

Agilent InfiniiVision 7000 Series Oscilloscopes

Programmer's Reference



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 05.10.0000

Edition

April 11, 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 sub-contract, Software is delivered and licensed as “Commercial computer software” as defined in DFAR 252.227-7014 (June 1995), or as a “commercial item” as defined in FAR 2.101(a) or as “Restricted computer software” as defined in FAR 52.227-19 (June 1987) or any equivalent

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

Safety Notices

CAUTION

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

WARNING

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

In This Book

This programmer's reference gives detailed information on all the commands available for controlling these oscilloscope models:

Table 1 InfiniiVision 7000 Series Oscilloscope Models

Channels	Input Bandwidth		
	1 GHz	500 MHz	300 MHz
4 analog + 16 digital (mixed-signal)	MS07104A	MS07054A	MS07034A
2 analog + 16 digital (mixed-signal)		MS07052A	MS07032A
4 analog	DS07104A	DS07054A	DS07034A
2 analog		DS07052A	DS07032A

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.

Command arguments and syntax are described for each command. Some command descriptions have example code.

- ["What's New"](#) on page 17
- ["Commands Quick Reference"](#) on page 21
- ["Commands by Subsystem"](#) on page 67
- ["Commands A-Z"](#) on page 513
- ["Obsolete and Discontinued Commands"](#) on page 539
- ["Error Messages"](#) on page 585
- ["Status Reporting"](#) on page 593
- ["More About Oscilloscope Commands"](#) on page 615
- ["Programming Examples"](#) on page 637

See the *Agilent InfiniiVision 7000 Series Oscilloscopes Programmer's Quick Start Guide* for information on installing the IO libraries, connecting the oscilloscope to the controller PC, and getting started with oscilloscope programming.

See your oscilloscope's *User's Guide* for more information on front-panel operation.

**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.

**Example
Programs**

The example programs are designed to work with multiple InfiniiVision 7000 Series oscilloscopes. Therefore, the commands may not match the example code exactly, but the example code should run because of the designed-in backward compatibility with earlier commands.

Contents

In This Book 3

1 What's New

What's New in Version 5.10 18

Version 5.00 at Introduction 19

2 Commands Quick Reference

Command Summary 22

Syntax Elements 63

Number Format 63

<NL> (Line Terminator) 63

[] (Optional Syntax Terms) 63

{ } (Braces) 63

::= (Defined As) 63

< > (Angle Brackets) 64

... (Ellipsis) 64

n,...,p (Value Ranges) 64

d (Digits) 64

Quoted ASCII String 64

Definite-Length Block Response Data 64

3 Commands by Subsystem

Common (*) Commands 69

*CLS (Clear Status) 73

*ESE (Standard Event Status Enable) 74

*ESR (Standard Event Status Register) 76

*IDN (Identification Number) 78

*LRN (Learn Device Setup) 79

*OPC (Operation Complete) 80

*OPT (Option Identification) 81

*RCL (Recall) 82

*RST (Reset) 83

*SAV (Save) 86

*SRE (Service Request Enable) 87

*STB (Read Status Byte) 89

*TRG (Trigger)	91
*TST (Self Test)	92
*WAI (Wait To Continue)	93
Root (:) Commands	94
:ACTivity	97
:AER (Arm Event Register)	98
:AUToscale	99
:AUToscale:AMODE	101
:AUToscale:CHANnels	102
:BLANK	103
:CDISplay	104
:DIGitize	105
:HWEenable (Hardware Event Enable Register)	107
:HWERegister:CONDition (Hardware Event Condition Register)	109
:HWERegister[:EVENT] (Hardware Event Event Register)	111
:MERGE	113
:OPEE (Operation Status Enable Register)	114
:OPERRegister:CONDition (Operation Status Condition Register)	116
:OPERRegister[:EVENT] (Operation Status Event Register)	118
:OVLenable (Overload Event Enable Register)	120
:OVLRegister (Overload Event Register)	122
:PRINt	124
:RUN	125
:SERial	126
:SINGle	127
:STATus	128
:STOP	129
:TER (Trigger Event Register)	130
:VIEW	131
:ACQuire Commands	132
:ACQuire:AALias	134
:ACQuire:COMPLete	135
:ACQuire:COUNt	136
:ACQuire:DAALias	137
:ACQuire:MODE	138
:ACQuire:POINts	139
:ACQuire:RSIGnal	140
:ACQuire:SEGMENTed:COUNt	141
:ACQuire:SEGMENTed:INDex	142
:ACQuire:SRATE	144
:ACQuire:TYPE	145

:BUS<n> Commands	147
:BUS<n>:BIT<m>	149
:BUS<n>:BITS	150
:BUS<n>:CLEar	152
:BUS<n>:DISPlay	153
:BUS<n>:LABel	154
:BUS<n>:MASK	155
:CALibrate Commands	156
:CALibrate:DATE	157
:CALibrate:LABel	158
:CALibrate:STARt	159
:CALibrate:STATus	160
:CALibrate:SWITCh	161
:CALibrate:TEMPerature	162
:CALibrate:TIME	163
:CHANnel<n> Commands	164
:CHANnel<n>:BWLimit	167
:CHANnel<n>:COUPling	168
:CHANnel<n>:DISPlay	169
:CHANnel<n>:IMPedance	170
:CHANnel<n>:INVert	171
:CHANnel<n>:LABel	172
:CHANnel<n>:OFFSet	173
:CHANnel<n>:PROBe	174
:CHANnel<n>:PROBe:ID	175
:CHANnel<n>:PROBe:SKEW	176
:CHANnel<n>:PROBe:STYPe	177
:CHANnel<n>:PROTection	178
:CHANnel<n>:RANGe	179
:CHANnel<n>:SCALE	180
:CHANnel<n>:UNITs	181
:CHANnel<n>:VERNier	182
:DIGital<n> Commands	183
:DIGital<n>:DISPlay	185
:DIGital<n>:LABel	186
:DIGital<n>:POSition	187
:DIGital<n>:SIZE	188
:DIGital<n>:THReshold	189
:DISPlay Commands	190
:DISPlay:CLEar	192

:DISPlay:DATA	193
:DISPlay:LABel	195
:DISPlay:LABList	196
:DISPlay:PERsistence	197
:DISPlay:SOURce	198
:DISPlay:VECTors	199
:EXTErnal Trigger Commands	200
:EXTErnal:BWLimit	202
:EXTErnal:IMPedance	203
:EXTErnal:PROBe	204
:EXTErnal:PROBe:ID	205
:EXTErnal:PROBe:STYPe	206
:EXTErnal:PROTEction	207
:EXTErnal:RANGe	208
:EXTErnal:UNITs	209
:FUNCTion Commands	210
:FUNCTion:CENTer	212
:FUNCTion:DISPlay	213
:FUNCTion:OFFSet	214
:FUNCTion:OPERation	215
:FUNCTion:RANGe	216
:FUNCTion:REFerence	217
:FUNCTion:SCALE	218
:FUNCTion:SOURce	219
:FUNCTion:SPAN	220
:FUNCTion:WINDow	221
:HARDcopy Commands	222
:HARDcopy:AREA	224
:HARDcopy:APRinter	225
:HARDcopy:FACTors	226
:HARDcopy:FFEed	227
:HARDcopy:INKSaver	228
:HARDcopy:PALette	229
:HARDcopy:PRinter:LIST	230
:HARDcopy:STARt	231
:MARKer Commands	232
:MARKer:MODE	234
:MARKer:X1Position	235
:MARKer:X1Y1source	236
:MARKer:X2Position	237

