(vi, ":CHAN1:PROBE 10" + vbLf, 0)

' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
' range value is 8 times the volts per division.

' Set the vertical range to 8 volts.
err = viVPrintf(vi, ":CHANNEL1:RANGE 8" + vbLf, 0)

' TIME_RANGE - Sets the full scale horizontal time in seconds. The
' range value is 10 times the time per division.

' Set the time range to 0.002 seconds.
err = viVPrintf(vi, ":TIM:RANG 2e-3" + vbLf, 0)

' TIME_REFERENCE - Possible values are LEFT and CENTER.
' - LEFT sets the display reference on time division from the left.
' - CENTER sets the display reference to the center of the screen.

' Set reference to center.
err = viVPrintf(vi, ":TIMEBASE:REFERENCE CENTER" + vbLf, 0)

' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
' TV trigger. Any channel can be selected.
err = viVPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1" + vbLf, 0)

' TRIGGER_MODE - Set the trigger mode to EDGE, GLITCh, PATtern, CAN,
' DURATION, IIC, LIN, SEquence, SPI, TV, or USB.

' Set the trigger mode to EDGE.
err = viVPrintf(vi, ":TRIGGER:MODE EDGE" + vbLf, 0)

' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
err = viVPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE" + vbLf, 0)

' The following commands are not executed and are shown for reference
' purposes only. To execute these commands, uncomment them.

' RUN_STOP - (not executed in this example)
' - RUN starts the acquisition of data for the active waveform
' display.
' - STOP stops the data acquisition and turns off AUTOSTORE.

' Start data acquisition.
' err = viVPrintf(vi, ":RUN" + vbLf, 0)

' Stop the data acquisition.
' err = viVPrintf(vi, ":STOP" + vbLf, 0)

' VIEW_BLANK - (not executed in this example)
' - VIEW turns on (starts displaying) a channel or pixel memory.
```

```

' - BLANK turns off (stops displaying) a channel or pixel memory.

' Turn channel 1 off.
' err = viVPrintf(vi, ":BLANK CHANNEL1" + vbLf, 0)

' Turn channel 1 on.
' err = viVPrintf(vi, ":VIEW CHANNEL1" + vbLf, 0)

' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
' err = viVPrintf(vi, ":TIMEBASE:MODE MAIN" + vbLf, 0)

End Sub

'
' Capture
' -----
' We will capture the waveform using the digitize command.
' -----

Private Sub Capture()

' ACQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
' PEAK, or AVERAGE.
err = viVPrintf(vi, ":ACQUIRE:TYPE NORMAL" + vbLf, 0)

' ACQUIRE_COMPLETE - Specifies the minimum completion criteria for
' an acquisition. The parameter determines the percentage of time
' buckets needed to be "full" before an acquisition is considered
' to be complete.
err = viVPrintf(vi, ":ACQUIRE:COMPLETE 100" + vbLf, 0)

' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface. Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.
'
' NOTE! The DIGITIZE command is highly recommended for triggering
' modes other than SINGLE. This ensures that sufficient data is
' available for measurement. If DIGITIZE is used with single mode,
' the completion criteria may never be met. The number of points
' gathered in Single mode is related to the sweep speed, memory
' depth, and maximum sample rate. For example, take an oscilloscope
' with a 1000-point memory, a sweep speed of 10 us/div (100 us
' total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s. Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured. If a
' measurement is immediately requested, there may have not been

```

```

' enough time for the data acquisition process to collect data,
' and the results may not be accurate. An error value of 9.9E+37
' may be returned over the bus in this situation.
'
err = viVPrintf(vi, ":DIGITIZE CHAN1" + vbLf, 0)

End Sub

'
' Analyze
' -----
' In analyze, we will do the following:
' - Save the system setup to a file and restore it.
' - Save the waveform data to a file on the computer.
' - Make single channel measurements.
' - Save the oscilloscope display to a file that can be sent to a
'   printer.
' -----

Private Sub Analyze()

' Set up arrays for multiple parameter query returning an array
' with viVScanf/viVQueryf. Set retCount to the maximum number
' of elements that the array can hold.
paramsArray(0) = VarPtr(retCount)
paramsArray(1) = VarPtr(byteArray(0))

' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
' message that contains the current state of the instrument. Its
' format is a definite-length binary block, for example,
'   #800002204<setup string><NL>
' where the setup string is 2204 bytes in length.
Dim lngSetupStringSize As Long
err = viVPrintf(vi, ":SYSTEM:SETUP?" + vbLf, 0)
retCount = ByteArraySize

' Unsigned integer bytes.
err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
lngSetupStringSize = retCount

' Output setup string to a file:
Dim strPath As String
Dim lngI As Long
strPath = "c:\scope\config\setup.dat"
Close #1 ' If #1 is open, close it.

' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
For lngI = 0 To lngSetupStringSize - 1
    Put #1, , byteArray(lngI) ' Write data.
Next lngI
Close #1 ' Close file.

' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
err = viVPrintf(vi, ":DISPLAY:DATA? BMP, SCREEN, COLOR" + vbLf, 0)
retCount = ByteArraySize

```

```

' Unsigned integer bytes.
err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1 ' If #1 is open, close it.

' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
For lngI = 0 To retCount - 1
    Put #1, , byteArray(lngI) ' Write data.
Next lngI
Close #1 ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1 ' Open file for input.
Get #1, , byteArray ' Read data.
Close #1 ' Close file.
' Write learn string back to oscilloscope using ":SYSTEM:SETUP"
' command:
retCount = lngSetupStringSize
err = viVPrintf(vi, ":SYSTEM:SETUP %#b" + vbLf, paramsArray(0))

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure
err = viVPrintf(vi, ":MEASURE:SOURCE CHANNEL1" + vbLf, 0)

' Query for frequency.
err = viVPrintf(vi, ":MEASURE:FREQUENCY?" + vbLf, 0)
' Read frequency.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(dblQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
err = viVPrintf(vi, ":MEASURE:DUTYCYCLE?" + vbLf, 0)
' Read duty cycle.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(dblQueryResult, 3) + "%"

' Query for risetime.
err = viVPrintf(vi, ":MEASURE:RISETIME?" + vbLf, 0)
' Read risetime.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Risetime:" + vbCrLf + _
    FormatNumber(dblQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
err = viVPrintf(vi, ":MEASURE:VPP?" + vbLf, 0)

```

```

' Read VPP.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Peak to peak voltage:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

' Query for Vmax.
err = viVPrintf(vi, ":MEASURE:VMAX?" + vbLf, 0)
' Read Vmax.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Maximum voltage:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query. Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
err = viVPrintf(vi, ":WAVEFORM:SOURCE CHAN1" + vbLf, 0)

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
err = viVPrintf(vi, ":WAVEFORM:POINTS 1000" + vbLf, 0)

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output. This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
err = viVPrintf(vi, ":WAVEFORM:FORMAT WORD" + vbLf, 0)
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
err = viVPrintf(vi, ":WAVEFORM:FORMAT BYTE" + vbLf, 0)
lngVSteps = 256
intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'   POINTS      : int32 - number of data points transferred.
'   COUNT       : int32 - 1 and is always 1.
'   XINCREMENT   : float64 - time difference between data points.
'   XORIGIN      : float64 - always the first data point in memory.
'   XREFERENCE   : int32 - specifies the data point associated with
'                   x-origin.
'   YINCREMENT   : float32 - voltage difference between data points.
'   YORIGIN      : float32 - value is the voltage at center screen.
'   YREFERENCE   : int32 - specifies the data point where y-origin
'                   occurs.
Dim intFormat As Integer

```

```

Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

' Query for the preamble.
err = viVPrintf(vi, ":WAVEFORM:PREAmble?" + vbLf, 0)
paramsArray(1) = VarPtr(dblArray(0))
retCount = DblArraySize

' Read preamble information.
err = viVScanf(vi, "%, #lf" + vbLf, paramsArray(0))
intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
sngYOrigin = dblArray(8)
lngYReference = dblArray(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'      FormatNumber(dblXIncrement * 1000000) + _
'      " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'      FormatNumber(dblXOrigin * 1000000) + _
'      " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'      CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'      FormatNumber(sngYIncrement * 1000) + _
'      " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'      FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'      CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
'      FormatNumber(lngVSteps * sngYIncrement / 8) + _
'      " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
'      FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
'      FormatNumber(lngPoints * dblXIncrement / 10 * _
'      1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _

```

```

        FormatNumber(((lngPoints / 2) * _
        dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
err = viVPrintf(vi, ":WAV:DATA?" + vbLf, 0)

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'     <header><waveform_data><NL>
'
' Where:
'
'     <header> = #800001000 (This is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
'Dim lngI As Long
Dim lngDataValue As Long

paramsArray(1) = VarPtr(byteArray(0))
retCount = ByteArraySize
' Unsigned integer bytes.
err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
' retCount is now actual number of bytes returned by query.
For lngI = 0 To retCount - 1 Step (retCount / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = CLng(byteArray(lngI)) * 256 + _
            CLng(byteArray(lngI + 1)) ' 16-bit value.
    Else
        lngDataValue = CLng(byteArray(lngI)) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) * sngYIncrement + _
            sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.

```



```

err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN1" + vbLf, 0)

' Read time at edge 1 on ch 1.
err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge1))

' Query time at 1st rising edge on ch2.
err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN2" + vbLf, 0)

' Read time at edge 1 on ch 2.
err = viVScanf(vi, "%lf", VarPtr(dblChan2Edge1))

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.

' Query time at 1st rising edge on ch1.
err = viVPrintf(vi, ":MEASURE:TEDGE? +2, CHAN1" + vbLf, 0)

' Read time at edge 2 on ch 1.
err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge2))

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

End Sub

```

## VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual C#, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.
- 4 Edit the program to use the VISA address of your oscilloscope.

**5 Add Agilent's VISA header file to your project:**

- a** Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
- b** Click **Add** and then click **Add Existing Item...**
- c** Navigate to the header file, visa32.cs (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but *do not click the Open button*.
- d** Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

**6 Build and run the program.**

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```

/*
 * Agilent VISA Example in C#
 * -----
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -----
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
    class VisaInstrumentApp
    {
        private static VisaInstrument oscp;

        public static void Main(string[] args)
        {
            try
            {
                oscp = new
                    VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR");

                Initialize();

                /* The extras function contains miscellaneous commands that
                 * do not need to be executed for the proper operation of
                 * this example. The commands in the extras function are
                 * shown for reference purposes only.
                 */
                // Extra();    // Uncomment to execute the extra function.
            }
            catch { }
        }
    }
}

```

```

        Capture();
        Analyze();
    }
    catch (System.ApplicationException err)
    {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
    }
    catch (System.SystemException err)
    {
        Console.WriteLine("*** System Error Message : " + err.Message);
    }
    catch (System.Exception err)
    {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
    }
    finally
    {
        oscp.Close();
    }
}

/*
 * Initialize()
 * -----
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
    StringBuilder strResults;

    /* RESET - This command puts the oscilloscope into a known
     * state. This statement is very important for programs to
     * work as expected. Most of the following initialization
     * commands are initialized by *RST. It is not necessary to
     * reinitialize them unless the default setting is not suitable
     * for your application.
     */
    oscp.DoCommand("*RST"); // Reset the to the defaults.
    oscp.DoCommand("*CLS"); // Clear the status data structures.

    /* IDN - Ask for the device's *IDN string.
     */
    strResults = oscp.DoQueryString("*IDN?");

    // Display results.
    Console.WriteLine("Result is: {0}", strResults);

    /* AUTOSCALE - This command evaluates all the input signals
     * and sets the correct conditions to display all of the
     * active signals.
     */
    oscp.DoCommand(":AUToscale");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from

```

```

    * 0.1 to 1000.
    */
    oscp.DoCommand(":CHANnel1:PROBe 10");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
    oscp.DoCommand(":CHANnel1:RANGe 8");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    * The range value is ten times the time per division.
    */
    oscp.DoCommand(":TIMEbase:RANGe 2e-3");

    /* TIME_REFERENCE - Possible values are LEFT and CENTER:
    * - LEFT sets the display reference one time division from
    *   the left.
    * - CENTER sets the display reference to the center of the
    *   screen.
    */
    oscp.DoCommand(":TIMEbase:REFerence CENTer");

    /* TRIGGER_SOURCE - Selects the channel that actually produces
    * the TV trigger. Any channel can be selected.
    */
    oscp.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

    /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCh,
    * PATtern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
    * UART, or USB.
    */
    oscp.DoCommand(":TRIGger:MODE EDGE");

    /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
    * trigger to either POSITIVE or NEGATIVE.
    */
    oscp.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
}

/*
 * Extra()
 * -----
 * The commands in this function are not executed and are shown
 * for reference purposes only. To execute these commands, call
 * this function from main.
 */
private static void Extra()
{
    /* RUN_STOP (not executed in this example):
    * - RUN starts the acquisition of data for the active
    *   waveform display.
    * - STOP stops the data acquisition and turns off AUTOSTORE.
    */
    oscp.DoCommand(":RUN");
    oscp.DoCommand(":STOP");

    /* VIEW_BLANK (not executed in this example):

```

```

    * - VIEW turns on (starts displaying) an active channel or
    * pixel memory.
    * - BLANK turns off (stops displaying) a specified channel or
    * pixel memory.
    */
    oscp.DoCommand(":BLANK CHANNEL1");
    oscp.DoCommand(":VIEW CHANNEL1");

    /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
    oscp.DoCommand(":TIMEbase:MODE MAIN");
}

/*
* Capture()
* -----
* This function prepares the scope for data acquisition and then
* uses the DIGITIZE MACRO to capture some data.
*/
private static void Capture()
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
    oscp.DoCommand(":ACQUIRE:TYPE NORMAL");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition. The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
    oscp.DoCommand(":ACQUIRE:COMPLETE 100");

    /* DIGITIZE - Used to acquire the waveform data for transfer
    * over the interface. Sending this command causes an
    * acquisition to take place with the resulting data being
    * placed in the buffer.
    */

    /* NOTE! The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement. Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be
    * reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate. An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
    oscp.DoCommand(":DIGitize CHANNEL1");
}

/*
* Analyze()

```

```

* -----
* In this example we will do the following:
* - Save the system setup to a file for restoration at a later
*   time.
* - Save the oscilloscope display to a file which can be
*   printed.
* - Make single channel measurements.
*/
private static void Analyze()
{
    byte[] ResultsArray;    // Results array.
    int nLength;           // Number of bytes returned from instrument.

    /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETup? query returns a
     * program message that contains the current state of the
     * instrument. Its format is a definite-length binary block,
     * for example,
     *   #800002204<setup string><NL>
     * where the setup string is 2204 bytes in length.
     */
    Console.WriteLine("Saving oscilloscope setup to " +
        "c:\\scope\\config\\setup.dat");
    if (File.Exists("c:\\scope\\config\\setup.dat"))
        File.Delete("c:\\scope\\config\\setup.dat");

    // Query and read setup string.
    nLength = oscp.DoQueryIEEEBlock(":SYSTEM:SETup?",
        out ResultsArray);
    Console.WriteLine("Read oscilloscope setup ({0} bytes).",
        nLength);

    // Write setup string to file.
    File.WriteAllBytes("c:\\scope\\config\\setup.dat",
        ResultsArray);
    Console.WriteLine("Wrote setup string ({0} bytes) to file.",
        nLength);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
     * string to the oscilloscope.
     */
    byte[] DataArray;
    int nBytesWritten;

    // Read setup string from file.
    DataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
    Console.WriteLine("Read setup string ({0} bytes) from file.",
        DataArray.Length);

    // Restore setup string.
    nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTEM:SETup",
        DataArray);
    Console.WriteLine("Restored setup string ({0} bytes).",
        nBytesWritten);

    /* IMAGE_TRANSFER - In this example, we query for the screen
     * data with the ":DISPLAY:DATA?" query. The .png format
     * data is saved to a file in the local file system.