:MARKer:X2Y2source	238
:MARKer:XDELta	239
:MARKer:Y1Position	240
:MARKer:Y2Position	241
:MARKer:YDELta	242
:MEASure Commands	243
:MEASure:CLEar	250
:MEASure:COUNter	251
:MEASure:DEFine	252
:MEASure:DELay	255
:MEASure:DUTYcycle	257
:MEASure:FALLtime	258
:MEASure:FREQuency	259
:MEASure:NWIDth	260
:MEASure:OVERshoot	261
:MEASure:PERiod	263
:MEASure:PHASe	264
:MEASure:PREShoot	265
:MEASure:PWIDth	266
:MEASure:RISetime	267
:MEASure:SDEVIation	268
:MEASure:SHOW	269
:MEASure:SOURce	270
:MEASure:TEDGe	272
:MEASure:TVALue	274
:MEASure:VAMPLitude	276
:MEASure:VAverage	277
:MEASure:VBASe	278
:MEASure:VMAX	279
:MEASure:VMIN	280
:MEASure:VPP	281
:MEASure:VRMS	282
:MEASure:VTIMe	283
:MEASure:VTOP	284
:MEASure:XMAX	285
:MEASure:XMIN	286
:POD Commands	287
:POD<n>:DISPlay	288
:POD<n>:SIZE	289
:POD<n>:THReshold	290

:RECall Commands	292
:RECall:FIleName	293
:RECall:IMAGe[:START]	294
:RECall:PWD	295
:RECall:SETup[:START]	296
:SAVE Commands	297
:SAVE:FIleName	299
:SAVE:IMAGe[:START]	300
:SAVE:IMAGe:AREA	301
:SAVE:IMAGe:FACTors	302
:SAVE:IMAGe:FORMat	303
:SAVE:IMAGe:INKSaver	304
:SAVE:IMAGe:PALette	305
:SAVE:PWD	306
:SAVE:SETup[:START]	307
:SAVE:WAVEform[:START]	308
:SAVE:WAVEform:FORMat	309
:SAVE:WAVEform:LENGth	310
:SBUS Commands	311
:SBUS:BUSDoctor:ADDResS	314
:SBUS:BUSDoctor:BAUDrate	315
:SBUS:BUSDoctor:CHANnel	316
:SBUS:BUSDoctor:MODE	317
:SBUS:CAN:COUNT:ERRor	318
:SBUS:CAN:COUNT:OVERload	319
:SBUS:CAN:COUNT:RESet	320
:SBUS:CAN:COUNT:TOTal	321
:SBUS:CAN:COUNT:UTILization	322
:SBUS:DISPlay	323
:SBUS:FLEXray:COUNT:NULL	324
:SBUS:FLEXray:COUNT:RESet	325
:SBUS:FLEXray:COUNT:SYNC	326
:SBUS:FLEXray:COUNT:TOTal	327
:SBUS:IIC:ASIZe	328
:SBUS:LIN:PARity	329
:SBUS:MODE	330
:SBUS:SPI:WIDTh	331
:SBUS:UART:BASE	332
:SBUS:UART:COUNT:ERRor	333
:SBUS:UART:COUNT:RESet	334
:SBUS:UART:COUNT:RXFRames	335

:SBUS:UART:COUNt:TXFRames	336
:SBUS:UART:FRAMing	337
:SYSTem Commands	338
:SYSTem:DATE	339
:SYSTem:DSP	340
:SYSTem:ERRor	341
:SYSTem:LOCK	342
:SYSTem:SETup	343
:SYSTem:TIME	345
:TIMebase Commands	346
:TIMebase:MODE	348
:TIMebase:POSition	349
:TIMebase:RANGe	350
:TIMebase:REFClock	351
:TIMebase:REFerence	352
:TIMebase:SCALe	353
:TIMebase:VERNier	354
:TIMebase:WINDow:POSition	355
:TIMebase:WINDow:RANGe	356
:TIMebase:WINDow:SCALe	357
:TRIGger Commands	358
General :TRIGger Commands	361
:TRIGger:HFReject	362
:TRIGger:HOLDoff	363
:TRIGger:MODE	364
:TRIGger:NREJect	365
:TRIGger:PATTern	366
:TRIGger:SWEep	368
:TRIGger:CAN Commands	369
:TRIGger:CAN:PATTern:DATA	371
:TRIGger:CAN:PATTern:DATA:LENGth	372
:TRIGger:CAN:PATTern:ID	373
:TRIGger:CAN:PATTern:ID:MODE	374
:TRIGger:CAN:SAMPlepoint	375
:TRIGger:CAN:SIGNal:BAUDrate	376
:TRIGger:CAN:SOURce	377
:TRIGger:CAN:TRIGger	378
:TRIGger:DURation Commands	380
:TRIGger:DURation:GREaterthan	381
:TRIGger:DURation:LESSthan	382

:TRIGger:DURation:PATtern	383
:TRIGger:DURation:QUALifier	384
:TRIGger:DURation:RANGe	385
:TRIGger:EBURst Commands	386
:TRIGger:EBURst:COUNt	387
:TRIGger:EBURst:IDLE	388
:TRIGger:EBURst:SLOPe	389
:TRIGger[:EDGE] Commands	390
:TRIGger[:EDGE]:COUPling	391
:TRIGger[:EDGE]:LEVel	392
:TRIGger[:EDGE]:REJect	393
:TRIGger[:EDGE]:SLOPe	394
:TRIGger[:EDGE]:SOURce	395
:TRIGger:FLEXray Commands	396
:TRIGger:FLEXray:ERRor:TYPE	397
:TRIGger:FLEXray:FRAMe:CCBase	399
:TRIGger:FLEXray:FRAMe:CCRepetition	400
:TRIGger:FLEXray:FRAMe:ID	401
:TRIGger:FLEXray:FRAMe:TYPE	402
:TRIGger:FLEXray:TIME:CBASe	403
:TRIGger:FLEXray:TIME:CREPetition	404
:TRIGger:FLEXray:TIME:SEGMENT	405
:TRIGger:FLEXray:TIME:SLOT	406
:TRIGger:FLEXray:TRIGger	407
:TRIGger:GLITch Commands	408
:TRIGger:GLITch:GREaterthan	410
:TRIGger:GLITch:LESSthan	411
:TRIGger:GLITch:LEVel	412
:TRIGger:GLITch:POLarity	413
:TRIGger:GLITch:QUALifier	414
:TRIGger:GLITch:RANGe	415
:TRIGger:GLITch:SOURce	416
:TRIGger:IIC Commands	417
:TRIGger:IIC:PATtern:ADDRes	418
:TRIGger:IIC:PATtern:DATA	419
:TRIGger:IIC:PATtern:DATA2	420
:TRIGger:IIC:SOURce:CLOCK	421
:TRIGger:IIC:SOURce:DATA	422
:TRIGger:IIC:TRIGger:QUALifier	423
:TRIGger:IIC:TRIGger[:TYPE]	424
:TRIGger:LIN Commands	426
:TRIGger:LIN:ID	427

:TRIGger:LIN:SAMPlEpoint	428
:TRIGger:LIN:SIGNal:BAUDrate	429
:TRIGger:LIN:SOURce	430
:TRIGger:LIN:STANdard	431
:TRIGger:LIN:SYNCbreak	432
:TRIGger:LIN:TRIGger	433
:TRIGger:SEQuence Commands	434
:TRIGger:SEQuence:COUNT	435
:TRIGger:SEQuence:EDGE	436
:TRIGger:SEQuence:FIND	437
:TRIGger:SEQuence:PATtern	438
:TRIGger:SEQuence:RESet	439
:TRIGger:SEQuence:TIMer	440
:TRIGger:SEQuence:TRIGger	441
:TRIGger:SPI Commands	442
:TRIGger:SPI:CLOCK:SLOPe	443
:TRIGger:SPI:CLOCK:TIMeout	444
:TRIGger:SPI:FRAMing	445
:TRIGger:SPI:PATtern:DATA	446
:TRIGger:SPI:PATtern:WIDTh	447
:TRIGger:SPI:SOURce:CLOCK	448
:TRIGger:SPI:SOURce:DATA	449
:TRIGger:SPI:SOURce:FRAMe	450
:TRIGger:TV Commands	451
:TRIGger:TV:LINE	452
:TRIGger:TV:MODE	453
:TRIGger:TV:POLarity	454
:TRIGger:TV:SOURce	455
:TRIGger:TV:STANdard	456
:TRIGger:UART Commands	457
:TRIGger:UART:BAUDrate	459
:TRIGger:UART:BITorder	460
:TRIGger:UART:BURSt	461
:TRIGger:UART:DATA	462
:TRIGger:UART:IDLE	463
:TRIGger:UART:PARity	464
:TRIGger:UART:POLarity	465
:TRIGger:UART:QUALifier	466
:TRIGger:UART:SOURce:RX	467
:TRIGger:UART:SOURce:TX	468
:TRIGger:UART:TYPE	469
:TRIGger:UART:WIDTh	470

:TRIGger:USB Commands	471
:TRIGger:USB:SOURce:DMINus	472
:TRIGger:USB:SOURce:DPLus	473
:TRIGger:USB:SPEed	474
:TRIGger:USB:TRIGger	475
:WAVeform Commands	476
:WAVeform:BYTeorder	484
:WAVeform:COUNt	485
:WAVeform:DATA	486
:WAVeform:FORMat	488
:WAVeform:POINts	489
:WAVeform:POINts:MODE	491
:WAVeform:PREamble	493
:WAVeform:SEGMENTed:COUNt	496
:WAVeform:SEGMENTed:TTAG	497
:WAVeform:SOURce	498
:WAVeform:SOURce:SUBSource	502
:WAVeform:TYPE	503
:WAVeform:UNSigned	504
:WAVeform:VIEW	505
:WAVeform:XINCrement	506
:WAVeform:XORigin	507
:WAVeform:XREFerence	508
:WAVeform:YINCrement	509
:WAVeform:YORigin	510
:WAVeform:YREFerence	511

4 Commands A-Z

5 Obsolete and Discontinued Commands

:CHANnel:ACTivity	544
:CHANnel:LABel	545
:CHANnel:THReshold	546
:CHANnel2:SKEW	547
:CHANnel<n>:INPut	548
:CHANnel<n>:PMODE	549
:DISPlay:CONNect	550
:DISPlay:ORDer	551
:ERASe	552
:EXTernal:INPut	553
:EXTernal:PMODE	554

:FUNction:VIEW	555
:HARDcopy:DESTination	556
:HARDcopy:DEvice	557
:HARDcopy:FILEname	558
:HARDcopy:FORMat	559
:HARDcopy:GRAYscale	560
:HARDcopy:IGColors	561
:HARDcopy:PDriver	562
:MEASure:LOWer	563
:MEASure:SCRatch	564
:MEASure:TDELta	565
:MEASure:THResholds	566
:MEASure:TMAX	567
:MEASure:TMIN	568
:MEASure:TSTART	569
:MEASure:TSTOP	570
:MEASure:TVOLt	571
:MEASure:UPPer	573
:MEASure:VDELta	574
:MEASure:VSTART	575
:MEASure:VSTOP	576
:PRINt?	577
:TIMEbase:DELay	579
:TRIGger:CAN:ACKNowledge	580
:TRIGger:CAN:SIGNal:DEFinition	581
:TRIGger:LIN:SIGNal:DEFinition	582
:TRIGger:THReshold	583
:TRIGger:TV:TVMode	584

6 Error Messages

7 Status Reporting

Status Reporting Data Structures	595
Status Byte Register (STB)	598
Service Request Enable Register (SRE)	600
Trigger Event Register (TER)	601
Output Queue	602
Message Queue	603
(Standard) Event Status Register (ESR)	604

(Standard) Event Status Enable Register (ESE)	605
Error Queue	606
Operation Status Event Register (:OPERegister[:EVENT])	607
Operation Status Condition Register (:OPERegister:CONDition)	608
Arm Event Register (AER)	609
Hardware Event Event Register (:HWERegister[:EVENT])	610
Hardware Event Condition Register (:HWERegister:CONDition)	611
Clearing Registers and Queues	612
Status Reporting Decision Chart	613

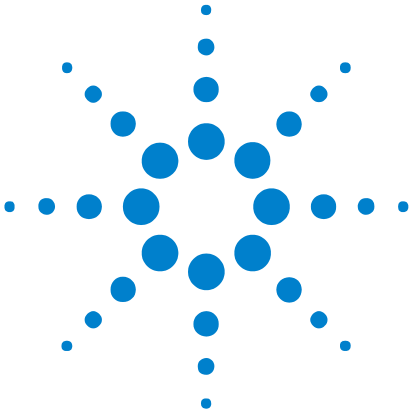
8 More About Oscilloscope Commands

Command Classifications	616
Core Commands	616
Non-Core Commands	616
Obsolete Commands	616
Valid Command/Query Strings	617
Program Message Syntax	617
Command Tree	621
Duplicate Mnemonics	633
Tree Traversal Rules and Multiple Commands	633
Query Return Values	635
All Oscilloscope Commands Are Sequential	636

9 Programming Examples

SICL Examples	638
SICL Example in C	638
VISA Examples	647
VISA Example in C	647
VISA Example in Visual Basic 6	655
VISA Example in C#	665
VISA Example in Visual Basic .NET	678
VISA COM Examples	692
VISA COM Example in Visual Basic 6	692
VISA COM Example in C#	701
VISA COM Example in Visual Basic .NET	713

Index



1 What's New

What's New in Version 5.10 18
Version 5.00 at Introduction 19

What's New in Version 5.10

New features in version 5.10 of the InfiniiVision 7000 Series oscilloscope software are:

- Segmented memory acquisition mode, enabled with Option SGM.

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

New Commands

Command	Description
:ACQUIRE:SEGMENTED:COUNT (see page 141)	Sets the number of memory segments.
:ACQUIRE:SEGMENTED:INDEX (see page 142)	Selects the segmented memory index.
:WAVEFORM:SEGMENTED:COUNT (see page 496)	Returns the number of segments in the currently acquired waveform data.
:WAVEFORM:SEGMENTED:TTAG (see page 497)	Returns the time tag for the selected segmented memory index.

Changed Commands

Command	Differences
:ACQUIRE:MODE (see page 138)	You can now select the SEGMENTED memory mode.

Version 5.00 at Introduction

The Agilent InfiniiVision 7000 Series oscilloscopes were introduced with version 5.00 of oscilloscope operating software. The command set is based on the 6000 Series oscilloscopes (and the 54620/54640 Series oscilloscopes before them).

1 What's New



2 Commands Quick Reference

Command Summary 22

Syntax Elements 63



Command Summary

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns																																				
*CLS (see page 73)	n/a	n/a																																				
*ESE <mask> (see page 74)	*ESE? (see page 75)	<p><mask> ::= 0 to 255; an integer in NR1 format:</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Weight</th> <th>Name</th> <th>Enables</th> </tr> </thead> <tbody> <tr> <td>7</td> <td>128</td> <td>PON</td> <td>Power On</td> </tr> <tr> <td>6</td> <td>64</td> <td>URQ</td> <td>User Request</td> </tr> <tr> <td>5</td> <td>32</td> <td>CME</td> <td>Command Error</td> </tr> <tr> <td>4</td> <td>16</td> <td>EXE</td> <td>Execution Error</td> </tr> <tr> <td>3</td> <td>8</td> <td>DDE</td> <td>Dev. Dependent Error</td> </tr> <tr> <td>2</td> <td>4</td> <td>QYE</td> <td>Query Error</td> </tr> <tr> <td>1</td> <td>2</td> <td>RQL</td> <td>Request Control</td> </tr> <tr> <td>0</td> <td>1</td> <td>OPC</td> <td>Operation Complete</td> </tr> </tbody> </table>	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
Bit	Weight	Name	Enables																																			
7	128	PON	Power On																																			
6	64	URQ	User Request																																			
5	32	CME	Command Error																																			
4	16	EXE	Execution Error																																			
3	8	DDE	Dev. Dependent Error																																			
2	4	QYE	Query Error																																			
1	2	RQL	Request Control																																			
0	1	OPC	Operation Complete																																			
n/a	*ESR? (see page 76)	<status> ::= 0 to 255; an integer in NR1 format																																				
n/a	*IDN? (see page 76)	<p>AGILENT TECHNOLOGIES,<model>, <serial number>,X.XX.XX</p> <p><model> ::= the model number of the instrument</p> <p><serial number> ::= the serial number of the instrument</p> <p><X.XX.XX> ::= the software revision of the instrument</p>																																				
n/a	*LRN? (see page 79)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format																																				
*OPC (see page 80)	*OPC? (see page 80)	ASCII "1" is placed in the output queue when all pending device operations have completed.																																				