```

```

*/
Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
oscp.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
nLength = oscp.DoQueryIEEEBlock(
    ":DISPlay:DATA? PNG, SCReen, COLOr", out ResultsArray);
Console.WriteLine("Read screen image ({0} bytes).", nLength);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
    ResultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.",
    nLength);

// Return I/O timeout to five seconds.
oscp.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

// Set source to measure.
oscp.DoCommand(":MEASure:SOURce CHANnel1");

// Query for frequency.
double fResults;
fResults = oscp.DoQueryValue(":MEASure:FREQuency?");
Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

// Query for peak to peak voltage.
fResults = oscp.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. To
 * obtain waveform data, you must specify the WAVEFORM
 * parameters for the waveform data prior to sending the
 * ":WAVEFORM:DATA?" query.
 *
 * Once these parameters have been sent, the
 * ":WAVEFORM:PREAMBLE?" query provides information concerning
 * the vertical and horizontal scaling of the waveform data.
 *
 * With the preamble information you can then use the
 * ":WAVEFORM:DATA?" query and read the data block in the
 * correct format.
 */

/* WAVE_FORMAT - Sets the data transmission mode for waveform
 * data output. This command controls how the data is

```

```

    * formatted when sent from the oscilloscope and can be set
    * to WORD or BYTE format.
    */

// Set waveform format to BYTE.
oscp.DoCommand(":WAVEform:FORMat BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
 * The number of time points available is returned by the
 * "ACQUIRE:POINTS?" query. This can be set to any binary
 * fraction of the total time points available.
 */
oscp.DoCommand(":WAVEform:POINTs 1000");

/* GET_PREAMBLE - The preamble contains all of the current
 * WAVEFORM settings returned in the form <preamble block><NL>
 * where the <preamble block> is:
 *   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
 *   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT,
 *                 2 = AVERAGE.
 *   POINTS      : int32 - number of data points transferred.
 *   COUNT       : int32 - 1 and is always 1.
 *   XINCREMENT  : float64 - time difference between data
 *                         points.
 *   XORIGIN     : float64 - always the first data point in
 *                         memory.
 *   XREFERENCE  : int32 - specifies the data point associated
 *                         with the x-origin.
 *   YINCREMENT  : float32 - voltage difference between data
 *                         points.
 *   YORIGIN     : float32 - value of the voltage at center
 *                         screen.
 *   YREFERENCE  : int32 - data point where y-origin occurs.
 */
Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = oscp.DoQueryValues(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMat: {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: {0:e}", fType);

double fPoints = fResultsArray[2];
Console.WriteLine("Preamble POINTs: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];

```



```

Console.WriteLine("Preamble XREference: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEform:SOURce" command.
 */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character. The query data has the following
 * format:
 *
 * <header><waveform data block><NL>
 *
 * Where:
 *
 * <header> = #800002048 (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the
 * ":WAVEFORM:POINTS" command. You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVEform:DATA?",
    out ResultsArray);
Console.WriteLine("Read waveform data ({0} bytes).", nLength);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv = fPoints * fXincrement / 10;
double fDelay = (fPoints / 2) * fXincrement + fXorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < 1000; i = i + 50)
    Console.WriteLine("Data point {0:d} = {1:f2} Volts at "

```

```

        + "{2:f10} Seconds", i,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin,
        ((float)i - fXreference) * fXincrement + fXorigin);

/* SAVE_WAVE_DATA - saves the waveform data to a CSV format
 * file named "waveform.csv".
 */
if (File.Exists("c:\\scope\\data\\waveform.csv"))
    File.Delete("c:\\scope\\data\\waveform.csv");

StreamWriter writer =
    File.CreateText("c:\\scope\\data\\waveform.csv");
for (int i = 0; i < 1000; i++)
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fXreference) * fXincrement + fXorigin,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin);
writer.Close();
}
}

class VisaInstrument
{
    private int m_nResourceManager;
    private int m_nSession;
    private string m_strVisaAddress;

    // Constructor.
    public VisaInstrument(string strVisaAddress)
    {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
    }

    public void DoCommand(string strCommand)
    {
        // Send the command.
        VisaSendCommandOrQuery(strCommand);

        // Check for instrument errors (another command and result).
        CheckForInstrumentErrors(strCommand);
    }

    public int DoCommandIEEEBlock(string strCommand,
        byte[] DataArray)
    {

```

```

// Send the command to the device.
string strCommandAndLength;
int nViStatus, nLength, nBytesWritten;

nLength = DataArray.Length;
strCommandAndLength = String.Format("{0} #8{1:D8}",
    strCommand, nLength);

// Write first part of command to formatted I/O write buffer.
nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength);
CheckVisaStatus(nViStatus);

// Write the data to the formatted I/O write buffer.
nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
    out nBytesWritten);
CheckVisaStatus(nViStatus);

// Write command termination character.
nViStatus = visa32.viPrintf(m_nSession, "\n");
CheckVisaStatus(nViStatus);

// Check for instrument errors (another command and result).
CheckForInstrumentErrors(strCommand);

return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
}

public double DoQueryValue(string strQuery)
{
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultValue();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
}

```

```

    }

    public double[] DoQueryValues(string strQuery)
    {
        // Send the query.
        VisaSendCommandOrQuery(strQuery);

        // Get the result string.
        double[] fResultsArray;
        fResultsArray = VisaGetResultValues();

        // Check for instrument errors (another command and result).
        CheckForInstrumentErrors(strQuery);

        // Return string results.
        return fResultsArray;
    }

    public int DoQueryIEEEBlock(string strQuery,
        out byte[] ResultsArray)
    {
        // Send the query.
        VisaSendCommandOrQuery(strQuery);

        // Get the result string.
        int length;    // Number of bytes returned from instrument.
        length = VisaGetResultIEEEBlock(out ResultsArray);

        // Check for instrument errors (another command and result).
        CheckForInstrumentErrors(strQuery);

        // Return string results.
        return length;
    }

    private void CheckForInstrumentErrors(string strCommand)
    {
        // Check for instrument errors.
        StringBuilder strInstrumentError = new StringBuilder(1000);
        bool bFirstError = true;
        do
        {
            VisaSendCommandOrQuery(":SYSTem:ERRor?");
            strInstrumentError = VisaGetResultString();

            if (strInstrumentError.ToString() != "+0, \"No error\\\"\\n\")
            {
                if (bFirstError)
                {
                    Console.WriteLine("ERROR(s) for command '{0}': ",
                        strCommand);
                    bFirstError = false;
                }
                Console.Write(strInstrumentError);
            }
        } while (strInstrumentError.ToString() != "+0, \"No error\\\"\\n\");
    }
}

```

```

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
    // Send command or query to the device.
    string strWithNewline;
    strWithNewline = String.Format("{0}\n", strCommandOrQuery);
    int nViStatus;
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
    CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
    StringBuilder strResults = new StringBuilder(1000);

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
    CheckVisaStatus(nViStatus);

    return strResults;
}

private double VisaGetResultValue()
{
    double fResults = 0;

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
    CheckVisaStatus(nViStatus);

    return fResults;
}

private double[] VisaGetResultValues()
{
    double[] fResultsArray;
    fResultsArray = new double[10];

    // Read return value string from the device.
    int nViStatus;
    nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
        fResultsArray);
    CheckVisaStatus(nViStatus);

    return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
    // Results array, big enough to hold a PNG.
    ResultsArray = new byte[300000];
    int length;    // Number of bytes returned from instrument.

    // Set the default number of bytes that will be contained in
    // the ResultsArray to 300,000 (300kB).

```

```

length = 300000;

// Read return value string from the device.
int nViStatus;
nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
    ResultsArray);
CheckVisaStatus(nViStatus);

// Write and read buffers need to be flushed after IEEE block?
nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
CheckVisaStatus(nViStatus);
nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
CheckVisaStatus(nViStatus);

return length;
}

private void OpenResourceManager()
{
    int nViStatus;
    nViStatus =
        visa32.viOpenDefaultRM(out this.m_nResourceManager);
    if (nViStatus < visa32.VI_SUCCESS)
        throw new
            ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
    int nViStatus;
    nViStatus = visa32.viOpen(this.m_nResourceManager,
        this.m_strVisaAddress, visa32.VI_NO_LOCK,
        visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
    CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
    int nViStatus;
    nViStatus = visa32.viSetAttribute(this.m_nSession,
        visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
    CheckVisaStatus(nViStatus);
}

public void CheckVisaStatus(int nViStatus)
{
    // If VISA error, throw exception.
    if (nViStatus < visa32.VI_SUCCESS)
    {
        StringBuilder strError = new StringBuilder(256);
        visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
            strError);
        throw new ApplicationException(strError.ToString());
    }
}

public void Close()

```

```

    {
        if (m_nSession != 0)
            visa32.viClose(m_nSession);
        if (m_nResourceManager != 0)
            visa32.viClose(m_nResourceManager);
    }
}
}

```

## VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual Basic, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the Visual Basic .NET source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add Agilent's VISA header file to your project:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add** and then choose **Add Existing Item...**
  - c Navigate to the header file, visa32.vb (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but *do not click the Open button*.
  - d Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

- e Right-click the project again and choose **Properties**; then, select "InfiniiVision.VisaInstrumentApp" as the **Startup object**.
- 6 Build and run the program.

For more information, see the 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 most of the commonly-used programming
' features of your Agilent oscilloscope.
' -----

Imports System
Imports System.IO

```

```

Imports System.Text

Namespace InfiniiVision
Class VisaInstrumentApp
    Private Shared oscp As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
        Try
            oscp = _
                New VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR")

            Initialize()

            ' The extras function contains miscellaneous commands that
            ' do not need to be executed for the proper operation of
            ' this example. The commands in the extras function are
            ' shown for reference purposes only.

            ' Extra()    ' Uncomment to execute the extra function.
            Capture()
            Analyze()
        Catch err As System.ApplicationException
            MsgBox("**** Error : " & err.Message, vbExclamation, _
                "VISA Error Message")
        Exit Sub
        Catch err As System.SystemException
            MsgBox("**** Error : " & err.Message, vbExclamation, _
                "System Error Message")
        Exit Sub
        Catch err As System.Exception
            Debug.Fail("Unexpected Error")
            MsgBox("**** Error : " & err.Message, vbExclamation, _
                "Unexpected Error")
        Exit Sub
    Finally
        oscp.Close()
    End Try
End Sub

' Initialize()
' -----
' This function initializes both the interface and the
' oscilloscope to a known state.

Private Shared Sub Initialize()
    Dim strResults As StringBuilder

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

    ' Reset the to the defaults.
    oscp.DoCommand("*RST")
    ' Clear the status data structures.

```



```

oscp.DoCommand("*CLS")

' IDN - Ask for the device's *IDN string.
strResults = oscp.DoQueryString("*IDN?")
' Display results.
Console.WriteLine("Result is: {0}", strResults)

' AUTOSCALE - This command evaluates all the input signals
' and sets the correct conditions to display all of the
' active signals.
oscp.DoCommand(":AUToscale")

' CHANNEL_PROBE - Sets the probe attenuation factor for the
' selected channel. The probe attenuation factor may be from
' 0.1 to 1000.
oscp.DoCommand(":CHANnel1:PROBe 10")

' CHANNEL_RANGE - Sets the full scale vertical range in volts.
' The range value is eight times the volts per division.
oscp.DoCommand(":CHANnel1:RANGE 8")

' TIME_RANGE - Sets the full scale horizontal time in seconds.
' The range value is ten times the time per division.
oscp.DoCommand(":TIMEbase:RANGE 2e-3")

' TIME_REFERENCE - Possible values are LEFT and CENTER:
' - LEFT sets the display reference one time division from
' the left.
' - CENTER sets the display reference to the center of the
' screen.
oscp.DoCommand(":TIMEbase:REFErence CENTer")

' TRIGGER_SOURCE - Selects the channel that actually produces
' the TV trigger. Any channel can be selected.
oscp.DoCommand(":TRIGger:TV:SOURCe CHANnel1")

' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCh,
' PATtern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
' UART, or USB.
oscp.DoCommand(":TRIGger:MODE EDGE")

' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
' trigger to either POSITIVE or NEGATIVE.
oscp.DoCommand(":TRIGger:EDGE:SLOPe POSitive")

End Sub

' Extra()
' -----
' The commands in this function are not executed and are shown
' for reference purposes only. To execute these commands, call
' this function from main.

Private Shared Sub Extra()

' RUN_STOP (not executed in this example):

```

```

' - RUN starts the acquisition of data for the active
'   waveform display.
' - STOP stops the data acquisition and turns off AUTOSTORE.
oscp.DoCommand(":RUN")
oscp.DoCommand(":STOP")

' VIEW_BLANK (not executed in this example):
' - VIEW turns on (starts displaying) an active channel or
'   pixel memory.
' - BLANK turns off (stops displaying) a specified channel or
'   pixel memory.
oscp.DoCommand(":BLANK CHANNEL1")
oscp.DoCommand(":VIEW CHANNEL1")

' TIME_MODE (not executed in this example) - Set the time base
' mode to MAIN, DELAYED, XY or ROLL.
oscp.DoCommand(":TIMEbase:MODE MAIN")

End Sub

' Capture()
' -----
' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.

Private Shared Sub Capture()

' ACQUIRE_TYPE - Sets the acquisition mode. There are three
' acquisition types NORMAL, PEAK, or AVERAGE.
oscp.DoCommand(":ACQUIRE:TYPE NORMAL")

' ACQUIRE_COMPLETE - Specifies the minimum completion criteria
' for an acquisition. The parameter determines the percentage
' of time buckets needed to be "full" before an acquisition is
' considered to be complete.
oscp.DoCommand(":ACQUIRE:COMPLETE 100")

' DIGITIZE - Used to acquire the waveform data for transfer
' over the interface. Sending this command causes an
' acquisition to take place with the resulting data being
' placed in the buffer.

' NOTE! The use of the DIGITIZE command is highly recommended
' as it will ensure that sufficient data is available for
' measurement. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data
' buffers to be cleared and internal hardware to be
' reconfigured.
' If a measurement is immediately requested there may not have
' been enough time for the data acquisition process to collect
' data and the results may not be accurate. An error value of
' 9.9E+37 may be returned over the bus in this situation.
'

oscp.DoCommand(":DIGITIZE CHANNEL1")
End Sub

```

```

' Analyze()
' -----
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
'   time.
' - Save the oscilloscope display to a file which can be
'   printed.
' - Make single channel measurements.

Private Shared Sub Analyze()

    ' Results array.
    Dim ResultsArray As Byte()
    ' Number of bytes returned from instrument.
    Dim nLength As Integer

    ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETup? query returns a
    ' program message that contains the current state of the
    ' instrument. Its format is a definite-length binary block,
    ' for example,
    '   #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    Console.WriteLine("Saving oscilloscope setup to " & _
        + "c:\scope\config\setup.dat")
    If File.Exists("c:\scope\config\setup.dat") Then
        File.Delete("c:\scope\config\setup.dat")
    End If

    ' Query and read setup string.
    nLength = oscp.DoQueryIEEEBlock(":SYSTEM:SETup?", ResultsArray)
    Console.WriteLine("Read oscilloscope setup ({0} bytes).", _
        nLength)

    ' Write setup string to file.
    File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
    Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
        nLength)

    ' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
    ' string to the oscilloscope.
    Dim DataArray As Byte()
    Dim nBytesWritten As Integer

    ' Read setup string from file.
    DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
    Console.WriteLine("Read setup string ({0} bytes) from file.", _
        DataArray.Length)

    ' Restore setup string.
    nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTEM:SETup", _
        DataArray)
    Console.WriteLine("Restored setup string ({0} bytes).", _
        nBytesWritten)

    ' IMAGE_TRANSFER - In this example, we query for the screen
    ' data with the ":DISPLAY:DATA?" query. The .png format

```

```

' data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to " _
    + "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
oscp.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
nLength = _
    oscp.DoQueryIEEEBlock(":DISPlay:DATA? PNG, SCReen, COLOr", _
        ResultsArray)
Console.WriteLine("Read screen image ({0} bytes).", nLength)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
    nLength)

' Return I/O timeout to five seconds.
oscp.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to
' make measurements on displayed waveforms.

' Set source to measure.
oscp.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = oscp.DoQueryValue(":MEASure:FREQuency?")
Console.WriteLine("The frequency is: {0:F4} kHz", _
    fResults / 1000)

' Query for peak to peak voltage.
fResults = oscp.DoQueryValue(":MEASure:VPP?")
Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
    fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. To
' obtain waveform data, you must specify the WAVEFORM
' parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.
'
' Once these parameters have been sent, the
' ":WAVEFORM:PREAMBLE?" query provides information concerning
' the vertical and horizontal scaling of the waveform data.
'
' With the preamble information you can then use the
' ":WAVEFORM:DATA?" query and read the data block in the
' correct format.

' WAVE_FORMAT - Sets the data transmission mode for waveform
' data output. This command controls how the data is
' formatted when sent from the oscilloscope and can be set

```

```

' to WORD or BYTE format.

' Set waveform format to BYTE.
oscp.DoCommand(":WAVEform:FORMat BYTE")

' WAVE_POINTS - Sets the number of points to be transferred.
' The number of time points available is returned by the
' "ACQUIRE:POINTS?" query. This can be set to any binary
' fraction of the total time points available.
oscp.DoCommand(":WAVEform:POINTs 1000")

' GET_PREAMBLE - The preamble contains all of the current
' WAVEFORM settings returned in the form <preamble block><NL>
' where the <preamble block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT,
'                   2 = AVERAGE.
'   POINTS      : int32 - number of data points transferred.
'   COUNT       : int32 - 1 and is always 1.
'   XINCREMENT  : float64 - time difference between data
'                           points.
'   XORIGIN     : float64 - always the first data point in
'                           memory.
'   XREFERENCE  : int32 - specifies the data point associated
'                           with the x-origin.
'   YINCREMENT  : float32 - voltage difference between data
'                           points.
'   YORIGIN     : float32 - value of the voltage at center
'                           screen.
'   YREFERENCE  : int32 - data point where y-origin occurs.
Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double()
fResultsArray = oscp.DoQueryValues(":WAVEform:PREAmble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMat: {0:e}", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: {0:e}", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINTs: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNT: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREFerence: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement)

```

```

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREference: {0:e}", fYreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVEform:SOURce" command.

' READ_WAVE_DATA - The wave data consists of two parts: the
' header, and the actual waveform data followed by a
' New Line (NL) character. The query data has the following
' format:
'
' <header><waveform data block><NL>
'
' Where:
'
' <header> = #800002048 (this is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.
' The size can vary depending on the number of points acquired
' for the waveform which can be set using the
' ":WAVEFORM:POINTS" command. You may then read that number
' of bytes from the oscilloscope; then, read the following NL
' character to terminate the query.

' Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVEform:DATA?", ResultsArray)
Console.WriteLine("Read waveform data ({0} bytes).", nLength)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < 1000
    Console.WriteLine("Data point {0:d} = {1:f2} Volts at " + _
        "{2:f10} Seconds", i, _
        (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, _
        (CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + 50
End While

```

```

' SAVE_WAVE_DATA - saves the waveform data to a CSV format
' file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter = _
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To 999
    writer.WriteLine("{0:E}, {1:f6}", _
        (CSng(index) - fXreference) * fXincrement + fXorigin, _
        (CSng(ResultsArray(index)) - fYreference) * fYincrement _
        + fYorigin)
Next
writer.Close()
End Sub
End Class

Class VisaInstrument
    Private m_nResourceManager As Integer
    Private m_nSession As Integer
    Private m_strVisaAddress As String

    ' Constructor.
    Public Sub New(ByVal strVisaAddress As String)
        ' Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress

        ' Open the default VISA resource manager.
        OpenResourceManager()

        ' Open a VISA resource session.
        OpenSession()

        ' Clear the interface.
        Dim nViStatus As Integer
        nViStatus = visa32.viClear(m_nSession)
    End Sub

    Public Sub DoCommand(ByVal strCommand As String)
        ' Send the command.
        VisaSendCommandOrQuery(strCommand)

        ' Check for instrument errors (another command and result).
        CheckForInstrumentErrors(strCommand)
    End Sub

    Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
        ByVal dataArray As Byte()) As Integer
        ' Send the command to the device.
        Dim strCommandAndLength As String
        Dim nViStatus As Integer
        Dim nLength As Integer
        Dim nBytesWritten As Integer

        nLength = dataArray.Length

```

```

strCommandAndLength = [String].Format("{0} #8{1:D8}", _
    strCommand, nLength)

' Write first part of command to formatted I/O write buffer.
nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
CheckVisaStatus(nViStatus)

' Write the data to the formatted I/O write buffer.
nViStatus = visa32.viBufWrite(m_nSession, dataArray, nLength, _
    nBytesWritten)
CheckVisaStatus(nViStatus)

' Write command termination character.
nViStatus = visa32.viPrintf(m_nSession, "" & Chr(10) & "")
CheckVisaStatus(nViStatus)

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strCommand)

Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
    As StringBuilder
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim strResults As New StringBuilder(1000)
    strResults = VisaGetResultString()

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResults As Double
    fResults = VisaGetResultValue()

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return fResults
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

```



```

' Get the result string.
Dim fResultsArray As Double()
fResultsArray = VisaGetResultValues()

' Check for instrument errors (another command and result).
CheckForInstrumentErrors(strQuery)

' Return string results.
Return fResultsArray
End Function

Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
    ByRef ResultsArray As Byte()) As Integer
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(ResultsArray)

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return length
End Function

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As New StringBuilder(1000)
    Dim bFirstError As Boolean = True
    Do
        VisaSendCommandOrQuery(":SYSTem:ERRor?")
        strInstrumentError = VisaGetResultString()

        If strInstrumentError.ToString() <> _
            "+0,""No error"" & Chr(10) & "" Then
            If bFirstError Then
                Console.WriteLine("ERROR(s) for command '{0}': ", _
                    strCommand)
                bFirstError = False
            End If
            Console.Write(strInstrumentError)
        End If
    Loop While strInstrumentError.ToString() <> _
        "+0,""No error"" & Chr(10) & ""
End Sub

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
    As String)
    ' Send command or query to the device.
    Dim strWithNewline As String
    strWithNewline = [String].Format("{0}" & Chr(10) & "", _
        strCommandOrQuery)
    Dim nViStatus As Integer
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline)

```

```

    CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
    Dim strResults As New StringBuilder(1000)

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
    CheckVisaStatus(nViStatus)

    Return strResults
End Function

Private Function VisaGetResultValue() As Double
    Dim fResults As Double = 0

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
    CheckVisaStatus(nViStatus)

    Return fResults
End Function

Private Function VisaGetResultValues() As Double()
    Dim fResultsArray As Double()
    fResultsArray = New Double(9) {}

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, _
        "%,10lf" & Chr(10) & "", fResultsArray)
    CheckVisaStatus(nViStatus)

    Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
    As Byte()) As Integer
    ' Results array, big enough to hold a PNG.
    ResultsArray = New Byte(299999) {}
    Dim length As Integer
    ' Number of bytes returned from instrument.
    ' Set the default number of bytes that will be contained in
    ' the ResultsArray to 300,000 (300kB).
    length = 300000

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
        ResultsArray)
    CheckVisaStatus(nViStatus)

    ' Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
    CheckVisaStatus(nViStatus)

```

```

        nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
        CheckVisaStatus(nViStatus)

        Return length
    End Function

Private Sub OpenResourceManager()
    Dim nViStatus As Integer
    nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
    If nViStatus < visa32.VI_SUCCESS Then
        Throw New _
            ApplicationException("Failed to open Resource Manager")
    End If
End Sub

Private Sub OpenSession()
    Dim nViStatus As Integer
    nViStatus = visa32.viOpen(Me.m_nResourceManager, _
        Me.m_strVisaAddress, visa32.VI_NO_LOCK, _
        visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
    CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    Dim nViStatus As Integer
    nViStatus = visa32.viSetAttribute(Me.m_nSession, _
        visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
    CheckVisaStatus(nViStatus)
End Sub

Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
    ' If VISA error, throw exception.
    If nViStatus < visa32.VI_SUCCESS Then
        Dim strError As New StringBuilder(256)
        visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
        Throw New ApplicationException(strError.ToString())
    End If
End Sub

Public Sub Close()
    If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
    End If
    If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
    End If
End Sub

End Class
End Namespace

```

## VISA COM Examples

- ["VISA COM Example in Visual Basic"](#) on page 764
- ["VISA COM Example in C#" on page 774](#)
- ["VISA COM Example in Visual Basic .NET"](#) on page 785

### VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

- 1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
- 2 Press ALT+F11 to launch the Visual Basic editor.
- 3 Reference the Agilent VISA COM library:
  - a Choose **Tools>References...** from the main menu.
  - b In the References dialog, check the "VISA COM 3.0 Type Library".
  - c Click **OK**.
- 4 Choose **Insert>Module**.
- 5 Cut-and-paste the code that follows into the editor.
- 6 Edit the program to use the VISA address of your oscilloscope, and save the changes.
- 7 Run the program.

```
'
' Agilent VISA COM Example in Visual Basic
' -----
' This program illustrates most of the commonly used programming
' features of your Agilent oscilloscopes.
' -----

Option Explicit

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

'
' MAIN PROGRAM
' -----
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form. Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
```

```

'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -----

Sub Main()

    On Error GoTo VisaComError

    ' Create the VISA COM I/O resource.
    Set myMgr = New VisaComLib.ResourceManager
    Set myScope = New VisaComLib.FormattedIO488

    ' GPIB.
    'Set myScope.IO = myMgr.Open("GPIB0::7::INSTR")

    ' LAN.
    'Set myScope.IO = myMgr.Open("TCPIP0::a-mso6102-90541::inst0::INSTR")

    ' USB.
    Set myScope.IO = myMgr.Open("USB0::2391::5970::30D3090541::0::INSTR")

    ' Initialize - Initialization will start the program with the
    ' oscilloscope in a known state.
    Initialize

    ' Capture - After initialization, you must make waveform data
    ' available to analyze. To do this, capture the data using the
    ' DIGITIZE command.
    Capture

    ' Analyze - Once the waveform has been captured, it can be analyzed.
    ' There are many parts of a waveform to analyze. This example shows
    ' some of the possible ways to analyze various parts of a waveform.
    Analyze

    Exit Sub

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

End Sub

'
' Initialize
' -----
' Initialize will start the program with the oscilloscope in a known
' state. This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
' - Oscilloscope
' - Channel 1 range
' - Display Grid
' - Timebase reference, range, and delay
' - Trigger mode and type
'

```

```
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' -----

Private Sub Initialize()

    On Error GoTo VisaComError

    ' Clear the interface.
    myScope.IO.Clear

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

    ' AUTOSCALE - This command evaluates all the input signals and sets
    ' the correct conditions to display all of the active signals.

    ' Same as pressing the Autoscale key.
    myScope.WriteString ":AUTOSCALE"

    ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
    ' channel. The probe attenuation factor may be set from 0.1 to 1000.
    myScope.WriteString ":CHAN1:PROBE 10" ' Set Probe to 10:1.

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts. The
    ' range value is 8 times the volts per division.

    ' Set the vertical range to 8 volts.
    myScope.WriteString ":CHANNEL1:RANGE 8"

    ' TIME_RANGE - Sets the full scale horizontal time in seconds. The
    ' range value is 10 times the time per division.

    ' Set the time range to 0.002 seconds.
    myScope.WriteString ":TIM:RANG 2e-3"

    ' TIME_REFERENCE - Possible values are LEFT and CENTER.
    ' - LEFT sets the display reference on time division from the left.
    ' - CENTER sets the display reference to the center of the screen.

    ' Set reference to center.
    myScope.WriteString ":TIMEBASE:REFERENCE CENTER"

    ' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
    ' TV trigger. Any channel can be selected.
    myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"

    ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITCh, PATtern, CAN,
    ' DURATION, IIC, LIN, SEquence, SPI, TV, or USB.

    ' Set the trigger mode to EDGE.
    myScope.WriteString ":TRIGGER:MODE EDGE"
```

```

' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

' Set the slope to positive.
myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"

' The following commands are not executed and are shown for reference
' purposes only. To execute these commands, uncomment them.

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

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

' TIMEBASE_MODE - (not executed in this example)
' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

' Set time base mode to main.
myScope.WriteString ":TIMEBASE:MODE MAIN"

Exit Sub

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

End Sub

'
' Capture
' -----
' We will capture the waveform using the digitize command.
' -----

Private Sub Capture()

On Error GoTo VisaComError

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

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

' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface. Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.

```

```

'
' NOTE! The DIGITIZE command is highly recommended for triggering
' modes other than SINGLE. This ensures that sufficient data is
' available for measurement. If DIGITIZE is used with single mode,
' the completion criteria may never be met. The number of points
' gathered in Single mode is related to the sweep speed, memory
' depth, and maximum sample rate. For example, take an oscilloscope
' with a 1000-point memory, a sweep speed of 10 us/div (100 us
' total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s. Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition. Now, use 1 us/div
' (10 us across the screen). 1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger. Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured. If a
' measurement is immediately requested, there may have not been
' enough time for the data acquisition process to collect data,
' and the results may not be accurate. An error value of 9.9E+37
' may be returned over the bus in this situation.
'
myScope.WriteString ":DIGITIZE CHAN1"

Exit Sub

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

End Sub

'
' Analyze
' -----
' In analyze, we will do the following:
' - Save the system setup to a file and restore it.
' - Save the waveform data to a file on the computer.
' - Make single channel measurements.
' - Save the oscilloscope display to a file that can be sent to a
'   printer.
' -----

Private Sub Analyze()

    On Error GoTo VisaComError

    ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
    ' message that contains the current state of the instrument. Its
    ' format is a definite-length binary block, for example,
    '   #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    myScope.WriteString ":SYSTEM:SETUP?"
    varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
    CheckForInstrumentErrors ' After reading query results.
    ' Output setup string to a file:

```



```

Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Close #1 ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult ' Write data.
Close #1 ' Close file.

' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
    Kill strPath
End If
Close #1 ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , byteData ' Write data.
Close #1 ' Close file.
myScope.IO.Timeout = 5000

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1 ' Open file for input.
Get #1, , varSetupString ' Read data.
Close #1 ' Close file.
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"

' Query for frequency.
myScope.WriteString ":MEASURE:FREQUENCY?"
varQueryResult = myScope.ReadNumber ' Read frequency.
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
myScope.WriteString ":MEASURE:DUTYCYCLE?"
varQueryResult = myScope.ReadNumber ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + _
    FormatNumber(varQueryResult, 3) + "%"

```

```

' Query for risetime.
myScope.WriteString ":MEASURE:RISETIME?"
varQueryResult = myScope.ReadNumber ' Read risetime.
MsgBox "Risetime:" + vbCrLf + _
    FormatNumber(varQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
myScope.WriteString ":MEASURE:VPP?"
varQueryResult = myScope.ReadNumber ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' Query for Vmax.
myScope.WriteString ":MEASURE:VMAX?"
varQueryResult = myScope.ReadNumber ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query. Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output. This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString ":WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
myScope.WriteString ":WAVEFORM:FORMAT BYTE"
lngVSteps = 256
intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings. It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'   POINTS      : int32 - number of data points transferred.
'   COUNT       : int32 - 1 and is always 1.
'   XINCREMENT  : float64 - time difference between data points.
'   XORIGIN     : float64 - always the first data point in memory.
'   XREFERENCE  : int32 - specifies the data point associated with

```

```

'           x-origin.
'   YINCREMENT   : float32 - voltage difference between data points.
'   YORIGIN      : float32 - value is the voltage at center screen.
'   YREFERENCE   : int32 - specifies the data point where y-origin
'                   occurs.

Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAmble?" ' Query for the preamble.
Preamble() = myScope.ReadList ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'           FormatNumber(dblXIncrement * 1000000) + _
'           " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'           FormatNumber(dblXOrigin * 1000000) + _
'           " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'           CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'           FormatNumber(sngYIncrement * 1000) + _
'           " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'           FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'           CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
'           FormatNumber(lngVSteps * sngYIncrement / 8) + _
'           " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
'           FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
'           FormatNumber(lngPoints * dblXIncrement / 10 * _

```

```

        1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
        FormatNumber(((lngPoints / 2) * _
        dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'     <header><waveform_data><NL>
'
' Where:
'     <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block. The
' size can vary depending on the number of points acquired for the
' waveform. You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20) ' 20 points.
    If intBytesPerData = 2 Then
        lngDataValue = varQueryResult(lngI) * 256 + _
            varQueryResult(lngI + 1) ' 16-bit value.
    Else
        lngDataValue = varQueryResult(lngI) ' 8-bit value.
    End If
    strOutput = strOutput + "Data point " + _
        CStr(lngI / intBytesPerData) + ", " + _
        FormatNumber((lngDataValue - lngYReference) * sngYIncrement + _
            sngYOrigin) + " V, " + _
        FormatNumber(((lngI / intBytesPerData - lngXReference) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"

```

```

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

Exit Sub

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

End Sub

Private Sub CheckForInstrumentErrors()

On Error GoTo VisaComError

Dim strErrVal As String
Dim strOut As String

myScope.WriteString "SYSTEM:ERROR?" ' Query any errors data.
strErrVal = myScope.ReadString ' Read: Errnum,"Error String".
While Val(strErrVal) <> 0 ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    myScope.WriteString ":SYSTEM:ERROR?" ' Request error message.
    strErrVal = myScope.ReadString ' Read error message.
Wend

If Not strOut = "" Then
MsgBox strOut, vbExclamation, "INST Error Messages"
myScope.FlushWrite (False)
myScope.FlushRead

```

```

End If

Exit Sub

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

End Sub

```

### VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual C#, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.
- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add a reference to the VISA COM 3.0 Type Library:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add Reference....**
  - c In the Add Reference dialog, select the **COM** tab.
  - d Select **VISA COM 3.0 Type Library**; then click **OK**.
- 6 Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```

/*
 * Agilent VISA COM Example in C#
 * -----
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -----
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
    class VisaComInstrumentApp
    {
        private static VisaComInstrument myScope;

        public static void Main(string[] args)

```

```

{
    try
    {
        myScope = new
            VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR");

        Initialize();

        /* The extras function contains miscellaneous commands that
         * do not need to be executed for the proper operation of
         * this example. The commands in the extras function are
         * shown for reference purposes only.
         */
        // Extra();    // Uncomment to execute the extra function.
        Capture();
        Analyze();
    }
    catch (System.ApplicationException err)
    {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
    }
    catch (System.SystemException err)
    {
        Console.WriteLine("*** System Error Message : " + err.Message);
    }
    catch (System.Exception err)
    {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
    }
    finally
    {
        myScope.Close();
    }
}

/*
 * Initialize()
 * -----
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
    string strResults;

    /* RESET - This command puts the oscilloscope into a known
     * state. This statement is very important for programs to
     * work as expected. Most of the following initialization
     * commands are initialized by *RST. It is not necessary to
     * reinitialize them unless the default setting is not suitable
     * for your application.
     */
    myScope.DoCommand("*RST");    // Reset the to the defaults.
    myScope.DoCommand("*CLS");    // Clear the status data structures.

    /* IDN - Ask for the device's *IDN string.