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 81)	<pre><return_value> ::= 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}</pre>
*RCL <value> (see page 82)	n/a	<pre><value> ::= {0 1 2 3 4 5 6 7 8 9}</pre>
*RST (see page 83)	n/a	See *RST (Reset) (see page 83)
*SAV <value> (see page 86)	n/a	<pre><value> ::= {0 1 2 3 4 5 6 7 8 9}</pre>
*SRE <mask> (see page 87)	*SRE? (see page 88)	<pre><mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables ----- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger</pre>

2 Commands Quick Reference

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see page 89)	<p><value> ::= 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 page 91)	n/a	n/a
n/a	*TST? (see page 92)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 93)	n/a	n/a

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 97)	:ACTivity? (see page 97)	<p><return value> ::= <edges>,<levels></p> <p><edges> ::= presence of edges (32-bit integer in NR1 format)</p> <p><levels> ::= logical highs or lows (32-bit integer in NR1 format)</p>
n/a	:AER? (see page 98)	{0 1}; an integer in NR1 format
:AUToscale [<source>[, ..., <source>]] (see page 99)	n/a	<p><source> ::= CHANnel<n> for DSO models</p> <p><source> ::= {CHANnel<n> DIGital0, ..., DIGital15 POD1 POD2} for MSO models</p> <p><source> can be repeated up to 5 times</p> <p><n> ::= 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 page 101)	:AUToscale:AMODE? (see page 101)	<value> ::= {NORMAL CURRENT}}
:AUToscale:CHANnels <value> (see page 102)	:AUToscale:CHANnels? (see page 102)	<value> ::= {ALL DISPLAYed}}
:BLANK [<source>] (see page 103)	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 page 104)	n/a	n/a
:DIGitize [<source>[,...,<source>]] (see page 105)	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 page 107)	:HWEenable? (see page 107)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister:CONDition? (see page 109)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENT]? (see page 111)	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see page 113)	n/a	<pixel memory> ::= {PMEMory{0 1 2 3 4 5 6 7 8 9}}
:OPEE <n> (see page 114)	:OPEE? (see page 115)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister:CONDition? (see page 116)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister[:EVENT]? (see page 118)	<n> ::= 16-bit integer in NR1 format

2 Commands Quick Reference

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:OVLenable <mask> (see page 120)	:OVLenable? (see page 121)	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input ----- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see page 122)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>] (see page 124)	n/a	<options> ::= [<print option>][,...,<print option>] <print option> ::= {COLor GRAYscale PRINter0 BMP8bit BMP PNG NOFactoRs FACToRs} <print option> can be repeated up to 5 times.
:RUN (see page 125)	n/a	n/a
n/a	:SERial (see page 126)	<return value> ::= unquoted string containing serial number
:SINGle (see page 127)	n/a	n/a
n/a	:STATus? <display> (see page 128)	{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 page 129)	n/a	n/a

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:TER? (see page 130)	{0 1}
:VIEW <source> (see page 131)	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 page 134)	{1 0}
:ACQUIRE:COMPLETE <complete> (see page 135)	:ACQUIRE:COMPLETE? (see page 135)	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNT <count> (see page 136)	:ACQUIRE:COUNT? (see page 136)	<count> ::= an integer from 1 to 65536 in NR1 format
:ACQUIRE:DAALIAS <mode> (see page 137)	:ACQUIRE:DAALIAS? (see page 137)	<mode> ::= {DISable AUTO}
:ACQUIRE:MODE <mode> (see page 138)	:ACQUIRE:MODE? (see page 138)	<mode> ::= {RTIME ETIME SEGmented}
n/a	:ACQUIRE:POINTS? (see page 139)	<# points> ::= an integer in NR1 format
:ACQUIRE:RSIGNAL <ref_signal_mode> (see page 140)	:ACQUIRE:RSIGNAL? (see page 140)	<ref_signal_mode> ::= {OFF OUT IN}
:ACQUIRE:SEGmented:COUNT <count> (see page 141)	:ACQUIRE:SEGmented:COUNT? (see page 141)	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
:ACQUIRE:SEGmented:INDEX <index> (see page 142)	:ACQUIRE:SEGmented:INDEX? (see page 142)	<index> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)

2 Commands Quick Reference

Table 4 :ACQUIRE Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:ACQUIRE:SRATE? (see page 144)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQUIRE:TYPE <type> (see page 145)	:ACQUIRE:TYPE? (see page 145)	<type> ::= {NORMAL AVERAGE HRESOLUTION PEAK}

Table 5 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 OFF} {1 ON}} (see page 149)	:BUS<n>:BIT<m>? (see page 149)	{0 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 OFF} {1 ON}} (see page 150)	:BUS<n>:BITS? (see page 150)	<channel_list>, {0 1} <channel_list> ::= (@<m>, <m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:CLEAR (see page 152)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPLAY {{0 OFF} {1 ON}} (see page 153)	:BUS<n>:DISPLAY? (see page 153)	{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 page 154)	:BUS<n>:LABel? (see page 154)	<string> ::= quoted ASCII string up to 16 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 155)	:BUS<n>:MASK? (see page 155)	<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn...n" where n ::= {0,...,9 A,...,F} for hexadecimal <n> ::= 1 or 2; an integer in NR1 format

Table 6 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 157)	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string> (see page 158)	:CALibrate:LABel? (see page 158)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:STARt (see page 159)	n/a	n/a
n/a	:CALibrate:STATus? (see page 160)	<return value> ::= ALL,<status_code>,<status_string > <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITCh? (see page 161)	{PROtected UNPRotected}

2 Commands Quick Reference

Table 6 :CALibrate Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:CALibrate:TEMPerature? (see page 162)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 163)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 7 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {0 OFF} {1 ON} (see page 167)	:CHANnel<n>:BWLimit? (see page 167)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUpling <coupling> (see page 168)	:CHANnel<n>:COUpling? (see page 168)	<coupling> ::= {AC DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {0 OFF} {1 ON} (see page 169)	:CHANnel<n>:DISPlay? (see page 169)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 170)	:CHANnel<n>:IMPedance? (see page 170)	<impedance> ::= {ONEMeg FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {0 OFF} {1 ON} (see page 171)	:CHANnel<n>:INVert? (see page 171)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see page 172)	:CHANnel<n>:LABel? (see page 172)	<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 page 173)	:CHANnel<n>:OFFSet? (see page 173)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 174)	:CHANnel<n>:PROBe? (see page 174)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 175)	<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 page 176)	:CHANnel<n>:PROBE:SKEW? (see page 176)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 177)	:CHANnel<n>:PROBE:STYPe? (see page 177)	<signal type> ::= {DIFFerential SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see page 178)	:CHANnel<n>:PROTECTioN? (see page 178)	{NORM TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 179)	:CHANnel<n>:RANGE? (see page 179)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 180)	:CHANnel<n>:SCALE? (see page 180)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see page 181)	:CHANnel<n>:UNITs? (see page 181)	<units> ::= {VOLT AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0 OFF} {1 ON}} (see page 182)	:CHANnel<n>:VERNier? (see page 182)	{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 page 185)	:DIGital<n>:DISPlay? (see page 185)	{0 1} <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:LABel <string> (see page 186)	:DIGital<n>:LABel? (see page 186)	<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 page 187)	:DIGital<n>:POSition? (see page 187)	<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

2 Commands Quick Reference

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

Command	Query	Options and Query Returns
:DIGital<n>:SIZE <value> (see page 188)	:DIGital<n>:SIZE? (see page 188)	<value> ::= {SMALl MEDium LARGE}
:DIGital<n>:THReshold <value>[suffix] (see page 189)	:DIGital<n>:THReshold ? (see page 189)	<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 192)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>] [,][<palette>]<displa y data> (see page 193)	:DISPlay:DATA? [<format>][,][<area>] [,][<palette>] (see page 193)	<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 195)	:DISPlay:LABel? (see page 195)	{0 1}
:DISPlay:LABList <binary block> (see page 196)	:DISPlay:LABList? (see page 196)	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see page 197)	:DISPlay:PERsistence? (see page 197)	<value> ::= {MINimum INFinite}}

Table 9 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:SOURce <value> (see page 198)	:DISPlay:SOURce? (see page 198)	<value> ::= {PMEMory{0 1 2 3 4 5 6 7 8 9}}
:DISPlay:VECTors {{1 ON} {0 OFF}} (see page 199)	:DISPlay:VECTors? (see page 199)	{1 0}

Table 10 :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see page 202)	:EXternal:BWLimit? (see page 202)	<bwlimit> ::= {0 OFF}
:EXternal:IMPedance <value> (see page 203)	:EXternal:IMPedance? (see page 203)	<impedance> ::= {ONEMeg FIFTy}
:EXternal:PROBe <attenuation> (see page 204)	:EXternal:PROBe? (see page 204)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXternal:PROBe:ID? (see page 205)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXternal:PROBe:STYPe <signal type> (see page 206)	:EXternal:PROBe:STYPe ? (see page 206)	<signal type> ::= {DIFFerential SINGle}
:EXternal:PROTEction[:CLEar] (see page 207)	:EXternal:PROTEction? (see page 207)	{NORM TRIP}
:EXternal:RANGe <range>[<suffix>] (see page 208)	:EXternal:RANGe? (see page 208)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXternal:UNITs <units> (see page 209)	:EXternal:UNITs? (see page 209)	<units> ::= {VOLT AMPere}

2 Commands Quick Reference

Table 11 :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENTer <frequency> (see page 212)	:FUNCTION:CENTer? (see page 212)	<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 213)	:FUNCTION:DISPlay? (see page 213)	{0 1}
:FUNCTION:OFFSet <offset> (see page 214)	:FUNCTION:OFFSet? (see page 214)	<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 215)	:FUNCTION:OPERation? (see page 215)	<operation> ::= {SUBTract MULTiPLY INTegrate DIFFerentiate FFT SQRT}
:FUNCTION:RANGE <range> (see page 216)	:FUNCTION:RANGE? (see page 216)	<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 217)	:FUNCTION:REFerence? (see page 217)	<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 218)	:FUNCTION:SCALE? (see page 218)	<scale value> ::= integer in NR1 format <suffix> ::= {V dB}
:FUNCTION:SOURce <source> (see page 219)	:FUNCTION:SOURce? (see page 219)	<source> ::= {CHANnel<n> ADD SUBT MULT} <n> ::= 1-2 or 1-4 in NR1 format

Table 11 :FUNction Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNction:SPAN (see page 220)	:FUNction:SPAN? (see page 220)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNction:WINDow <window> (see page 221)	:FUNction:WINDow? (see page 221)	<window> ::= {RECTangular HANNing FLATtop}

Table 12 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 224)	:HARDcopy:AREA? (see page 224)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 225)	:HARDcopy:APRinter? (see page 225)	<active_printer> ::= {<index> <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 OFF} {1 ON}} (see page 226)	:HARDcopy:FACTors? (see page 226)	{0 1}
:HARDcopy:FFEed {{0 OFF} {1 ON}} (see page 227)	:HARDcopy:FFEed? (see page 227)	{0 1}
:HARDcopy:INKSaver {{0 OFF} {1 ON}} (see page 228)	:HARDcopy:INKSaver? (see page 228)	{0 1}
:HARDcopy:PALETTE <palette> (see page 229)	:HARDcopy:PALETTE? (see page 229)	<palette> ::= {COLor GRAYscale NONE}
n/a	:HARDcopy:PRinter:LIST? (see page 230)	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y N} <name> ::= name of printer
:HARDcopy:START (see page 231)	n/a	n/a

Table 13 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 234)	:MARKer:MODE? (see page 234)	<mode> ::= {OFF MEASurement MANual}
:MARKer:X1Position <position>[suffix] (see page 235)	:MARKer:X1Position? (see page 235)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 236)	:MARKer:X1Y1source? (see page 236)	<source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 237)	:MARKer:X2Position? (see page 237)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 238)	:MARKer:X2Y2source? (see page 238)	<source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 239)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see page 240)	:MARKer:Y1Position? (see page 240)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 241)	:MARKer:Y2Position? (see page 241)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 242)	<return_value> ::= Y cursors delta value in NR3 format

Table 14 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 250)	n/a	n/a
:MEASure:COUNter [<source>] (see page 251)	:MEASure:COUNter? [<source>] (see page 251)	<source> ::= {CHANnel<n>} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15 } 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 page 252)	:MEASure:DEFine? DELay (see page 253)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 252)	:MEASure:DEFine? THResholds (see page 253)	<threshold spec> ::= {STANdard} {<threshold mode>,<upper>,<middle>,<lower>} <threshold mode> ::= {PERCent ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 255)	:MEASure:DELay? [<source1>] [,<source2>] (see page 255)	<source1,2> ::= {CHANnel<n> FUNction MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 257)	:MEASure:DUTYcycle? [<source>] (see page 257)	<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 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 258)	:MEASure:FALLtime? [<source>] (see page 258)	<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 259)	:MEASure:FREQuency? [<source>] (see page 259)	<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 260)	:MEASure:NWIDth? [<source>] (see page 260)	<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 261)	:MEASure:OVERshoot? [<source>] (see page 261)	<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 263)	:MEASure:PERiod? [<source>] (see page 263)	<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 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PHASe [<source1>] [,<source2>] (see page 264)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 264)	<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 265)	:MEASure:PREShoOt? [<source>] (see page 265)	<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 266)	:MEASure:PWIDth? [<source>] (see page 266)	<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 267)	:MEASure:RISEtime? [<source>] (see page 267)	<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 268)	:MEASure:SDEVIation? [<source>] (see page 268)	<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 269)	:MEASure:SHOW? (see page 269)	{1}
:MEASure:SOURce [<source1>] [,<source2>] (see page 270)	:MEASure:SOURce? (see page 270)	<source1,2> ::= {CHANnel<n> FUNctIon MATH} for DSO models <source1,2> ::= {CHANnel<n> DIGital0,...,DIGital15 FUNctIon MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= {<source> NONE}

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 272)	<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 page 274)	<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 276)	:MEASure:VAMPLitude? [<source>] (see page 276)	<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 277)	:MEASure:VAverage? [<source>] (see page 277)	<source> ::= {CHANnel<n> FUNctIon MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VBASe [<source>] (see page 278)	:MEASure:VBASe? [<source>] (see page 278)	<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 279)	:MEASure:VMAX? [<source>] (see page 279)	<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 280)	:MEASure:VMIN? [<source>] (see page 280)	<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 281)	:MEASure:VPP? [<source>] (see page 281)	<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:VRMS [<source>] (see page 282)	:MEASure:VRMS? [<source>] (see page 282)	<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 283)	<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