```

```

    */
    strResults = myScope.DoQueryString("*IDN?");

    // Display results.
    Console.WriteLine("Result is: {0}", strResults);

    /* AUTOSCALE - This command evaluates all the input signals
     * and sets the correct conditions to display all of the
     * active signals.
     */
    myScope.DoCommand(":AUToscale");

    /* CHANNEL_PROBE - Sets the probe attenuation factor for the
     * selected channel. The probe attenuation factor may be from
     * 0.1 to 1000.
     */
    myScope.DoCommand(":CHANnel1:PROBe 10");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
     * The range value is eight times the volts per division.
     */
    myScope.DoCommand(":CHANnel1:RANGe 8");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
     * The range value is ten times the time per division.
     */
    myScope.DoCommand(":TIMEbase:RANGe 2e-3");

    /* TIME_REFERENCE - Possible values are LEFT and CENTER:
     * - LEFT sets the display reference one time division from
     *   the left.
     * - CENTER sets the display reference to the center of the
     *   screen.
     */
    myScope.DoCommand(":TIMEbase:REFeRence CENTER");

    /* TRIGGER_SOURCE - Selects the channel that actually produces
     * the TV trigger. Any channel can be selected.
     */
    myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

    /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCh,
     * PATtern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
     * UART, or USB.
     */
    myScope.DoCommand(":TRIGger:MODE EDGE");

    /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
     * trigger to either POSITIVE or NEGATIVE.
     */
    myScope.DoCommand(":TRIGger:EDGE:SLOPe Positive");
}

/*
 * Extra()
 * -----
 * The commands in this function are not executed and are shown

```



```

* for reference purposes only. To execute these commands, call
* this function from main.
*/
private static void Extra()
{
    /* RUN_STOP (not executed in this example):
    * - RUN starts the acquisition of data for the active
    *   waveform display.
    * - STOP stops the data acquisition and turns off AUTOSTORE.
    */
    myScope.DoCommand(":RUN");
    myScope.DoCommand(":STOP");

    /* VIEW_BLANK (not executed in this example):
    * - VIEW turns on (starts displaying) an active channel or
    *   pixel memory.
    * - BLANK turns off (stops displaying) a specified channel or
    *   pixel memory.
    */
    myScope.DoCommand(":BLANK CHANNEL1");
    myScope.DoCommand(":VIEW CHANNEL1");

    /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
    myScope.DoCommand(":TIMEbase:MODE MAIN");
}

/*
* Capture()
* -----
* This function prepares the scope for data acquisition and then
* uses the DIGITIZE MACRO to capture some data.
*/
private static void Capture()
{
    /* ACQUIRE_TYPE - Sets the acquisition mode. There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
    myScope.DoCommand(":ACQUIRE:TYPE NORMAL");

    /* ACQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition. The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
    myScope.DoCommand(":ACQUIRE:COMPLETE 100");

    /* DIGITIZE - Used to acquire the waveform data for transfer
    * over the interface. Sending this command causes an
    * acquisition to take place with the resulting data being
    * placed in the buffer.
    */

    /* NOTE! The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement. Keep in mind when the oscilloscope is running,

```

```

    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be
    * reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate. An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze()
 * -----
 * In this example we will do the following:
 * - Save the system setup to a file for restoration at a later
 *   time.
 * - Save the oscilloscope display to a file which can be
 *   printed.
 * - Make single channel measurements.
 */
private static void Analyze()
{
    byte[] ResultsArray;    // Results array.
    int nBytes;             // Number of bytes returned from instrument.

    /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETup? query returns a
     * program message that contains the current state of the
     * instrument. Its format is a definite-length binary block,
     * for example,
     *      #800002204<setup string><NL>
     * where the setup string is 2204 bytes in length.
     */
    Console.WriteLine("Saving oscilloscope setup to " +
        "c:\\scope\\config\\setup.dat");
    if (File.Exists("c:\\scope\\config\\setup.dat"))
        File.Delete("c:\\scope\\config\\setup.dat");

    // Query and read setup string.
    ResultsArray = myScope.DoQueryIEEEBlock(":SYSTEM:SETup?");
    nBytes = ResultsArray.Length;
    Console.WriteLine("Read oscilloscope setup ({0} bytes).",
        nBytes);

    // Write setup string to file.
    File.WriteAllBytes("c:\\scope\\config\\setup.dat",
        ResultsArray);
    Console.WriteLine("Wrote setup string ({0} bytes) to file.",
        nBytes);

    /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
     * string to the oscilloscope.
     */
    byte[] DataArray;

    // Read setup string from file.

```

```

dataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
Console.WriteLine("Read setup string ({0} bytes) from file.",
    dataArray.Length);

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", dataArray);
Console.WriteLine("Restored setup string.");

/* IMAGE_TRANSFER - In this example, we query for the screen
 * data with the ":DISPLAY:DATA?" query. The .png format
 * data is saved to a file in the local file system.
 */
Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
resultsArray = myScope.DoQueryIEEEBlock(
    ":DISPlay:DATA? PNG, SCReen, COLOr");
nBytes = resultsArray.Length;
Console.WriteLine("Read screen image ({0} bytes).", nBytes);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
    resultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.",
    nBytes);

// Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

// Set source to measure.
myScope.DoCommand(":MEASure:SOURce CHANnel1");

// Query for frequency.
double fResults;
fResults = myScope.DoQueryValue(":MEASure:FREQuency?");
Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

// Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. To
 * obtain waveform data, you must specify the WAVEFORM
 * parameters for the waveform data prior to sending the
 * ":WAVEFORM:DATA?" query.

```

```

*
* Once these parameters have been sent, the
* ":WAVEFORM:PREAMBLE?" query provides information concerning
* the vertical and horizontal scaling of the waveform data.
*
* With the preamble information you can then use the
* ":WAVEFORM:DATA?" query and read the data block in the
* correct format.
*/

/* WAVE_FORMAT - Sets the data transmission mode for waveform
* data output. This command controls how the data is
* formatted when sent from the oscilloscope and can be set
* to WORD or BYTE format.
*/

// Set waveform format to BYTE.
myScope.DoCommand(":WAVEform:FORMat BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
* The number of time points available is returned by the
* "ACQUIRE:POINTS?" query. This can be set to any binary
* fraction of the total time points available.
*/
myScope.DoCommand(":WAVEform:POINTs 1000");

/* GET_PREAMBLE - The preamble contains all of the current
* WAVEFORM settings returned in the form <preamble block><NL>
* where the <preamble block> is:
*   FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
*   TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT,
*                 2 = AVERAGE.
*   POINTS      : int32 - number of data points transferred.
*   COUNT       : int32 - 1 and is always 1.
*   XINCREMENT  : float64 - time difference between data
*                         points.
*   XORIGIN     : float64 - always the first data point in
*                         memory.
*   XREFERENCE  : int32 - specifies the data point associated
*                         with the x-origin.
*   YINCREMENT  : float32 - voltage difference between data
*                         points.
*   YORIGIN     : float32 - value of the voltage at center
*                         screen.
*   YREFERENCE  : int32 - data point where y-origin occurs.
*/
Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = myScope.DoQueryValues(":WAVEform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMat: {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: {0:e}", fType);

double fPoints = fResultsArray[2];

```

```

Console.WriteLine("Preamble POINTs: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNT: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Preamble XREFerence: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVEform:SOURce" command.
 */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character. The query data has the following
 * format:
 *
 *      <header><waveform data block><NL>
 *
 * Where:
 *
 *      <header> = #800002048      (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the
 * ":WAVEFORM:POINTS" command. You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?");
nBytes = ResultsArray.Length;
Console.WriteLine("Read waveform data ({0} bytes).", nBytes);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv = fPoints * fXincrement / 10;

```

```

double fDelay = (fPoints / 2) * fXincrement + fXorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < nBytes; i = i + (nBytes / 20))
{
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at "
        + "{2:f10} Seconds", i,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin,
        ((float)i - fXreference) * fXincrement + fXorigin);
}

/* SAVE_WAVE_DATA - saves the waveform data to a CSV format
 * file named "waveform.csv".
 */
if (File.Exists("c:\\scope\\data\\waveform.csv"))
    File.Delete("c:\\scope\\data\\waveform.csv");

StreamWriter writer =
    File.CreateText("c:\\scope\\data\\waveform.csv");
for (int i = 0; i < nBytes; i++)
{
    writer.WriteLine("{0:E}, {1:f6}",
        ((float)i - fXreference) * fXincrement + fXorigin,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin);
}
writer.Close();
Console.WriteLine("Waveform data ({0} points) written to " +
    "c:\\scope\\data\\waveform.csv.", nBytes);
}
}

class VisaComInstrument
{
    private ResourceManagerClass 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 instrument errors.
    CheckForInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result string.
    string strResults;
    strResults = m_IoObject.ReadString();

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return results string.
    return strResults;
}

public double DoQueryValue(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result number.
    double fResult;
    fResult = (double)m_IoObject.ReadNumber(
        IEEEASCIIType.ASCIIType_R8, true);

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);

    // Return result number.
    return fResult;
}

public double[] DoQueryValues(string strQuery)
{
    // Send the query.
    m_IoObject.WriteString(strQuery, true);

    // Get the result numbers.
    double[] fResultsArray;
    fResultsArray = (double[])m_IoObject.ReadList(
        IEEEASCIIType.ASCIIType_R8, ",;");

    // Check for instrument errors.
    CheckForInstrumentErrors(strQuery);
}

```

```

        // Return result numbers.
        return fResultsArray;
    }

    public byte[] DoQueryIEEEBlock(string strQuery)
    {
        // Send the query.
        m_IoObject.WriteString(strQuery, true);

        // Get the results array.
        byte[] ResultsArray;
        ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
            IEEEBinaryType.BinaryType_UI1, false, true);

        // Check for instrument errors.
        CheckForInstrumentErrors(strQuery);

        // Return results array.
        return ResultsArray;
    }

    public void DoCommandIEEEBlock(string strCommand,
        byte[] DataArray)
    {
        // Send the command.
        m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

        // Check for instrument errors.
        CheckForInstrumentErrors(strCommand);
    }

    private void CheckForInstrumentErrors(string strCommand)
    {
        string strInstrumentError;
        bool bFirstError = true;

        // Repeat until all errors are displayed.
        do
        {
            // Send the ":SYSTem:ERror?" query, and get the result string.
            m_IoObject.WriteString(":SYSTem:ERror?", true);
            strInstrumentError = m_IoObject.ReadString();

            // If there is an error, print it.
            if (strInstrumentError.ToString() != "+0,\"No error\"\\n")
            {
                if (bFirstError)
                {
                    // Print the command that caused the error.
                    Console.WriteLine("ERROR(s) for command '{0}': ",
                        strCommand);
                    bFirstError = false;
                }
                Console.Write(strInstrumentError);
            }
        } while (strInstrumentError.ToString() != "+0,\"No error\"\\n");
    }
}

```



```

private void OpenIo()
{
    m_ResourceManager = new ResourceManagerClass();
    m_IoObject = new FormattedIO488Class();

    // Open the default VISA COM IO object.
    try
    {
        m_IoObject.IO =
            (IMessage)m_ResourceManager.Open(m_strVisaAddress,
            AccessMode.NO_LOCK, 0, "");
    }
    catch (Exception e)
    {
        Console.WriteLine("An error occurred: {0}", e.Message);
    }
}

public void SetTimeoutSeconds(int nSeconds)
{
    m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
    try
    {
        m_IoObject.IO.Close();
    }
    catch {}

    try
    {
        Marshal.ReleaseComObject(m_IoObject);
    }
    catch {}

    try
    {
        Marshal.ReleaseComObject(m_ResourceManager);
    }
    catch {}
}
}

```

## VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:

- 1 Open Visual Studio.
- 2 Create a new Visual Basic, Windows, Console Application project.
- 3 Cut-and-paste the code that follows into the C# source file.

- 4 Edit the program to use the VISA address of your oscilloscope.
- 5 Add a reference to the VISA COM 3.0 Type Library:
  - a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
  - b Choose **Add Reference...**
  - c In the Add Reference dialog, select the **COM** tab.
  - d Select **VISA COM 3.0 Type Library**; then click **OK**.
  - e Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.
- 6 Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA COM Example in Visual Basic .NET
' -----
' This program illustrates most of the commonly used programming
' features of your Agilent oscilloscopes.
' -----

Imports System
Imports System.IO
Imports System.Text
Imports 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::2391::5957::MY47250010::0::INSTR")

        Initialize()

        ' The extras function contains miscellaneous commands that
        ' do not need to be executed for the proper operation of
        ' this example. The commands in the extras function are
        ' shown for reference purposes only.

        ' Extra(); // Uncomment to execute the extra function.
        Capture()
        Analyze()
      Catch err As System.ApplicationException
        Console.WriteLine("*** VISA Error Message : " + err.Message)
      End Try
    End Sub
  End Class
End Namespace
```

```

Catch err As System.SystemException
    Console.WriteLine("*** System Error Message : " + err.Message)
Catch err As System.Exception
    System.Diagnostics.Debug.Fail("Unexpected Error")
    Console.WriteLine("*** Unexpected Error : " + err.Message)
Finally
    myScope.Close()
End Try
End Sub

' Initialize()
' -----
' This function initializes both the interface and the
' oscilloscope to a known state.

Private Shared Sub Initialize()
    Dim strResults As String

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

    ' Reset to the defaults.
    myScope.DoCommand("*RST")

    ' Clear the status data structures.
    myScope.DoCommand("*CLS")

    ' IDN - Ask for the device's *IDN string.
    strResults = myScope.DoQueryString("*IDN?")

    ' Display results.
    Console.WriteLine("Result is: {0}", strResults)

    ' AUTOSCALE - This command evaluates all the input signals
    ' and sets the correct conditions to display all of the
    ' active signals.
    myScope.DoCommand(":AUToscale")

    ' CHANNEL_PROBE - Sets the probe attenuation factor for the
    ' selected channel. The probe attenuation factor may be from
    ' 0.1 to 1000.
    myScope.DoCommand(":CHANnel1:PROBe 10")

    ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
    ' The range value is eight times the volts per division.
    myScope.DoCommand(":CHANnel1:RANGe 8")

    ' TIME_RANGE - Sets the full scale horizontal time in seconds.
    ' The range value is ten times the time per division.
    myScope.DoCommand(":TIMEbase:RANGe 2e-3")

    ' TIME_REFERENCE - Possible values are LEFT and CENTER:
    ' - LEFT sets the display reference one time division from

```

```

' the left.
' - CENTER sets the display reference to the center of the
' screen.
myScope.DoCommand(":TIMEbase:REFErence CENTer")

' TRIGGER_SOURCE - Selects the channel that actually produces
' the TV trigger. Any channel can be selected.
myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1")

' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITCh,
' PATTErn, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
' UART, or USB.
myScope.DoCommand(":TRIGger:MODE EDGE")

' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
' trigger to either POSITIVE or NEGATIVE.
myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")

End Sub

'
' Extra()
' -----
' The commands in this function are not executed and are shown
' for reference purposes only. To execute these commands, call
' this function from main.
'

Private Shared Sub Extra()
' RUN_STOP (not executed in this example):
' - RUN starts the acquisition of data for the active
' waveform display.
' - STOP stops the data acquisition and turns off AUTOSTORE.
'

myScope.DoCommand(":RUN")
myScope.DoCommand(":STOP")

' VIEW_BLANK (not executed in this example):
' - VIEW turns on (starts displaying) an active channel or
' pixel memory.
' - BLANK turns off (stops displaying) a specified channel or
' pixel memory.
'

myScope.DoCommand(":BLANK CHANnel1")
myScope.DoCommand(":VIEW CHANnel1")

' TIME_MODE (not executed in this example) - Set the time base
' mode to MAIN, DELAYED, XY or ROLL.
'

myScope.DoCommand(":TIMEbase:MODE MAIN")
End Sub

' Capture()
' -----

```

```
' This function prepares the scope for data acquisition and then
' uses the DIGITIZE MACRO to capture some data.
```

```
Private Shared Sub Capture()
    ' ACQUIRE_TYPE - Sets the acquisition mode. There are three
    ' acquisition types NORMAL, PEAK, or AVERAGE.
    myScope.DoCommand(":ACquire:TYPE NORMal")

    ' ACQUIRE_COMPLETE - Specifies the minimum completion criteria
    ' for an acquisition. The parameter determines the percentage
    ' of time buckets needed to be "full" before an acquisition is
    ' considered to be complete.
    myScope.DoCommand(":ACquire:COMplete 100")

    ' DIGITIZE - Used to acquire the waveform data for transfer
    ' over the interface. Sending this command causes an
    ' acquisition to take place with the resulting data being
    ' placed in the buffer.

    ' NOTE! The use of the DIGITIZE command is highly recommended
    ' as it will ensure that sufficient data is available for
    ' measurement. Keep in mind when the oscilloscope is running,
    ' communication with the computer interrupts data acquisition.
    ' Setting up the oscilloscope over the bus causes the data
    ' buffers to be cleared and internal hardware to be
    ' reconfigured.
    ' If a measurement is immediately requested there may not have
    ' been enough time for the data acquisition process to collect
    ' data and the results may not be accurate. An error value of
    ' 9.9E+37 may be returned over the bus in this situation.
    myScope.DoCommand(":DIGitize CHANnel1")
End Sub
```

```
' Analyze()
' -----
' In this example we will do the following:
' - Save the system setup to a file for restoration at a later
'   time.
' - Save the oscilloscope display to a file which can be
'   printed.
' - Make single channel measurements.
```

```
Private Shared Sub Analyze()
    ' Results array.
    Dim ResultsArray As Byte()

    ' Number of bytes returned from instrument.
    Dim nBytes As Integer

    ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETup? query returns a
    ' program message that contains the current state of the
    ' instrument. Its format is a definite-length binary block,
    ' for example,
    ' #800002204<setup string><NL>
    ' where the setup string is 2204 bytes in length.
    Console.WriteLine("Saving oscilloscope setup to " + _
```

```

        "c:\scope\config\setup.dat")
If File.Exists("c:\scope\config\setup.dat") Then
    File.Delete("c:\scope\config\setup.dat")
End If

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nBytes = ResultsArray.Length
Console.WriteLine("Read oscilloscope setup ({0} bytes).", nBytes)

' Write setup string to file.
File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
    nBytes)

' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
' string to the oscilloscope.
Dim dataArray As Byte()

' Read setup string from file.
dataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
Console.WriteLine("Read setup string ({0} bytes) from file.", _
    dataArray.Length)

' Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", dataArray)
Console.WriteLine("Restored setup string.")

' IMAGE_TRANSFER - In this example, we query for the screen
' data with the ":DISPLAY:DATA?" query. The .png format
' data is saved to a file in the local file system.
Console.WriteLine("Transferring screen image to " + _
    "c:\scope\data\screen.png")
If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
End If

' Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15)

' Get the screen data in PNG format.
ResultsArray = _
    myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, SCReen, COLor")
nBytes = ResultsArray.Length
Console.WriteLine("Read screen image ({0} bytes).", nBytes)

' Store the screen data in a file.
File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
    nBytes)

' Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5)

' MEASURE - The commands in the MEASURE subsystem are used to
' make measurements on displayed waveforms.

```

```

' Set source to measure.
myScope.DoCommand(":MEASure:SOURce CHANnel1")

' Query for frequency.
Dim fResults As Double
fResults = myScope.DoQueryValue(":MEASure:FREQuency?")
Console.WriteLine("The frequency is: {0:F4} kHz", _
    fResults / 1000)

' Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?")
Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
    fResults)

' WAVEFORM_DATA - Get waveform data from oscilloscope. To
' obtain waveform data, you must specify the WAVEFORM
' parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.
'
' Once these parameters have been sent, the
' ":WAVEFORM:PREAmBLE?" query provides information concerning
' the vertical and horizontal scaling of the waveform data.
'
' With the preamble information you can then use the
' ":WAVEFORM:DATA?" query and read the data block in the
' correct format.

' WAVE_FORMAT - Sets the data transmission mode for waveform
' data output. This command controls how the data is
' formatted when sent from the oscilloscope and can be set
' to WORD or BYTE format.

' Set waveform format to BYTE.
myScope.DoCommand(":WAVEform:FORMat BYTE")

' WAVE_POINTS - Sets the number of points to be transferred.
' The number of time points available is returned by the
' "ACQUIRE:POINTS?" query. This can be set to any binary
' fraction of the total time points available.
myScope.DoCommand(":WAVEform:POINTs 1000")

' GET_PREAmBLE - The preamble contains all of the current
' WAVEFORM settings returned in the form <preamble block><NL>
' where the <preamble block> is:
'     FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'     TYPE        : int16 - 0 = NORMAL, 1 = PEAK DETECT,
'                   2 = AVERAGE.
'     POINTS      : int32 - number of data points transferred.
'     COUNT       : int32 - 1 and is always 1.
'     XINCREMENT  : float64 - time difference between data
'                           points.
'     XORIGIN     : float64 - always the first data point in
'                           memory.
'     XREFERENCE  : int32 - specifies the data point associated
'                           with the x-origin.
'     YINCREMENT  : float32 - voltage difference between data
'                           points.

```

```

'   YORIGIN      : float32 - value of the voltage at center
'                   screen.
'   YREFERENCE   : int32 - data point where y-origin occurs.

Console.WriteLine("Reading preamble.")
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryValues(":WAVEform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
Console.WriteLine("Preamble FORMat: {0:e}", fFormat)

Dim fType As Double = fResultsArray(1)
Console.WriteLine("Preamble TYPE: {0:e}", fType)

Dim fPoints As Double = fResultsArray(2)
Console.WriteLine("Preamble POINTs: {0:e}", fPoints)

Dim fCount As Double = fResultsArray(3)
Console.WriteLine("Preamble COUNT: {0:e}", fCount)

Dim fXincrement As Double = fResultsArray(4)
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

Dim fXorigin As Double = fResultsArray(5)
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

Dim fXreference As Double = fResultsArray(6)
Console.WriteLine("Preamble XREFerence: {0:e}", fXreference)

Dim fYincrement As Double = fResultsArray(7)
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement)

Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVEform:SOURce" command.

' READ_WAVE_DATA - The wave data consists of two parts: the
' header, and the actual waveform data followed by a
' New Line (NL) character. The query data has the following
' format:
'
'   <header><waveform data block><NL>
'
' Where:
'
'   <header> = #800002048      (this is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.
' The size can vary depending on the number of points acquired
' for the waveform which can be set using the

```



```

' ":WAVEFORM:POINTS" command. You may then read that number
' of bytes from the oscilloscope; then, read the following NL
' character to terminate the query.

' Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVEform:DATA?")
nBytes = ResultsArray.Length
Console.WriteLine("Read waveform data ({0} bytes).", nBytes)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < nBytes
    Console.WriteLine("Data point {0:d} = {1:f6} Volts at " + _
        "{2:f10} Seconds", i, _
        (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
        fYorigin, (CSng(i) - fXreference) * fXincrement + fXorigin)
    i = i + (nBytes / 20)
End While

' SAVE_WAVE_DATA - saves the waveform data to a CSV format
' file named "waveform.csv".
If File.Exists("c:\scope\data\waveform.csv") Then
    File.Delete("c:\scope\data\waveform.csv")
End If

Dim writer As StreamWriter = _
    File.CreateText("c:\scope\data\waveform.csv")
For index As Integer = 0 To nBytes - 1
    writer.WriteLine("{0:E}, {1:f6}", _
        (CSng(index) - fXreference) * fXincrement + fXorigin, _
        (CSng(ResultsArray(index)) - fYreference) * fYincrement + _
        fYorigin)
Next
writer.Close()
Console.WriteLine("Waveform data ({0} points) written to " + _
    "c:\scope\data\waveform.csv.", nBytes)
End Sub
End Class

Class VisaComInstrument
    Private m_ResourceManager As ResourceManagerClass
    Private m_IoObject As FormattedIO488Class
    Private m_strVisaAddress As String

```

```

' Constructor.
Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA COM IO object.
    OpenIo()

    ' Clear the interface.
    m_IoObject.IO.Clear()
End Sub

Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    m_IoObject.WriteString(strCommand, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(strCommand)
End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result string.
    Dim strResults As String
    strResults = m_IoObject.ReadString()

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return results string.
    Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result number.
    Dim fResult As Double
    fResult = _
        CDb1(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return result number.
    Return fResult
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the result numbers.
    Dim fResultsArray As Double()

```

```

fResultsArray = _
    m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

' Check for instrument errors.
CheckForInstrumentErrors(strQuery)

' Return result numbers.
Return fResultsArray
End Function

Public _
    Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
        ' Send the query.
        m_IoObject.WriteString(strQuery, True)

        ' Get the results array.
        Dim ResultsArray As Byte()
        ResultsArray = _
            m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
                False, True)

        ' Check for instrument errors.
        CheckForInstrumentErrors(strQuery)

        ' Return results array.
        Return ResultsArray
    End Function

Public _
    Sub DoCommandIEEEBlock(ByVal strCommand As String, _
        ByVal dataArray As Byte())
        ' Send the command.
        m_IoObject.WriteIEEEBlock(strCommand, dataArray, True)

        ' Check for instrument errors.
        CheckForInstrumentErrors(strCommand)
    End Sub

Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    Dim strInstrumentError As String
    Dim bFirstError As Boolean = True

    ' Repeat until all errors are displayed.
    Do
        ' Send the ":SYSTem:ERRor?" query, and get the result string.
        m_IoObject.WriteString(":SYSTem:ERRor?", True)
        strInstrumentError = m_IoObject.ReadString()

        ' If there is an error, print it.
        If strInstrumentError.ToString() <> "+0,"No error"" " _
            & Chr(10) & "" Then
            If bFirstError Then
                ' Print the command that caused the error.
                Console.WriteLine("ERROR(s) for command '{0}': ", _
                    strCommand)
                bFirstError = False
            End If
        End If
    Loop

```

```

        Console.WriteLine(strInstrumentError)
    End If
    Loop While strInstrumentError.ToString() <> "+0,""No error"" _
        & Chr(10) & ""
End Sub

Private Sub OpenIo()
    m_ResourceManager = New ResourceManagerClass()
    m_IoObject = New FormattedIO488Class()

    ' Open the default VISA COM IO object.
    Try
        m_IoObject.IO = _
            DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
                AccessMode.NO_LOCK, 0, ""), IMessage)
    Catch e As Exception
        Console.WriteLine("An error occurred: {0}", e.Message)
    End Try
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
    m_IoObject.IO.Timeout = nSeconds * 1000
End Sub

Public Sub Close()
    Try
        m_IoObject.IO.Close()
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_IoObject)
    Catch
    End Try