2 Commands Quick Reference

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 284)	:MEASure:VTOP? [<source>] (see page 284)	<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 285)	:MEASure:XMAX? [<source>] (see page 285)	<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 286)	:MEASure:XMIN? [<source>] (see page 286)	<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 page 288)	:POD<n>:DISPlay? (see page 288)	{0 1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see page 289)	:POD<n>:SIZE? (see page 289)	<value> ::= {SMALl MEDium LARGE}
:POD<n>:THReshold <type>[suffix] (see page 290)	:POD<n>:THReshold? (see page 290)	<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 page 293)	:RECall:FILEname? (see page 293)	<base_name> ::= quoted ASCII string
:RECall:IMAGe[:START] [<file_spec>] (see page 294)	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 page 295)	<path_info> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 296)	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 page 299)	:SAVE:FILEname? (see page 299)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:START] [<file_spec>] (see page 300)	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 page 301)	:SAVE:IMAGe:AREA? (see page 301)	<area> ::= {GRATicule SCReen}
:SAVE:IMAGe:FACTors {{0 OFF} {1 ON}} (see page 302)	:SAVE:IMAGe:FACTors? (see page 302)	{0 1}
:SAVE:IMAGe:FORMat <format> (see page 303)	:SAVE:IMAGe:FORMat? (see page 303)	<format> ::= {TIFF {BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGe:INKSaver {{0 OFF} {1 ON}} (see page 304)	:SAVE:IMAGe:INKSaver? (see page 304)	{0 1}

2 Commands Quick Reference

Table 17 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:IMAGe:PALette <palette> (see page 305)	:SAVE:IMAGe:PALette? (see page 305)	<palette> ::= {COLor GRAYscale MONochrome}
n/a	:SAVE:PWD? (see page 306)	<path_info> ::= quoted ASCII string
:SAVE:SETUp[:START] [<file_spec>] (see page 307)	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:START] [<file_name>] (see page 308)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 309)	:SAVE:WAVeform:FORMat ? (see page 309)	<format> ::= {ALB ASCiixy CSV BINary NONE}
:SAVE:WAVeform:LENGth <length> (see page 310)	:SAVE:WAVeform:LENGth ? (see page 310)	<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 314)	:SBUS:BUSDoctor:ADDRe ss? (see page 314)	<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 315)	:SBUS:BUSDoctor:BAUDr ate? (see page 315)	<baudrate> ::= {2500000 5000000 10000000}
:SBUS:BUSDoctor:CHANn el <channel> (see page 316)	:SBUS:BUSDoctor:CHANn el? (see page 316)	<channel> ::= {A B}
:SBUS:BUSDoctor:MODE <mode> (see page 317)	:SBUS:BUSDoctor:MODE? (see page 317)	<mode> ::= {ASYNchronous SYNchronous PC}
n/a	:SBUS:CAN:COUNT:ERRor ? (see page 318)	<frame_count> ::= integer in NR1 format

Table 18 :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:CAN:COUNT:OVERload? (see page 319)	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see page 320)	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal? (see page 321)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILization? (see page 322)	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0 OFF} {1 ON}} (see page 323)	:SBUS:DISPlay? (see page 323)	{0 1}
n/a	:SBUS:FLEXray:COUNT:NULL? (see page 324)	<frame_count> ::= integer in NR1 format
:SBUS:FLEXray:COUNT:RESet (see page 325)	n/a	n/a
n/a	:SBUS:FLEXray:COUNT:SYNC? (see page 326)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:FLEXray:COUNT:TOTal? (see page 327)	<frame_count> ::= integer in NR1 format
:SBUS:IIC:ASize <size> (see page 328)	:SBUS:IIC:ASIZE? (see page 328)	<size> ::= {BIT7 BIT8}
:SBUS:LIN:PARity {{0 OFF} {1 ON}} (see page 329)	:SBUS:LIN:PARity? (see page 329)	{0 1}
:SBUS:MODE <mode> (see page 330)	:SBUS:MODE? (see page 330)	<mode> ::= {IIC SPI CAN LIN FLEXray UART}
:SBUS:SPI:WIDTh <word_width> (see page 331)	:SBUS:SPI:WIDTh? (see page 331)	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see page 332)	:SBUS:UART:BASE? (see page 332)	<base> ::= {ASCIi BINary HEX}
n/a	:SBUS:UART:COUNT:ERROr? (see page 333)	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESet (see page 334)	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFRames? (see page 335)	<frame_count> ::= integer in NR1 format

2 Commands Quick Reference

Table 18 :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:UART:COUNT:TXFRAMES? (see page 336)	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see page 337)	:SBUS:UART:FRAMing? (see page 337)	<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 page 339)	:SYSTem:DATE? (see page 339)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,...,12 JANuary FEBruary MARch APRil MAY JUNE JULy AUGust SEPtember OCTober NOVember DECember} <day> ::= {1,..31}
:SYSTem:DSP <string> (see page 340)	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 341)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 585).
:SYSTem:LOCK (see page 342)	:SYSTem:LOCK? (see page 342)	<value> ::= {ON OFF}
:SYSTem:SETup <setup_data> (see page 343)	:SYSTem:SETup? (see page 343)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 345)	:SYSTem:TIME? (see page 345)	<time> ::= hours,minutes,seconds in NR1 format

Table 20 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 348)	:TIMebase:MODE? (see page 348)	<value> ::= {MAIN WINDow XY ROLL}
:TIMebase:POSition <pos> (see page 349)	:TIMebase:POSition? (see page 349)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 350)	:TIMebase:RANGe? (see page 350)	<range_value> ::= 5 ns through 500 s in NR3 format
:TIMebase:REFClock {0 OFF} {1 ON} (see page 351)	:TIMebase:REFClock? (see page 351)	{0 1}
:TIMebase:REFerence {LEFT CENTER RIGHT} (see page 352)	:TIMebase:REFerence? (see page 352)	<return_value> ::= {LEFT CENTER RIGHT}
:TIMebase:SCALe <scale_value> (see page 353)	:TIMebase:SCALe? (see page 353)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0 OFF} {1 ON}} (see page 354)	:TIMebase:VERNier? (see page 354)	{0 1}
:TIMebase:WINDow:POSition <pos> (see page 355)	:TIMebase:WINDow:POSition? (see page 355)	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 356)	:TIMebase:WINDow:RANGe? (see page 356)	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCALe <scale_value> (see page 357)	:TIMebase:WINDow:SCALe? (see page 357)	<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 362)	:TRIGger:HFReject? (see page 362)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 363)	:TRIGger:HOLDoff? (see page 363)	<holdoff_time> ::= 60 ns to 10 s in NR3 format

Table 21 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:MODE <mode> (see page 364)	:TRIGger:MODE? (see page 364)	<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 365)	:TRIGger:NREJect? (see page 365)	{0 1}
:TRIGger:PATtern <value>, <mask> [, <edge source>, <edge>] (see page 366)	:TRIGger:PATtern? (see page 367)	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnxxxx"; 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 368)	:TRIGger:SWEep? (see page 368)	<sweep> ::= {AUTO NORMal}

Table 22 :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATtern:DATA <value>, <mask> (see page 371)	:TRIGger:CAN:PATtern:DATA? (see page 371)	<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 372)	:TRIGger:CAN:PATtern:DATA:LENGth? (see page 372)	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATtern:ID <value>, <mask> (see page 373)	:TRIGger:CAN:PATtern:ID? (see page 373)	<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 page 374)	:TRIGger:CAN:PATtern:ID:MODE? (see page 374)	<value> ::= {STANdard EXTended} (with Option AMS)
:TRIGger:CAN:SAMPlepo int <value> (see page 375)	:TRIGger:CAN:SAMPlepo int? (see page 375)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:TRIGger:CAN:SIGNAL:BAUDrate <baudrate> (see page 376)	:TRIGger:CAN:SIGNAL:BAUDrate? (see page 376)	<baudrate> ::= {10000 20000 33300 50000 62500 83300 100000 125000 250000 500000 800000 1000000}