    Try
        Marshal.ReleaseComObject(m_ResourceManager)
    Catch
    End Try
End Sub
End Class
End Namespace

```

# Index

## Symbols

+9.9E+37, infinity representation, 696  
+9.9E+37, measurement error, 294

## Numerics

0 (zero) values in waveform data, 533  
1 (one) values in waveform data, 533  
10 MHz REF BNC, enabling/disabling, 398  
10 MHz reference signal, 180  
54620/54640 series oscilloscopes, command differences from, 33  
82350A GPIB interface, 4

## A

AC coupling, trigger edge, 438  
AC input coupling for specified channel, 208  
accumulate activity, 137  
acknowledge, 629  
ACQuire commands, 172  
acquire data, 145, 185  
acquire mode on autoscale, 141  
acquire reset conditions, 123  
acquire sample rate, 184  
ACQuire subsystem, 53  
acquired data points, 179  
acquisition anti-alias control, 174  
acquisition count, 176  
acquisition mode, 172, 178, 550  
acquisition type, 172, 185  
active edges, 137  
active printer, 270  
activity logic levels, 137  
activity on digital channels, 137  
add function, 545  
add math function, 259  
add math function as g(t) source, 255  
ADDRess commands, 560  
address field size, IIC serial decode, 374  
address, IIC trigger pattern, 465  
Addresses softkey, 40  
AER (Arm Event Register), 138, 156, 158, 659  
Agilent Connection Expert, 41  
Agilent Interactive IO application, 45  
Agilent IO Control icon, 41  
Agilent IO Libraries Suite, 4, 37, 50, 52  
Agilent IO Libraries Suite, installing, 38  
ALB waveform data format, 355  
alphabetical list of commands, 559  
amplitude, vertical, 321  
analog channel coupling, 208

analog channel display, 209  
analog channel impedance, 210  
analog channel input, 596  
analog channel inversion, 211  
analog channel labels, 212, 236  
analog channel offset, 213  
analog channel protection lock, 389  
analog channel range, 219  
analog channel scale, 220  
analog channel source for glitch, 463  
analog channel units, 221  
analog channels only oscilloscopes, 4  
analog probe attenuation, 214  
analog probe sensing, 597  
analog probe skew, 216, 595  
analyzing captured data, 49  
angle brackets, 105  
annotate channels, 212  
anti-alias control, 174  
AREA commands, 560  
area for hardcopy print, 269  
area for saved image, 347  
Arm Event Register (AER), 138, 156, 158, 659  
arrange waveforms, 599  
ASCII format, 535  
ASCII format for data transfer, 525  
ASCII string, quoted, 105  
ASCIixy waveform data format, 355  
assign channel names, 212  
attenuation factor (external trigger) probe, 244  
attenuation for oscilloscope probe, 214  
AUT option for probe sense, 597, 602  
auto trigger sweep mode, 405  
automatic measurements constants, 214  
automatic probe type detection, 597, 602  
Automation-Ready CD, 38  
autoscale, 139  
autoscale acquire mode, 141  
autoscale channels, 142  
AUToscale command, 52  
average value measurement, 322  
averaging acquisition type, 173, 525  
averaging, synchronizing with, 672

## B

bandwidth filter limits, 242  
bandwidth filter limits to 20 MHz, 207  
base value measurement, 323  
basic instrument functions, 111  
Bat On bit, 149, 151  
baud rate, 423, 476, 506  
BAUDRate commands, 560

begin acquisition, 145, 165, 167  
BHARis window for minimal spectral leakage, 266  
binary block data, 105, 390, 533  
BINary waveform data format, 355  
bit order, 507  
bit selection command, bus, 189  
bit weights, 116  
bitmap display, 233  
bits in Service Request Enable Register, 128  
bits in Standard Event Status Enable Register, 115  
bits in Status Byte Register, 130  
bits selection command, bus, 190  
blank, 143  
block data, 105, 119, 233, 390  
block response data, 56  
blocking synchronization, 667  
blocking wait, 666  
BMP (bitmap) hardcopy format, 608  
braces, 104  
built-in measurements, 49  
burst, minimum time before next, 435  
bus bit selection command, 189  
bus bits selection commands, 190  
bus clear command, 192  
BUS commands, 187  
bus commands, 188  
bus display, 193  
bus label command, 194  
bus mask command, 195  
BUSDoctor commands, 561  
button disable, 388  
BWLimit commands, 561  
byte format for data transfer, 525, 535  
BYTeorder, 531

## C

C#, VISA COM example, 774  
C#, VISA example, 737  
C, SiCL library example, 700  
C, VISA library example, 718  
CAL PROTECT switch, 196, 201  
calculating preshoot of waveform, 310  
calculating the waveform overshoot, 306  
calibrate, 197, 198, 201, 203  
CALibrate commands, 196  
calibrate date, 197  
calibrate introduction, 196  
calibrate label, 198  
calibrate start, 199  
calibrate status, 200

- calibrate switch, 201
- calibrate temperature, 202
- calibrate time, 203
- CAN, 418
- CAN acknowledge, 422, 629
- CAN baud rate, 423
- CAN commands, 561
- CAN frame counters, reset, 366
- CAN id pattern, 420
- CAN signal definition, 630
- CAN source, 424
- CAN trigger, 419, 425
- CAN trigger commands, 416
- CAN trigger pattern id mode, 421
- CAN triggering, 405
- capture data, 145
- capturing data, 48
- CDISplay, 144
- center frequency set, 252, 253
- center of screen, 558
- center reference, 399
- center screen, vertical value at, 258
- channel, 171, 212, 592, 594
- CHANnel commands, 204, 205
- channel coupling, 208
- channel display, 209
- channel input impedance, 210
- channel inversion, 211
- channel label, 212, 593
- channel labels, 235, 236
- channel numbers, 599
- channel overload, 218
- channel probe ID, 245
- channel protection, 218
- channel reset conditions, 123
- channel selected to produce trigger, 463, 502
- channel signal type, 217
- channel skew for oscilloscope probe, 216, 595
- channel status, 168, 599
- channel threshold, 594
- channel vernier, 222
- channel, stop displaying, 143
- channels to autoscale, 142
- channels, how autoscale affects, 139
- characters to display, 386
- classes of input signals, 266
- classifications, command, 676
- clear, 232
- clear bus command, 192
- CLEar commands, 562
- clear cumulative edge variables, 592
- clear display, 144
- clear markers, 295, 613
- clear measurement, 295, 613
- clear message queue, 113
- Clear method, 51
- clear screen, 600
- clear status, 113
- clear waveform area, 230
- clipped high waveform data value, 533
- clipped low waveform data value, 533
- clock, 468, 490, 491, 495
- CLOCK commands, 562
- CLS (Clear Status), 113
- CME (Command Error) status bit, 115, 117
- CMOS threshold voltage for digital channels, 229, 594
- CMOS trigger threshold voltage, 632
- code, \*RST, 125
- code, :ACQuire:COMPLet, 175
- code, :ACQuire:SEGMENTed, 182
- code, :ACQuire:TYPE, 186
- code, :AUToscale, 140
- code, :CHANnel:LABel, 212
- code, :CHANnel:PROBe, 214
- code, :CHANnel:RANGe, 219
- code, :DIGitize, 145
- code, :DISPlay:DATA, 234
- code, :DISPlay:LABel, 235
- code, :DISPlay:ORDer, 599
- code, :MEASure:PERiod, 316
- code, :MEASure:TEDGe, 318
- code, :POD:THReshold, 336
- code, :RUN/:STOP, 165
- code, :SYSTem:SETup, 390
- code, :TIMebase:DElay, 628
- code, :TIMebase:MODE, 395
- code, :TIMebase:RANGe, 397
- code, :TIMebase:REfERENCE, 399
- code, :TRIGger:MODE, 411
- code, :TRIGger:SLOPe, 441
- code, :TRIGger:SOURce, 442
- code, :VIEW and :BLANK, 171
- code, :WAVEform, 546
- code, :WAVEform:DATA, 533
- code, :WAVEform:POINts, 537
- code, :WAVEform:PREAmble, 541
- code, :WAVEform:SEGMENTed, 182
- code, SICL library example in C, 700
- code, SICL library example in Visual Basic, 709
- code, VISA COM library example in C#, 774
- code, VISA COM library example in Visual Basic, 764
- code, VISA COM library example in Visual Basic .NET, 785
- code, VISA library example in C, 718
- code, VISA library example in C#, 737
- code, VISA library example in Visual Basic, 727
- code, VISA library example in Visual Basic .NET, 751
- colon, root commands prefixed by, 136
- color palette for hardcopy, 274
- color palette for image, 351
- Comma Separated Values (CSV) hardcopy format, 608
- Comma Separated Values (CSV) waveform data format, 355
- command classifications, 676
- command differences from 54620/54640 series oscilloscopes, 33
- command errors detected in Standard Event Status, 117
- command header, 678
- command headers, common, 680
- command headers, compound, 679
- command headers, simple, 679
- command strings, valid, 677
- command tree, 681
- commands by subsystem, 107
- commands in alphabetical order, 559
- commands quick reference, 61
- commands sent over interface, 111
- commands, more about, 675
- commands, obsolete and discontinued, 587
- common (\*) commands, 108, 109, 111
- common command headers, 680
- completion criteria for an acquisition, 175, 176
- compound command headers, 679
- compound header, 693
- computer control examples, 699
- conditions for external trigger, 240
- conditions, reset, 123
- configurations, oscilloscope, 119, 122, 126, 390
- Configure softkey, 40
- connect oscilloscope, 39
- connect sampled data points, 598
- constants for making automatic measurements, 214
- constants for scaling display factors, 214
- constants for setting trigger levels, 214
- Control softkey, 39, 40
- controller initialization, 48
- copy display, 164
- core commands, 676
- count, 482, 532
- COUNt commands, 562
- count values, 176
- count, Nth edge of burst, 434
- counter, 296
- coupling, 438
- COUPling commands, 563
- coupling for channels, 208
- CSV (Comma Separated Values) hardcopy format, 608
- CSV (Comma Separated Values) waveform data format, 355
- cumulative edge activity, 592
- current logic levels on digital channels, 137
- current oscilloscope configuration, 119, 122, 126, 390
- current probe, 221, 249
- cursor mode, 279
- cursor position, 280, 282, 284, 285, 287
- cursor readout, 614, 618, 619
- cursor reset conditions, 123
- cursor source, 281, 283
- cursor time, 614, 618, 619
- cursors track measurements, 314
- cursors, how autoscale affects, 139
- cursors, X1, X2, Y1, Y2, 278
- cycle base, FLEXray time trigger, 450
- cycle count baase, FLEXray frame trigger, 446
- cycle count repetition, FLEXray frame trigger, 447
- cycle measured, 302

cycle repetition, FLEXray time trigger, 451  
cycle time, 308

## D

D- source, 519  
D+ source, 520  
data, 418, 466, 469, 493, 496, 533  
data 2, 467  
DATA commands, 563  
data conversion, 525  
data displayed, 233  
data format for transfer, 525  
data output order, 531  
data pattern length, 419  
data pattern width, 494  
data point index, 555  
data points, 179  
data required to fill time buckets, 175  
data structures, status reporting, 645  
data transfer, 233  
data, erasing, 144  
data, saving and recalling, 230  
DATE commands, 563  
date, calibration, 197  
date, system, 385  
dB versus frequency, 252  
DC coupling for edge trigger, 438  
DC input coupling for specified channel, 208  
dc RMS measured on waveform, 328  
DDE (Device Dependent Error) status bit, 115, 117  
decision chart, status reporting, 663  
default conditions, 123  
define channel labels, 212  
define glitch trigger, 461  
define logic thresholds, 594  
define measurement, 298  
define measurement source, 315  
define trigger, 413, 428, 429, 430, 432, 462, 483  
defined as, 104  
definite-length block query response, 56  
definite-length block response data, 105  
DEFinition commands, 564  
DElay commands, 564  
delay measured to calculate phase, 309  
delay measurement, 298  
delay measurements, 317  
delay parameters for measurement, 300  
delay, how autoscale affects, 139  
delayed time base, 395, 628  
delayed time base mode, how autoscale affects, 139  
delayed window horizontal scale, 404  
delta time, 614  
delta voltage measurement, 623  
delta X cursor, 278  
delta Y cursor, 278  
DeskJet, 606  
destination, 238  
detecting probe types, 597, 602

device for hardcopy, 606  
device-defined error queue clear, 113  
differences from 54620/54640 series  
    oscilloscope commands, 33  
differential signal type, 217, 246  
differentiate math function, 177, 252, 259, 545  
DIFFerentiate source for function, 264, 603  
digital channel commands, 223, 225, 226, 227, 229  
digital channel data, 525  
digital channel labels, 236  
digital channel order, 599  
digital channel source for glitch trigger, 463  
digital channels, 4  
digital channels, activity and logic levels  
    on, 137  
digital channels, groups of, 333, 334, 336  
DIGital commands, 223  
digital pod, stop displaying, 143  
digital reset conditions, 123  
digitize channels, 145  
DIGitize command, 49, 53  
digits, 105  
disable anti-alias mode, 177  
disable front panel, 388  
disable function, 604  
disabling calibration, 201  
disabling channel display, 209  
disabling status register bits, 114, 127  
discontinued and obsolete commands, 587  
display channel labels, 235  
display clear, 232  
DISPlay commands, 230, 564  
display commands introduction, 230  
display connect, 598  
display data, 233  
display date, 385  
display factors scaling, 214  
display for channels, 209  
display frequency span, 265  
display measurements, 293, 314  
display order, 599  
display persistence, 237  
display reference, 396, 399  
display reset conditions, 123  
display serial number, 166  
display source, 238  
display vectors, 239  
display wave position, 599  
display, clearing, 144  
display, oscilloscope, 225, 237, 238, 239, 254, 334, 386  
display, serial decode bus, 369  
displaying a baseline, 415  
displaying unsynchronized signal, 415  
DNS IP, 39  
domain, 39  
Domain softkey, 40  
driver, printer, 611  
DSO models, 4  
duplicate mnemonics, 693  
duration, 428, 429, 432

duration for glitch trigger, 457, 458, 462  
duration pattern, 430  
duration qualifier, trigger, 428, 429, 431  
DURation trigger commands, 427  
duration triggering, 405  
duty cycle measurement, 49, 293, 302

## E

EBURst trigger commands, 433  
ECL channel threshold, 594  
ECL threshold voltage for digital channels, 229  
ECL trigger threshold voltage, 632  
edge, 483  
edge activity, 592  
edge counter, 482  
edge counter, Nth edge of burst, 434  
edge coupling, 438  
edge define, 413, 483  
edge fall time, 303  
edge parameter for delay measurement, 300  
edge preshoot measured, 310  
edge rise time, 312  
edge slope, 441  
edge source, 442  
EDGE trigger commands, 437  
edge triggering, 405  
edges (activity) on digital channels, 137  
edges in measurement, 298  
ellipsis, 105  
enable channel labels, 235  
enabling calibration, 201  
enabling channel display, 209  
enabling status register bits, 114, 127  
end of string (EOS) terminator, 678  
end of text (EOT) terminator, 678  
end or identify (EOI), 678  
enter pattern, 413  
EOI (end or identify), 678  
EOS (end of string) terminator, 678  
EOT (end of text) terminator, 678  
Epson, 606  
equivalent-time acquisition mode, 173, 178  
erase data, 144, 232  
erase functions, 144  
erase measurements, 613  
erase screen, 600  
ERRor commands, 565  
error frame count (CAN), 364  
error frame count (UART), 379  
error messages, 387, 635  
error number, 387  
error queue, 387, 656  
error, measurement, 293  
ESB (Event Status Bit), 128, 130  
ESE (Standard Event Status Enable Register), 114, 655  
ESR (Standard Event Status Register), 116, 654  
event status conditions occurred, 130  
Event Status Enable Register (ESE), 114, 655  
Event Status Register (ESR), 116, 170, 654  
example code, \*RST, 125



example code, :ACQuire:COMPlEte, 175  
 example code, :ACQuire:SEGMeNted, 182  
 example code, :ACQuire:TYpE, 186  
 example code, :AUToscale, 140  
 example code, :CHANnel:LABel, 212  
 example code, :CHANnel:PROBe, 214  
 example code, :CHANnel:RANGe, 219  
 example code, :DIGitize, 145  
 example code, :DISPlay:DATA, 234  
 example code, :DISPlay:LABel, 235  
 example code, :DISPlay:ORDer, 599  
 example code, :MEASure:PERiod, 316  
 example code, :MEASure:TEDGe, 318  
 example code, :POD:THReshold, 336  
 example code, :RUN/:STOP, 165  
 example code, :SYSTem:SEtUp, 390  
 example code, :TIMebase:DELAy, 628  
 example code, :TIMebase:MODE, 395  
 example code, :TIMebase:RANGe, 397  
 example code, :TIMebase:REFeRence, 399  
 example code, :TRIGger:MODE, 411  
 example code, :TRIGger:SLOPe, 441  
 example code, :TRIGger:SOURce, 442  
 example code, :VIEW and :BLANK, 171  
 example code, :WAVEform, 546  
 example code, :WAVEform:DATA, 533  
 example code, :WAVEform:POINts, 537  
 example code, :WAVEform:PREAmble, 541  
 example code, :WAVEform:SEGMeNted, 182  
 example programs, 4, 699  
 EXE (Execution Error) status bit, 115, 117  
 execution error detected in Standard Event Status, 117  
 exponential notation, 104  
 external glitch trigger source, 463  
 external range, 248  
 external trigger, 240, 243, 244, 442, 601  
 EXTeRnal trigger commands, 240  
 external trigger input impedance, 243, 601  
 EXTeRnal trigger level, 439  
 external trigger overload, 247  
 external trigger probe attenuation factor, 244  
 external trigger probe ID, 245  
 external trigger probe sensing, 602  
 external trigger protection, 247  
 external trigger signal type, 246  
 EXTeRnal trigger source, 442  
 external trigger units, 249

## F

FACTors commands, 565  
 failure, self test, 132  
 fall time measurement, 293, 303  
 falling edge, 413, 483  
 Fast Fourier Transform (FFT) functions, 252, 253, 261, 264, 265, 266, 603  
 FF values in waveform data, 533  
 FFT (Fast Fourier Transform) functions, 252, 253, 261, 264, 265, 266, 603  
 FFT (Fast Fourier Transform) operation, 259, 545

FFT math function, 177  
 fifty ohm impedance, disable setting, 389  
 FILeName commands, 565  
 filename for hardcopy, 607  
 filename for recall, 339  
 filename for save, 345  
 filter for frequency reject, 440  
 filter for high frequency reject, 409  
 filter for noise reject, 412  
 filter used to limit bandwidth, 207, 242  
 filters to Fast Fourier Transforms, 266  
 find stage in sequence trigger, 484  
 fine horizontal adjustment (vernier), 401  
 fine vertical adjustment (vernier), 222  
 finish pending device operations, 120  
 first point displayed, 555  
 FLATtop window for amplitude measurements, 266  
 FLEXray commands, 565  
 FlexRay frame counters, reset, 371  
 FLEXray trigger, 454  
 FLEXray trigger commands, 443  
 FlexRay triggering, 405  
 format, 535, 540  
 FORMat commands, 566  
 format for block data, 119  
 format for generic video, 499, 503  
 format for hardcopy, 605, 608  
 format for image, 349  
 format for waveform data, 355  
 FormattedIO488 object, 51  
 formfeed for hardcopy, 267, 272  
 frame, 497  
 FRAME commands, 566  
 frame counters (CAN), error, 364  
 frame counters (CAN), overload, 365  
 frame counters (CAN), reset, 366  
 frame counters (CAN), total, 367  
 frame counters (FlexRay), null, 370, 372  
 frame counters (FlexRay), reset, 371  
 frame counters (FlexRay), total, 373  
 frame counters (UART), error, 379  
 frame counters (UART), reset, 380  
 frame counters (UART), Rx frames, 381  
 frame counters (UART), Tx frames, 382  
 frame ID, FLEXray frame trigger, 448  
 frame type, FLEXray frame trigger, 449  
 framing, 492  
 FRAMing commands, 566  
 frequency measurement, 49, 293, 304  
 frequency resolution, 266  
 frequency span of display, 265  
 frequency versus dB, 252  
 front panel mode, 415  
 front panel Single key, 167  
 front panel Stop key, 169  
 front-panel lock, 388  
 full-scale horizontal time, 397, 403  
 full-scale vertical axis defined, 260  
 function, 171, 253, 254, 258, 259, 260, 261, 262, 264, 265, 266, 603, 604  
 FUNCtion commands, 250

function memory, 168  
 function turned on or off, 604  
 functions, 545  
 functions, erasing, 144

## G

g(t) source, first input channel, 256  
 g(t) source, math operation, 255  
 g(t) source, second input channel, 257  
 gateway IP, 39  
 general trigger commands, 408  
 GENeric, 499, 503  
 generic video format, 499, 503  
 glitch duration, 462  
 glitch qualifier, 461  
 glitch source, 463  
 GLITCh trigger commands, 455  
 glitch trigger duration, 457  
 glitch trigger polarity, 460  
 glitch trigger source, 457  
 GOFT commands, 567  
 GPIB interface, 39, 40  
 graphics, 233  
 graticule area for hardcopy print, 269  
 graticule area for saved image, 347  
 graticule colors, invert for hardcopy, 273, 610  
 graticule colors, invert for image, 350  
 graticule data, 233  
 grayscale palette for hardcopy, 274  
 grayscale palette for image, 351  
 grayscale on hardcopy, 609  
 greater than qualifier, 461  
 greater than time, 428, 432, 457, 462  
 GREATERthan commands, 567  
 groups of digital channels, 333, 334, 336, 594

## H

HANNing window for frequency resolution, 266  
 hardcopy, 164, 267  
 HARDcopy commands, 267  
 hardcopy device, 606  
 hardcopy factors, 271, 348  
 hardcopy filename, 607  
 hardcopy format, 605, 608  
 hardcopy formfeed, 272  
 hardcopy grayscale, 609  
 hardcopy invert graticule colors, 273, 610  
 hardcopy palette, 274  
 hardcopy print, area, 269  
 hardcopy printer driver, 611  
 hardware event condition register, 149  
 Hardware Event Condition Register (:HWERegister:CONDition), 149  
 Hardware Event Condition Register (:OPERRegister:CONDition), 661  
 Hardware Event Enable Register (HWEEnable), 147  
 hardware event event register, 151



Hardware Event Event Register  
(:HWERegister[:EVENT]), 151, 660  
header, 678  
high-frequency reject filter, 409, 440  
high-resolution acquisition type, 173  
hold until operation complete, 120  
holdoff time, 410  
holes in waveform data, 533  
horizontal adjustment, fine (vernier), 401  
horizontal position, 402  
horizontal scale, 400, 404  
horizontal scaling, 540  
horizontal time, 397, 403, 614  
hostname, 39  
HWEenable (Hardware Event Enable Register), 147  
HWERegister:CONDition (Hardware Event Condition Register), 149, 661  
HWERegister[:EVENT] (Hardware Event Event Register), 151, 660

## I

I/O softkey, 39, 40  
I1080L50HZ, 499, 503  
I1080L60HZ, 499, 503  
ID commands, 567  
id mode, 421  
identification number, 118  
identification of options, 121  
identifier, 420  
identifier, LIN, 474  
idle, 435  
IDLE commands, 568  
idle until operation complete, 120  
IDN (Identification Number), 118  
IEEE 488.2 standard, 111  
IGColors commands, 568  
IIC address, 465  
IIC clock, 468  
IIC commands, 568  
IIC data, 466, 469  
IIC data 2, 467  
IIC serial decode address field size, 374  
IIC trigger commands, 464  
IIC trigger qualifier, 470  
IIC trigger type, 471  
IIC triggering, 405  
IMAGe commands, 568  
image format, 349  
image invert graticule colors, 350  
image memory, 168, 238  
image palette, 351  
image, recall, 340  
image, save, 346  
image, save with inksaver, 350  
impedance, 210  
IMPedance commands, 568  
impedance for external trigger input, 243, 601  
infinity representation, 696  
initialization, 48, 51  
initialize, 123

initialize label list, 236  
initiate acquisition, 145  
inksaver, save image with, 350  
input, 243, 601  
input coupling for channels, 208  
input impedance for channels, 210, 596  
input impedance for external trigger, 243, 601  
input inversion for specified channel, 211  
insert label, 212  
installed options identified, 121  
instruction header, 678  
instrument number, 118  
instrument options identified, 121  
instrument requests service, 130  
instrument serial number, 166  
instrument settings, 267  
instrument status, 58  
instrument type, 118  
integrate math function, 252, 259, 545  
INTEgrate source for function, 264, 603  
INTERN files, 238  
internal low-pass filter, 207, 242  
introduction to :ACQuire commands, 172  
introduction to :BUS commands, 188  
introduction to :CALibrate commands, 196  
introduction to :CHANnel commands, 205  
introduction to :DIGital commands, 223  
introduction to :DISPlay commands, 230  
introduction to :EXTernal commands, 240  
introduction to :FUNction commands, 252  
introduction to :HARDcopy commands, 267  
introduction to :MARKer commands, 278  
introduction to :MEASure commands, 293  
introduction to :POD commands, 333  
introduction to :RECall commands, 338  
introduction to :SAVE commands, 344  
introduction to :SBUS commands, 358  
introduction to :SYSTem commands, 384  
introduction to :TIMebase commands, 394  
introduction to :TRIGger commands, 405  
introduction to :WAVEform commands, 525  
introduction to common (\*) commands, 111  
introduction to root (:) commands, 136  
invert graticule colors for hardcopy, 273, 610  
invert graticule colors for image, 350  
inverting input for channels, 211  
IO library, referencing, 50  
IP address, 39  
IP Options softkey, 40

## K

key disable, 388  
key press detected in Standard Event Status Register, 117  
knob disable, 388  
known state, 123

## L

label, 226, 593

label command, bus, 194  
LABel commands, 568  
label list, 212, 236  
labels, 212, 235, 236  
labels to store calibration information, 198  
labels, specifying, 230  
LAN interface, 39, 42  
LAN Settings softkey, 40  
language for program examples, 47  
LaserJet, 606  
leakage into peak spectrum, 266  
learn string, 119, 390  
least significant byte first, 531  
left reference, 399  
legal values for channel offset, 213  
legal values for frequency span, 265  
legal values for offset, 258  
LENGth commands, 568  
length for waveform data, 356  
less than qualifier, 461  
less than time, 429, 432, 458, 462  
LESSthan commands, 569  
LEVel commands, 569  
level for trigger voltage, 439, 459  
LF coupling, 438  
license information, 121  
limits for line number, 499  
LIN acknowledge, 475  
LIN baud rate, 476  
LIN identifier, 474  
LIN serial decode bus parity bits, 375  
LIN source, 477  
LIN standard, 478  
LIN sync break, 479  
LIN trigger, 480  
LIN trigger commands, 473  
LIN trigger definition, 631  
LIN triggering, 405  
line glitch trigger source, 463  
line number for TV trigger, 499  
line terminator, 104  
LINE trigger level, 439  
LINE trigger source, 442  
list of channel labels, 236  
load utilization (CAN), 368  
local lockout, 388  
lock, 388  
LOCK commands, 569  
lock, analog channel protection, 389  
lockout message, 388  
logic level activity, 592  
long form, 678  
lower threshold, 308  
lower threshold voltage for measurement, 612  
lowercase characters in commands, 677  
low-frequency reject filter, 440  
low-pass filter used to limit bandwidth, 207, 242  
LRN (Learn Device Setup), 119  
lsbfirst, 531

## M

magnitude of occurrence, 319  
 main sweep range, 402  
 main time base mode, 395  
 making measurements, 293  
 MAN option for probe sense, 597, 602  
 manual cursor mode, 279  
 MARKer commands, 277  
 marker mode, 285  
 marker position, 286  
 marker readout, 618, 619  
 marker set for voltage measurement, 624, 625  
 marker sets start time, 615  
 marker time, 614  
 markers for delta voltage measurement, 623  
 markers track measurements, 314  
 markers, command overview, 278  
 markers, mode, 279  
 markers, time at start, 619  
 markers, time at stop, 618  
 markers, X delta, 284  
 markers, X1 position, 280  
 markers, X1Y1 source, 281  
 markers, X2 position, 282  
 markers, X2Y2 source, 283  
 markers, Y delta, 287  
 markers, Y1 position, 285  
 markers, Y2 position, 286  
 mask, 114, 127, 413, 430  
 mask command, bus, 195  
 master summary status bit, 130  
 math function, stop displaying, 143  
 math operations, 252  
 MAV (Message Available), 113, 128, 130  
 maximum duration, 428, 429, 458  
 maximum position, 396  
 maximum range for delayed window, 403  
 maximum scale for delayed window, 404  
 maximum vertical value measurement, 324  
 maximum vertical value, time of, 331, 616  
 MEASure commands, 288  
 measure overshoot, 306  
 measure period, 308  
 measure phase between channels, 309  
 measure preshoot, 310  
 measure start voltage, 624  
 measure stop voltage, 625  
 measure value at a specified time, 329  
 measure value at top of waveform, 330  
 measurement error, 293  
 measurement setup, 293, 315  
 measurement source, 315  
 measurements, average value, 322  
 measurements, base value, 323  
 measurements, built-in, 49  
 measurements, clear, 295, 613  
 measurements, command overview, 293  
 measurements, counter, 296  
 measurements, dc RMS, 328  
 measurements, definition setup, 298  
 measurements, delay, 300

measurements, duty cycle, 302  
 measurements, fall time, 303  
 measurements, frequency, 304  
 measurements, how autoscale affects, 139  
 measurements, lower threshold level, 612  
 measurements, maximum vertical value, 324  
 measurements, maximum vertical value, time of, 331, 616  
 measurements, minimum vertical value, 325  
 measurements, minimum vertical value, time of, 332, 617  
 measurements, overshoot, 306  
 measurements, period, 308  
 measurements, phase, 309  
 measurements, preshoot, 310  
 measurements, pulse width, negative, 305  
 measurements, pulse width, positive, 311  
 measurements, ratio of AC RMS values, 327  
 measurements, resetting, 144  
 measurements, rise time, 312  
 measurements, show, 314  
 measurements, source channel, 315  
 measurements, standard deviation, 313  
 measurements, start marker time, 618  
 measurements, stop marker time, 619  
 measurements, thresholds, 615  
 measurements, time between start and stop markers, 614  
 measurements, time between trigger and edge, 317  
 measurements, time between trigger and vertical value, 319  
 measurements, time between trigger and voltage level, 620  
 measurements, upper threshold value, 622  
 measurements, vertical amplitude, 321  
 measurements, vertical peak-to-peak, 326  
 measurements, voltage difference, 623  
 memory setup, 126, 390  
 merge, 153  
 message available bit, 130  
 message available bit clear, 113  
 message displayed, 130  
 message error, 635  
 message queue, 653  
 messages ready, 130  
 midpoint of thresholds, 308  
 minimum duration, 428, 429, 432, 457  
 minimum vertical value measurement, 325  
 minimum vertical value, time of, 332, 617  
 mixed-signal oscilloscopes, 4  
 mnemonics, duplicate, 693  
 mode, 178, 185, 279, 395, 500  
 MODE commands, 571  
 mode, serial decode, 376  
 model number, 118  
 models, oscilloscope, 3  
 modes for triggering, 411  
 Modify softkey, 40  
 monochrome palette for image, 351  
 most significant byte first, 531  
 move, 252

move cursors, 618, 619  
 msbfirst, 531  
 MSG (Message), 128, 130  
 MSO models, 4  
 MSS (Master Summary Status), 130  
 multiple commands, 693  
 multiple queries, 57  
 multiply math function, 252, 259, 545  
 multiply math function as g(t) source, 255

## N

name channels, 212  
 name list, 236  
 negative glitch trigger polarity, 460  
 negative pulse width, 305  
 negative pulse width measurement, 49  
 negative slope, 441, 490  
 negative slope, Nth edge in burst, 436  
 negative TV trigger polarity, 501  
 new line (NL) terminator, 104, 678  
 NL (new line) terminator, 104, 678  
 noise reject filter, 412  
 non-core commands, 676  
 non-interlaced GENeric mode, 503  
 non-volatile memory, label list, 194, 226, 236  
 normal acquisition type, 172, 525  
 normal trigger sweep mode, 405  
 notices, 2  
 NR1 number format, 104  
 NR3 number format, 104  
 Nth edge in a burst idle, 435  
 Nth edge in burst slope, 436  
 Nth edge of burst counter, 434  
 NTSC, 499, 503  
 null frame count (FlexRay), 370  
 NULL string, 386  
 number format, 104  
 number of points, 179, 536, 538  
 number of time buckets, 536, 538  
 numeric variables, 56  
 numeric variables, reading query results into multiple, 58  
 nwidth, 305

## O

obsolete and discontinued commands, 587  
 obsolete commands, 676  
 occurrence reported by magnitude, 620  
 offset, 252  
 OFFSet commands, 571  
 offset value for channel voltage, 213  
 offset value for selected function, 258  
 one values in waveform data, 533  
 OPC (Operation Complete) command, 120  
 OPC (Operation Complete) status bit, 115, 117  
 OPEE (Operation Status Enable Register), 154  
 Open method, 51  
 operating configuration, 119, 390  
 operating state, 126

OPERation commands, 571  
 operation complete, 120  
 operation status condition register, 156  
 Operation Status Condition Register  
 (:OPERRegister:CONDition), 156, 658  
 operation status conditions occurred, 130  
 Operation Status Enable Register (OPEE), 154  
 operation status event register, 158  
 Operation Status Event Register  
 (:OPERRegister[:EVENT]), 158, 657  
 operation, math, 252  
 operations for function, 259  
 OPERRegister:CONDition (Operation Status  
 Condition Register), 156, 658  
 OPERRegister[:EVENT] (Operation Status Event  
 Register), 158, 657  
 OPT (Option Identification), 121  
 optional syntax terms, 104  
 options, 121  
 order of digital channels on display, 599  
 order of output, 531  
 oscilloscope connection, opening, 51  
 oscilloscope connection, verifying, 41  
 oscilloscope external trigger, 240  
 oscilloscope models, 3  
 oscilloscope rate, 184  
 oscilloscope, connecting, 39  
 oscilloscope, initialization, 48  
 oscilloscope, operation, 4  
 oscilloscope, program structure, 48  
 oscilloscope, setting up, 39  
 oscilloscope, setup, 52  
 output messages ready, 130  
 output queue, 120, 652  
 output queue clear, 113  
 output sequence, 531  
 overlapped commands, 697  
 overload, 218, 247  
 Overload Event Enable Register (OVL), 160  
 Overload Event Register (OVLr), 162  
 overload frame count (CAN), 365  
 overload protection, 160, 162  
 overshoot of waveform, 306  
 overvoltage, 218, 247  
 OVL (Overload Event Enable Register), 160  
 OVLr (Overload Event Register), 162  
 OVLr bit, 151, 156, 158

## P

P1080L24HZ, 499, 503  
 P1080L25HZ, 499, 503  
 P480L60HZ, 499, 503  
 P720L60HZ, 499, 503  
 PAL, 499, 503  
 PALette commands, 571  
 palette for hardcopy, 274  
 palette for image, 351  
 PAL-M, 499, 503  
 parameters for delay measurement, 300  
 parametric measurements, 293  
 parity, 511

parity bits, LIN serial decode bus, 375  
 PARity commands, 572  
 parser, 136, 693  
 pass, self test, 132  
 path information, recall, 341  
 path information, save, 352  
 pattern, 413, 418, 420, 430, 465, 466, 467,  
 485, 493  
 pattern and edge, 413  
 PATtern commands, 572  
 pattern duration, 428, 429, 457, 458  
 pattern length, 419  
 pattern trigger, 413  
 pattern triggering, 405  
 pattern width, 494  
 peak detect, 185  
 peak detect acquisition type, 173, 525  
 peaks, 252  
 peak-to-peak vertical value measurement, 326  
 pending operations, 120  
 percent of waveform overshoot, 306  
 percent thresholds, 298  
 period measured to calculate phase, 309  
 period measurement, 49, 293, 308  
 persistence, waveform, 230, 237  
 phase measured between channels, 309  
 phase measurements, 317  
 pixel memory, 238  
 pixel memory, saving display to, 153  
 PLL Locked bit, 149, 156  
 pod, 333, 334, 335, 336, 545, 594  
 POD commands, 333  
 pod, stop displaying, 143  
 points, 179, 536, 538  
 POINts commands, 572  
 points in waveform data, 525  
 polarity, 501, 512  
 POLarity commands, 572  
 polarity for glitch trigger, 460  
 polling synchronization with timeout, 668  
 polling wait, 666  
 PON (Power On) status bit, 115, 117  
 position, 227, 282, 396, 402  
 POSition commands, 572  
 position cursors, 618, 619  
 position in delayed view, 402  
 position waveforms, 599  
 positive glitch trigger polarity, 460  
 positive pulse width, 311  
 positive pulse width measurement, 49  
 positive slope, 441, 490  
 positive slope, Nth edge in burst, 436  
 positive TV trigger polarity, 501  
 positive width, 311  
 preamble data, 525, 540  
 predefined logic threshold, 594  
 predefined threshold voltages, 632  
 preset conditions, 123  
 preshoot measured on waveform, 310  
 previously stored configuration, 122  
 print command, 164  
 print job, start, 276

print query, 626  
 printer, 606  
 printer driver for hardcopy, 611  
 printer hardcopy format, 608  
 printer, active, 270  
 printing, 267  
 printing in grayscale, 609  
 probe, 439  
 probe attenuation affects channel voltage  
 range, 219  
 probe attenuation factor (external trigger), 244  
 probe attenuation factor for selected  
 channel, 214  
 PROBe commands, 573  
 probe ID, 215, 245  
 probe sense for oscilloscope, 597, 602  
 probe skew value, 216, 595  
 program data, 678  
 program data syntax rules, 680  
 program initialization, 48  
 program message, 51, 111  
 program message syntax, 677  
 program message terminator, 678  
 program structure, 48  
 programming examples, 4, 699  
 protecting against calibration, 201  
 protection, 160, 162, 218, 247  
 PROTection commands, 573  
 protection lock, 389  
 pulse width, 305, 311  
 pulse width duration trigger, 457, 458, 462  
 pulse width measurement, 49, 293  
 pulse width trigger, 412  
 pulse width trigger level, 459  
 pulse width triggering, 405  
 PWD commands, 573  
 pwidth, 311

## Q

qualifier, 462  
 QUALifier commands, 573  
 qualifier, trigger duration, 428, 429, 431  
 queries, multiple, 57  
 query error detected in Standard Event  
 Status, 117  
 query responses, block data, 56  
 query responses, reading, 55  
 query results, reading into numeric  
 variables, 56  
 query results, reading into string variables, 56  
 query return values, 696  
 query setup, 267, 278, 293, 390  
 query subsystem, 188, 223, 252  
 querying setup, 205  
 querying the subsystem, 405  
 queues, clearing, 662  
 quick reference, commands, 61  
 quoted ASCII string, 105  
 QYE (Query Error) status bit, 115, 117

## R

range, 252, 403  
 RANge commands, 573  
 range for channels, 219  
 range for duration trigger, 432  
 range for external trigger, 248  
 range for full-scale vertical axis, 260  
 range for glitch trigger, 462  
 range for time base, 397  
 range of offset values, 213  
 range of reference level values, 261  
 range qualifier, 461  
 ranges, value, 105  
 rate, 184  
 ratio of AC RMS values measured between channels, 327  
 RCL (Recall), 122  
 read configuration, 119  
 read trace memory, 233  
 ReadIEEEBlock method, 51, 55, 57  
 ReadList method, 51, 55  
 ReadNumber method, 51, 55  
 readout, 614  
 ReadString method, 51, 55  
 real-time acquisition mode, 173, 178  
 recall, 122, 338, 390  
 RECall commands, 338  
 recall filename, 339  
 recall image, 340  
 recall setup, 342  
 recalling and saving data, 230  
 RECTangular window for transient signals, 266  
 reference, 252, 399  
 reference clock, 398  
 REference commands, 574  
 reference for time base, 628  
 reference level, Fast Fourier Transform (FFT) function, 261  
 reference signal (10 MHz), 180  
 reference signal mode, 398  
 registers, 116, 122, 126, 138, 147, 149, 151, 154, 156, 158, 160, 162  
 registers, clearing, 662  
 reject filter, 440  
 reject high frequency, 409  
 reject noise, 412  
 remote control examples, 699  
 remove cursor information, 279  
 remove labels, 235  
 remove message from display, 386  
 reorder channels, 139  
 repetitive acquisitions, 165  
 report errors, 387  
 report transition, 317, 319  
 reporting status, 643  
 reporting the setup, 405  
 request service, 130  
 Request-for-OPC flag clear, 113  
 reset, 123, 486  
 RESet commands, 574  
 reset conditions, 123