2 Commands Quick Reference

Table 22 :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see page 377)	:TRIGger:CAN:SOURce? (see page 377)	<source> ::= {CHANnel<n> EXTErnal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 378)	:TRIGger:CAN:TRIGger? (see page 379)	<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 page 381)	:TRIGger:DURation:GREaterthan? (see page 381)	<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 page 382)	:TRIGger:DURation:LESSthan? (see page 382)	<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 page 383)	:TRIGger:DURation:PATTERN? (see page 383)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn...n" n ::= {0,...,9 A,...,F}
:TRIGger:DURation:QUALifier <qualifier> (see page 384)	:TRIGger:DURation:QUALifier? (see page 384)	<qualifier> ::= {GREaterthan LESSthan INRange OUTRange TIMEout}
:TRIGger:DURation:RANGE <greater than time>[suffix], <less than time>[suffix] (see page 385)	:TRIGger:DURation:RANGE? (see page 385)	<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 page 387)	:TRIGger:EBURst:COUNT ? (see page 387)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 388)	:TRIGger:EBURst:IDLE? (see page 388)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 389)	:TRIGger:EBURst:SLOPe ? (see page 389)	<slope> ::= {NEGative POSitive}

Table 25 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPl ing {AC DC LF} (see page 391)	:TRIGger[:EDGE]:COUPl ing? (see page 391)	{AC DC LF}
:TRIGger[:EDGE]:LEVEl <level> [, <source>] (see page 392)	:TRIGger[:EDGE]:LEVEl ? [<source>] (see page 392)	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]:REJec t {OFF LF HF} (see page 393)	:TRIGger[:EDGE]:REJec t? (see page 393)	{OFF LF HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 394)	:TRIGger[:EDGE]:SLOPe ? (see page 394)	<polarity> ::= {POSitive NEGative EITHer ALTErnate}
:TRIGger[:EDGE]:SOURc e <source> (see page 395)	:TRIGger[:EDGE]:SOURc e? (see page 395)	<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

2 Commands Quick Reference

Table 26 :TRIGger:FLEXray Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FLEXray:ERRor:TYPE <error_type> (see page 397)	:TRIGger:FLEXray:ERRor:TYPE? (see page 397)	<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 page 399)	:TRIGger:FLEXray:FRAMe:CCBase? (see page 399)	<cycle_count_base> ::= integer from 0-63
:TRIGger:FLEXray:FRAMe:CCRepetition <cycle_count_repetition> (see page 400)	:TRIGger:FLEXray:FRAMe:CCRepetition? (see page 400)	<cycle_count_repetition> ::= {ALL <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:FRAMe:ID <frame_id> (see page 401)	:TRIGger:FLEXray:FRAMe:ID? (see page 401)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:TRIGger:FLEXray:FRAMe:TYPE <frame_type> (see page 402)	:TRIGger:FLEXray:FRAMe:TYPE? (see page 402)	<frame_type> ::= {NORMAL STARTup NULL SYNC NSTARTup NNULl NSYNc ALL}
:TRIGger:FLEXray:TIME:CBASE <cycle_base> (see page 403)	:TRIGger:FLEXray:TIME:CBASE? (see page 403)	<cycle_base> ::= integer from 0-63
:TRIGger:FLEXray:TIME:CREPpetition <cycle_repetition> (see page 404)	:TRIGger:FLEXray:TIME:CREPpetition? (see page 404)	<cycle_repetition> ::= {ALL <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:TIME:SEGMENT <segment_type> (see page 405)	:TRIGger:FLEXray:TIME:SEGMENT? (see page 405)	<segment_type> ::= {STATIC DYNAMIC SYMbol IDLE}
:TRIGger:FLEXray:TIME:SLOT <slot_type>, <slot_id> (see page 406)	:TRIGger:FLEXray:TIME:SLOT? (see page 406)	<slot_type> ::= {ALL EMPTY} <slot_id> ::= {ALL <slot #>} <slot #> ::= integer from 1-2047
:TRIGger:FLEXray:TRIGger <condition> (see page 407)	:TRIGger:FLEXray:TRIGger? (see page 407)	<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 page 410)	:TRIGger:GLITch:GREAt erthan? (see page 410)	<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 page 411)	:TRIGger:GLITch:LESSt han? (see page 411)	<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 page 412)	:TRIGger:GLITch:LEVel ? (see page 412)	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 page 413)	:TRIGger:GLITch:POLAr ity? (see page 413)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALi fier <qualifier> (see page 414)	:TRIGger:GLITch:QUALi fier? (see page 414)	<qualifier> ::= {GREAterthan LESSthan RANGE}

2 Commands Quick Reference

Table 27 :TRIGger:GLITCh Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITCh:RANGe <greater than time>[suffix], <less than time>[suffix] (see page 415)	:TRIGger:GLITCh:RANGe? (see page 415)	<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:SOURC e <source> (see page 416)	:TRIGger:GLITCh:SOURC e? (see page 416)	<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:ADDEss <value> (see page 418)	:TRIGger:IIC:PATTErn:ADDEss? (see page 418)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC:PATTErn:DATA <value> (see page 419)	:TRIGger:IIC:PATTErn:DATA? (see page 419)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC:PATTErn:DATA2 <value> (see page 420)	:TRIGger:IIC:PATTErn:DATA2? (see page 420)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC[:SOURC e]:CLOCK <source> (see page 421)	:TRIGger:IIC[:SOURC e]:CLOCK? (see page 421)	<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[:SOURC e]:DATA <source> (see page 422)	:TRIGger:IIC[:SOURC e]:DATA? (see page 422)	<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 (continued)

Command	Query	Options and Query Returns
:TRIGger:IIC:TRIGger:QUALifier <value> (see page 423)	:TRIGger:IIC:TRIGger:QUALifier? (see page 423)	<value> ::= {EQUAL NOTequal LESSthan GREATERthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see page 424)	:TRIGger:IIC:TRIGger[:TYPE]? (see page 424)	<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 page 427)	:TRIGger:LIN:ID? (see page 427)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:TRIGger:LIN:SAMplepo int <value> (see page 428)	:TRIGger:LIN:SAMplepo int? (see page 428)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:TRIGger:LIN:SIGNAL:B AUDrate <baudrate> (see page 429)	:TRIGger:LIN:SIGNAL:B AUDrate? (see page 429)	<baudrate> ::= {2400 9600 19200}
:TRIGger:LIN:SOURce <source> (see page 430)	:TRIGger:LIN:SOURce? (see page 430)	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std> (see page 431)	:TRIGger:LIN:STANdard ? (see page 431)	<std> ::= {LIN13 LIN20}
:TRIGger:LIN:SYNCbrea k <value> (see page 432)	:TRIGger:LIN:SYNCbrea k? (see page 432)	<value> ::= integer = {11 12 13}
:TRIGger:LIN:TRIGger <condition> (see page 433)	:TRIGger:LIN:TRIGger? (see page 433)	<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 page 435)	:TRIGger:SEQuence:COU Nt? (see page 435)	<count> ::= integer in NR1 format
:TRIGger:SEQuence:EDG E{1 2} <source>, <slope> (see page 436)	:TRIGger:SEQuence:EDG E{1 2}? (see page 436)	<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 page 437)	:TRIGger:SEQuence:FIN D? (see page 437)	<value> ::= {PATtern1,ENTerEd PATtern1,EXITed EDGE1 PATtern1,AND,EDGE1}
:TRIGger:SEQuence:PAT Tern{1 2} <value>, <mask> (see page 438)	:TRIGger:SEQuence:PAT Tern{1 2}? (see page 438)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xn timer" n ::= {0,...,9 A,...,F}
:TRIGger:SEQuence:RES et <value> (see page 439)	:TRIGger:SEQuence:RES et? (see page 439)	<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 page 440)	:TRIGger:SEQuence:TIM er? (see page 440)	<time_value> ::= time from 100 ns to 10 seconds in NR3 format
:TRIGger:SEQuence:TRI Gger <value> (see page 441)	:TRIGger:SEQuence:TRI Gger? (see page 441)	<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 443)	:TRIGger:SPI:CLOCK:SL OPe? (see page 443)	<slope> ::= {NEGative POSitive}
:TRIGger:SPI:CLOCK:TI Meout <time_value> (see page 444)	:TRIGger:SPI:CLOCK:TI Meout? (see page 444)	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 445)	:TRIGger:SPI:FRAMing? (see page 445)	<value> ::= {CHIPselect NOTChipselect TIMEout}
:TRIGger:SPI:PATtern: DATA <value>, <mask> (see page 446)	:TRIGger:SPI:PATtern: DATA? (see page 446)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnxxxxx" where n ::= {0,...,9 A,...,F}
:TRIGger:SPI:PATtern: WIDTh <width> (see page 447)	:TRIGger:SPI:PATtern: WIDTh? (see page 447)	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:C LOCK <source> (see page 448)	:TRIGger:SPI:SOURce:C LOCK? (see page 448)	<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 449)	:TRIGger:SPI:SOURce:D ATA? (see page 449)	<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 450)	:TRIGger:SPI:SOURce:F RAME? (see page 450)	<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