reset measurements, 144, 232  
 resolution of printed copy, 609  
 resource session object, 51  
 ResourceManager object, 51  
 restore configurations, 119, 122, 126, 390  
 restore labels, 235  
 restore setup, 122  
 return values, query, 696  
 returning acquisition type, 185  
 returning number of data points, 179  
 right reference, 399  
 rise time measurement, 293  
 rise time of positive edge, 312  
 rising edge, 413, 483  
 RMS value measurement, 328  
 roll time base mode, 395  
 root (:) commands, 134, 136  
 root level commands, 108  
 RQL (Request Control) status bit, 115, 117  
 RQS (Request Service), 130  
 RST (Reset), 123  
 rules, tree traversal, 693  
 rules, truncation, 678  
 run, 131, 165  
 Run bit, 156, 158  
 running configuration, 126, 390  
 Rx frame count (UART), 381  
 Rx source, 514

## S

sample rate, 184  
 sampled data, 598  
 sampled data points, 533  
 SAMPlpoint commands, 574  
 SAV (Save), 126  
 save, 126, 344  
 SAVE commands, 343  
 save filename, 345  
 save image, 346  
 save image with inksaver, 350  
 save setup, 353  
 SAVE TO INTERN, 153  
 save waveform data, 354  
 save waveforms to pixel memory, 153  
 saved image, area, 347  
 saving and recalling data, 230  
 SBUS commands, 357  
 scale, 262, 400, 404  
 SCALe commands, 575  
 scale factors output on hardcopy, 271, 348  
 scale for channels, 220  
 scale units for channels, 221  
 scale units for external trigger, 249  
 scaling display factors, 214  
 SCPI commands, 59  
 scratch measurements, 613  
 screen area for hardcopy print, 269  
 screen area for saved image, 347  
 screen data, 233  
 SECAM, 499, 503  
 seconds per division, 400  
 segment, FLEXray time trigger, 452  
 SEGmented commands, 576  
 segments, count of waveform, 543  
 segments, setting number of memory, 181  
 segments, setting the index, 182  
 segments, time tag, 544  
 select measurement channel, 315  
 self-test, 132  
 sensing a channel probe, 597  
 sensing a external trigger probe, 602  
 sensitivity of oscilloscope input, 214  
 sequence, 484, 485, 486  
 sequence trigger, 488  
 SEQuence trigger commands, 481  
 sequence triggering, 405  
 sequencer edge counter, 482  
 sequencer timer, 487  
 sequential commands, 697  
 serial clock, 468, 495  
 serial data, 469, 496  
 serial decode bus, 358  
 serial decode bus display, 369  
 serial decode mode, 376  
 serial frame, 497  
 serial number, 166  
 service request, 130  
 Service Request Enable Register (SRE), 128, 650  
 set, 123  
 set center frequency, 253  
 set conditions, 139  
 set cursors, 618, 619  
 set date, 385  
 set delay, 139  
 set thresholds, 139  
 set time, 392  
 set time/div, 139  
 set up oscilloscope, 39  
 setting digital display, 225  
 setting digital label, 194, 226  
 setting digital position, 227  
 setting digital threshold, 229  
 setting display, 254  
 setting external trigger level, 240  
 setting impedance for channels, 210  
 setting inversion for channels, 211  
 setting pod display, 334  
 setting pod size, 335  
 setting pod threshold, 336  
 settings, 122, 126  
 settings, instrument, 267  
 setup, 173, 188, 205, 223, 230, 252, 267, 390  
 SEUp commands, 576  
 setup configuration, 122, 126, 390  
 setup defaults, 123  
 setup memory, 122  
 setup reported, 405  
 setup, recall, 342  
 setup, save, 353  
 short form, 3, 678  
 show channel labels, 235  
 show measurements, 293, 314

SICL example in C, 700  
 SICL example in Visual Basic, 709  
 SICL examples, 700  
 SIGNal commands, 576  
 signal type, 217, 246  
 simple command headers, 679  
 single acquisition, 167  
 single-ended signal type, 217, 246  
 single-shot DUT, synchronizing with, 670  
 size, 228, 335  
 SIZE commands, 576  
 skew, 216, 595  
 slope, 441, 490  
 slope (direction) of waveform, 620  
 SLOPe commands, 576  
 slope not valid in TV trigger mode, 441  
 slope of edge, 483  
 slope parameter for delay measurement, 300  
 slope, Nth edge in burst, 436  
 software version, 118  
 source, 238, 252, 315, 424, 477, 545  
 SOURce commands, 576  
 source for function, 263, 264, 603  
 source for trigger, 442  
 source for TV trigger, 502  
 SOURce1 commands, 577  
 SOURce2 commands, 577  
 span, 252  
 span of frequency on display, 265  
 specify measurement, 315  
 SPI, 490, 491, 493  
 SPI commands, 577  
 SPI decode word width, 377  
 SPI trigger, 492, 494  
 SPI trigger clock, 495  
 SPI trigger commands, 489  
 SPI trigger data, 496  
 SPI trigger frame, 497  
 SPI triggering, 405  
 square root math function, 259  
 SRE (Service Request Enable Register), 128, 650  
 SRQ (Service Request interrupt), 147, 154  
 STANDard commands, 577  
 standard deviation measured on waveform, 313  
 Standard Event Status Enable Register (ESE), 114, 655  
 Standard Event Status Register (ESR), 116, 654  
 standard for video, 503  
 standard, LIN, 478  
 start acquisition, 131, 145, 165, 167  
 start and stop edges, 298  
 START commands, 577  
 start cursor, 618  
 start measurement, 293  
 start print job, 276  
 start time, 462, 618  
 start time marker, 615  
 state memory, 126  
 state of instrument, 119, 390  
 status, 129, 168, 170

Status Byte Register (STB), 127, 129, 130, 648  
 STATus commands, 577  
 status data structure clear, 113  
 status registers, 58  
 status reporting, 643  
 STB (Status Byte Register), 127, 129, 130, 648  
 step size for frequency span, 265  
 stop, 145, 169  
 stop acquisition, 169  
 stop cursor, 619  
 stop displaying channel, 143  
 stop displaying math function, 143  
 stop displaying pod, 143  
 stop time, 462, 619  
 storage, 126  
 store instrument setup, 119, 126  
 store setup, 126  
 store waveforms to pixel memory, 153  
 storing calibration information, 198  
 string variables, 56  
 string variables, reading multiple query results into, 57  
 string variables, reading query results into multiple, 57  
 string, quoted ASCII, 105  
 subnet mask, 39  
 subsource, waveform source, 549  
 subsystem commands, 108, 693  
 subtract math function, 252, 259, 545  
 subtract math function as g(t) source, 255  
 sweep mode, trigger, 405, 415  
 sweep speed set to fast to measure fall time, 303  
 sweep speed set to fast to measure rise time, 312  
 switch disable, 388  
 switch, calibration protect, 201  
 sync break, LIN, 479  
 sync frame count (FlexRay), 372  
 syntax elements, 104  
 syntax rules, program data, 680  
 syntax, optional terms, 104  
 syntax, program message, 677  
 SYSTem commands, 384  
 system commands, 385, 386, 387, 388, 390, 392  
 system commands introduction, 384

## T

tdelta, 614  
 tedge, 317  
 telnet ports 5024 and 5025, 533  
 Telnet sockets, 59  
 temporary message, 386  
 TER (Trigger Event Register), 170, 651  
 test, self, 132  
 text, writing to display, 386  
 threshold, 229, 336, 594, 632  
 THReshold commands, 578  
 threshold voltage (lower) for measurement, 612  
 threshold voltage (upper) for measurement, 622  
 thresholds, 298, 615  
 thresholds used to measure period, 308  
 thresholds, how autoscale affects, 139  
 TIFF image format, 349  
 time base, 395, 396, 397, 399, 400, 628  
 time base commands introduction, 394  
 time base reset conditions, 123  
 time base window, 402, 403, 404  
 time between points, 614  
 time buckets, 175, 176  
 TIME commands, 578  
 time delay, 628  
 time delta, 614  
 time difference between data points, 553  
 time duration, 428, 429, 432, 462  
 time holdoff for trigger, 410  
 time interval, 317, 319, 614  
 time interval between trigger and occurrence, 620  
 time marker sets start time, 615  
 time per division, 397  
 time record, 266  
 time slot, FLEXray time trigger, 453  
 time specified, 329  
 time, calibration, 203  
 time, start marker, 618  
 time, stop marker, 619  
 time, system, 392  
 time/div, how autoscale affects, 139  
 time-at-max measurement, 616  
 time-at-min measurement, 617  
 TIMEbase commands, 393  
 timebase vernier, 401  
 TIMEbase:MODE, 54  
 time-ordered label list, 236  
 timeout, 491  
 timer, 487  
 timing measurement, 293  
 title channels, 212  
 top of waveform value measured, 330  
 TOTAl commands, 579  
 total frame count (CAN), 367  
 total frame count (FlexRay), 373  
 trace memories, how autoscale affects, 139  
 trace memory, 168, 171  
 trace memory data, 233  
 track measurements, 314  
 trademarks, 2  
 transfer instrument state, 119, 390  
 transmit, 233  
 tree traversal rules, 693  
 tree, command, 681  
 TRG (Trigger), 128, 130, 131  
 trigger (external) input impedance, 243, 601  
 trigger armed event register, 156, 158  
 trigger burst, UART, 508  
 TRIGger CAN commands, 416  
 trigger channel source, 463, 502  
 TRIGger commands, 405, 579  
 TRIGger commands, general, 408



## Index

trigger data, UART, 509  
trigger duration, 428, 429  
TRIGger DURation commands, 427  
TRIGger EBUrst commands, 433  
trigger edge, 483  
TRIGger EDGE commands, 437  
trigger edge coupling, 438  
trigger edge slope, 441  
trigger event bit, 170  
Trigger Event Register (TER), 651  
TRIGger FLEXray commands, 443  
TRIGger GLITCh commands, 455  
trigger holdoff, 410  
trigger idle, UART, 510  
TRIGger IIC commands, 464  
trigger level constants, 214  
trigger level voltage, 439  
TRIGger LIN commands, 473  
trigger occurred, 130  
trigger pattern, 413, 430  
trigger qualifier, 431  
trigger qualifier, UART, 513  
trigger reset conditions, 123  
TRIGger SEquence commands, 481  
trigger SPI clock slope, 490  
TRIGger SPI commands, 489  
trigger status bit, 170  
trigger sweep mode, 405  
TRIGger TV commands, 498  
trigger type, UART, 516  
TRIGger UART commands, 504  
TRIGger USB commands, 518  
trigger, CAN, 425  
trigger, CAN acknowledge, 629  
trigger, CAN pattern data, 418  
trigger, CAN pattern data length, 419  
trigger, CAN pattern ID, 420  
trigger, CAN pattern ID mode, 421  
trigger, CAN sample point, 422  
trigger, CAN signal baudrate, 423  
trigger, CAN signal definition, 630  
trigger, CAN source, 424  
trigger, duration greater than, 428  
trigger, duration less than, 429  
trigger, duration pattern, 430  
trigger, duration qualifier, 431  
trigger, duration range, 432  
trigger, edge coupling, 438  
trigger, edge level, 439  
trigger, edge reject, 440  
trigger, edge slope, 441  
trigger, edge source, 442  
trigger, FLEXray, 454  
trigger, FLEXray error, 444  
trigger, glitch greater than, 457  
trigger, glitch less than, 458  
trigger, glitch level, 459  
trigger, glitch polarity, 460  
trigger, glitch qualifier, 461  
trigger, glitch range, 462  
trigger, glitch source, 463  
trigger, high frequency reject filter, 409

trigger, holdoff, 410  
trigger, IIC clock source, 468  
trigger, IIC data source, 469  
trigger, IIC pattern address, 465  
trigger, IIC pattern data, 466  
trigger, IIC pattern data 2, 467  
trigger, IIC qualifier, 470  
trigger, IIC signal baudrate, 476  
trigger, IIC type, 471  
trigger, LIN, 480  
trigger, LIN sample point, 475  
trigger, LIN signal definition, 631  
trigger, LIN source, 477  
trigger, mode, 411  
trigger, noise reject filter, 412  
trigger, Nth edge in burst slope, 436  
trigger, Nth edge of burst count, 434  
trigger, pattern, 413  
trigger, sequence, 488  
trigger, sequence count, 482  
trigger, sequence edge, 483  
trigger, sequence find, 484  
trigger, sequence pattern, 485  
trigger, sequence reset, 486  
trigger, sequence timer, 487  
trigger, SPI clock slope, 490  
trigger, SPI clock source, 495  
trigger, SPI clock timeout, 491  
trigger, SPI data source, 496  
trigger, SPI frame source, 497  
trigger, SPI framing, 492  
trigger, SPI pattern data, 493  
trigger, SPI pattern width, 494  
trigger, sweep, 415  
trigger, threshold, 632  
trigger, TV line, 499  
trigger, TV mode, 500, 633  
trigger, TV polarity, 501  
trigger, TV source, 502  
trigger, TV standard, 503  
trigger, UART baudrate, 506  
trigger, UART bit order, 507  
trigger, UART parity, 511  
trigger, UART polarity, 512  
trigger, UART Rx source, 514  
trigger, UART Tx source, 515  
trigger, UART width, 517  
trigger, USB, 522  
trigger, USB D- source, 519  
trigger, USB D+ source, 520  
trigger, USB speed, 521  
truncation rules, 678  
TST (Self Test), 132  
tstart, 618  
tstop, 619  
TTL threshold voltage for digital channels, 229, 594  
TTL trigger threshold voltage, 632  
turn function on or off, 604  
turn off channel, 143  
turn off channel labels, 235  
turn off cursors, 139

turn off delayed time base mode, 139  
turn off digital pod, 143  
turn off math function, 143  
turn off measurements, 139  
turn off trace memories, 139  
turn on channel labels, 235  
turn on channel number display, 599  
turn on channels, 139  
turning channel display on and off, 209  
turning off/on function calculation, 254  
turning vectors on or off, 598  
TV mode, 500, 633  
TV trigger commands, 498  
TV trigger line number setting, 499  
TV trigger mode, 502  
TV trigger polarity, 501  
TV trigger standard setting, 503  
TV triggering, 405  
tvmode, 633  
Tx data, UART, 549  
Tx frame count (UART), 382  
Tx source, 515  
type, 185, 550  
TYPE commands, 582

## U

UART baud rate, 506  
UART bit order, 507  
UART commands, 583  
UART frame counters, reset, 380  
UART parity, 511  
UART polarity, 512  
UART Rx source, 514  
UART trigger burst, 508  
UART trigger commands, 504  
UART trigger data, 509  
UART trigger idle, 510  
UART trigger qualifier, 513  
UART trigger type, 516  
UART Tx data, 549  
UART Tx source, 515  
UART width, 517  
UNITs commands, 583  
units per division, 220, 221, 249, 400  
units per division (vertical) for function, 220, 262  
unsigned mode, 551  
upper threshold, 308  
upper threshold voltage for measurement, 622  
uppercase characters in commands, 677  
URQ (User Request) status bit, 115, 117  
USB (Device) interface, 39  
USB source, 519, 520  
USB speed, 521  
USB trigger, 522  
USB trigger commands, 518  
USB triggering, 405  
user defined channel labels, 212  
user defined threshold, 594  
user event conditions occurred, 130  
User's Guide, 4

user-defined threshold voltage for digital channels, [229](#)  
 user-defined trigger threshold, [632](#)  
 USR (User Event bit), [128](#), [130](#)  
 Utility button, [39](#), [40](#)  
 utilization, CAN bus, [368](#)

## V

valid command strings, [677](#)  
 valid pattern time, [428](#), [429](#)  
 value, [319](#)  
 value measured at base of waveform, [323](#)  
 value measured at specified time, [329](#)  
 value measured at top of waveform, [330](#)  
 value ranges, [105](#)  
 values required to fill time buckets, [176](#)  
 VBA, [50](#), [764](#)  
 vectors, [239](#)  
 vectors turned on or off, [598](#)  
 vectors, turning on or off, [230](#)  
 vernier, channel, [222](#)  
 vernier, horizontal, [401](#)  
 vertical adjustment, fine (vernier), [222](#)  
 vertical amplitude measurement, [321](#)  
 vertical axis defined by RANGE, [260](#)  
 vertical axis range for channels, [219](#)  
 vertical offset for channels, [213](#)  
 vertical peak-to-peak measured on waveform, [326](#)  
 vertical scale, [220](#), [262](#)  
 vertical scaling, [540](#)  
 vertical threshold, [594](#)  
 vertical value at center screen, [258](#)  
 vertical value maximum measured on waveform, [324](#)  
 vertical value measurements to calculate overshoot, [306](#)  
 vertical value minimum measured on waveform, [325](#)  
 video line to trigger on, [499](#)  
 video standard selection, [503](#)  
 view, [171](#), [252](#), [552](#), [599](#)  
 view turns function on or off, [604](#)  
 VISA COM example in C#, [774](#)  
 VISA COM example in Visual Basic, [764](#)  
 VISA COM example in Visual Basic .NET, [785](#)  
 VISA example in C, [718](#)  
 VISA example in C#, [737](#)  
 VISA example in Visual Basic, [727](#)  
 VISA example in Visual Basic .NET, [751](#)  
 VISA examples, [718](#), [764](#)  
 Visual Basic .NET, VISA COM example, [785](#)  
 Visual Basic .NET, VISA example, [751](#)  
 Visual Basic 6.0, [51](#)  
 Visual Basic for Applications, [50](#), [764](#)  
 Visual Basic, SICL library example, [709](#)  
 Visual Basic, VISA COM example, [764](#)  
 Visual Basic, VISA example, [727](#)  
 voltage crossing reported or not found, [620](#)  
 voltage difference between data points, [556](#)  
 voltage difference measured, [623](#)

voltage level for active trigger, [439](#)  
 voltage marker used to measure waveform, [624](#), [625](#)  
 voltage offset value for channels, [213](#)  
 voltage probe, [221](#), [249](#)  
 voltage ranges for channels, [219](#)  
 voltage ranges for external trigger, [248](#)  
 voltage threshold, [298](#)

## W

WAI (Wait To Continue), [133](#)  
 wait, [133](#)  
 wait for operation complete, [120](#)  
 Wait Trig bit, [156](#), [158](#)  
 waveform base value measured, [323](#)  
 WAVEform command, [49](#)  
 WAVEform commands, [523](#), [584](#)  
 waveform data, [525](#)  
 waveform data format, [355](#)  
 waveform data length, [356](#)  
 waveform data, save, [354](#)  
 waveform introduction, [525](#)  
 waveform maximum vertical value measured, [324](#)  
 waveform minimum vertical value measured, [325](#)  
 waveform must cross voltage level to be an occurrence, [620](#)  
 WAVEform parameters, [54](#)  
 waveform peak-to-peak vertical value measured, [326](#)  
 waveform period, [308](#)  
 waveform persistence, [230](#)  
 waveform RMS value measured, [328](#)  
 waveform source channels, [545](#)  
 waveform source subsource, [549](#)  
 waveform standard deviation value measured, [313](#)  
 waveform vertical amplitude, [321](#)  
 waveform voltage measured at marker, [624](#), [625](#)  
 waveform, byte order, [531](#)  
 waveform, count, [532](#)  
 waveform, data, [533](#)  
 waveform, format, [535](#)  
 waveform, points, [536](#), [538](#)  
 waveform, preamble, [540](#)  
 waveform, source, [545](#)  
 waveform, type, [550](#)  
 waveform, unsigned, [551](#)  
 waveform, view, [552](#)  
 waveform, X increment, [553](#)  
 waveform, X origin, [554](#)  
 waveform, X reference, [555](#)  
 waveform, Y increment, [556](#)  
 waveform, Y origin, [557](#)  
 waveform, Y reference, [558](#)  
 WAVEform:FORMat, [54](#)  
 Web control, [59](#)  
 what's new, [19](#)  
 width, [462](#), [517](#)

WIDTH commands, [584](#)  
 window, [402](#), [403](#), [404](#)  
 window time, [397](#)  
 window time base mode, [395](#)  
 windows, [266](#)  
 windows as filters to Fast Fourier Transforms, [266](#)  
 windows for Fast Fourier Transform functions, [266](#)  
 word format, [535](#)  
 word format for data transfer, [525](#)  
 word width, SPI decode, [377](#)  
 write text to display, [386](#)  
 write trace memory, [233](#)  
 WriteEEBlock method, [51](#), [57](#)  
 WriteList method, [51](#)  
 WriteNumber method, [51](#)  
 WriteString method, [51](#)

## X

X axis markers, [278](#)  
 X delta, [284](#)  
 X1 and X2 cursor value difference, [284](#)  
 X1 cursor, [278](#), [280](#), [281](#)  
 X2 cursor, [278](#), [282](#), [283](#)  
 X-axis functions, [394](#)  
 X-increment, [553](#)  
 X-of-max measurement, [331](#)  
 X-of-min measurement, [332](#)  
 X-origin, [554](#)  
 X-reference, [555](#)  
 X-Y mode, [394](#), [395](#)

## Y

Y axis markers, [278](#)  
 Y1 and Y2 cursor value difference, [287](#)  
 Y1 cursor, [278](#), [281](#), [285](#), [287](#)  
 Y2 cursor, [278](#), [283](#), [286](#), [287](#)  
 Y-axis value, [557](#)  
 Y-increment, [556](#)  
 Y-origin, [557](#), [558](#)  
 Y-reference, [558](#)

## Z

zero values in waveform data, [533](#)