2 Commands Quick Reference

Table 32 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 452)	:TRIGger:TV:LINE? (see page 452)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 453)	:TRIGger:TV:MODE? (see page 453)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LINE VERTICAL LFIELD1 LFIELD2 LALTERNATE LVERTICAL}
:TRIGger:TV:POLarity <polarity> (see page 454)	:TRIGger:TV:POLarity? (see page 454)	<polarity> ::= {POSITIVE NEGATIVE}
:TRIGger:TV:SOURce <source> (see page 455)	:TRIGger:TV:SOURce? (see page 455)	<source> ::= {CHANNEL<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard> (see page 456)	:TRIGger:TV:STANdard? (see page 456)	<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:BAUDrat e <baudrate> (see page 459)	:TRIGger:UART:BAUDrat e? (see page 459)	<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:BITorde r <bitorder> (see page 460)	:TRIGger:UART:BITorde r? (see page 460)	<bitorder> ::= {LSBFirst MSBFirst}
:TRIGger:UART:BURSt <value> (see page 461)	:TRIGger:UART:BURSt? (see page 461)	<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 462)	:TRIGger:UART:DATA? (see page 462)	<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 463)	:TRIGger:UART:IDLE? (see page 463)	<time_value> ::= time from 10 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 464)	:TRIGger:UART:PARity? (see page 464)	<parity> ::= {EVEN ODD NONE}
:TRIGger:UART:POLarity <polarity> (see page 465)	:TRIGger:UART:POLarity? (see page 465)	<polarity> ::= {HIGH LOW}
:TRIGger:UART:QUALifier <value> (see page 466)	:TRIGger:UART:QUALifier? (see page 466)	<value> ::= {EQUAL NOTequal GREATERthan LESSthan}
:TRIGger:UART:SOURce: RX <source> (see page 467)	:TRIGger:UART:SOURce: RX? (see page 467)	<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 468)	:TRIGger:UART:SOURce: TX? (see page 468)	<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 469)	:TRIGger:UART:TYPE? (see page 469)	<value> ::= {RSTArt RSTop RDATA RD1 RD0 RDX PARityerror TSTArt TSTop TDATA TD1 TD0 TDX}
:TRIGger:UART:WIDTH <width> (see page 470)	:TRIGger:UART:WIDTH? (see page 470)	<width> ::= {5 6 7 8 9}

2 Commands Quick Reference

Table 34 :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:D MINus <source> (see page 472)	:TRIGger:USB:SOURce:D MINus? (see page 472)	<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 473)	:TRIGger:USB:SOURce:D PLus? (see page 473)	<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 474)	:TRIGger:USB:SPEEd? (see page 474)	<value> ::= {LOW FULL}
:TRIGger:USB:TRIGger <value> (see page 475)	:TRIGger:USB:TRIGger? (see page 475)	<value> ::= {SOP EOP ENTersuspend EXITsuspend RESet}

Table 35 :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see page 484)	:WAVEform:BYTeorder? (see page 484)	<value> ::= {LSBFirst MSBFirst}
n/a	:WAVEform:COUNT? (see page 485)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see page 486)	<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 488)	:WAVEform:FORMat? (see page 488)	<value> ::= {WORD BYTE ASCII}

Table 35 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVEform:POINTs <# points> (see page 489)	:WAVEform:POINTs? (see page 489)	<# points> ::= {100 250 500 1000 <points_mode>} if waveform points mode is NORMAl <# points> ::= {100 250 500 1000 2000 ... 8000000 in 1-2-5 sequence <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl MAXimum RAW}
:WAVEform:POINTs:MODE <points_mode> (see page 491)	:WAVEform:POINTs:MODE ? (see page 492)	<points_mode> ::= {NORMAl MAXimum RAW}
n/a	:WAVEform:PREamble? (see page 493)	<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"> • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: <ul style="list-style-type: none"> • 0 for NORMAl type • 1 for PEAK detect type • 2 for AVERAge type • 3 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMAl; an integer in NR1 format
n/a	:WAVEform:SEGmented:COUNT? (see page 496)	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see page 497)	<time_tag> ::= in NR3 format (with Option SGM)

2 Commands Quick Reference

Table 35 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SOURce <source> (see page 498)	:WAVeform:SOURce? (see page 498)	<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 page 502)	:WAVeform:SOURce:SUBS ource? (see page 502)	<subsource> ::= {{NONE RX} TX}
n/a	:WAVeform:TYPE? (see page 503)	<return_mode> ::= {NORM PEAK AVER HRES}
:WAVeform:UNSigned {{0 OFF} {1 ON}} (see page 504)	:WAVeform:UNSigned? (see page 504)	{0 1}
:WAVeform:VIEW <view> (see page 505)	:WAVeform:VIEW? (see page 505)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 506)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 507)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 508)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 509)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 510)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 511)	<return_value> ::= y-reference value in the current preamble in NR1 format

Syntax Elements

- "Number Format" on page 63
- "<NL> (Line Terminator)" on page 63
- "[] (Optional Syntax Terms)" on page 63
- "{ } (Braces)" on page 63
- " ::= (Defined As)" on page 63
- "< > (Angle Brackets)" on page 64
- "... (Ellipsis)" on page 64
- "n,...,p (Value Ranges)" on page 64
- "d (Digits)" on page 64
- "Quoted ASCII String" on page 64
- "Definite-Length Block Response Data" on page 64

Number Format

NR1 specifies integer data.

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

<NL> (Line Terminator)

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

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

[] (Optional Syntax Terms)

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

{ } (Braces)

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

::= (Defined As)

::= means "defined as".

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

< > (Angle Brackets)

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

... (Ellipsis)

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

n,...,p (Value Ranges)

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

d (Digits)

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

Quoted ASCII String

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

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

has a quoted ASCII string of:

```
'One'
```

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

Definite-Length Block Response Data

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

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

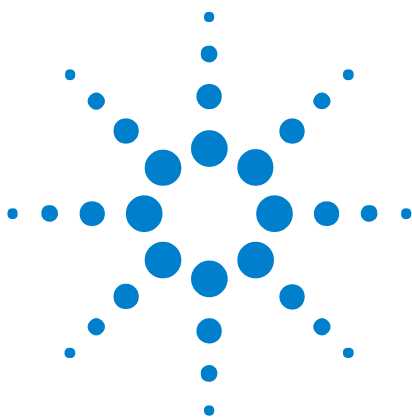

```
#800001000<1000 bytes of data> <NL>
```

8 is the number of digits that follow

00001000 is the number of bytes to be transmitted

<1000 bytes of data> is the actual data

2 Commands Quick Reference



3 Commands by Subsystem

Subsystem	Description
"Common (*) Commands" on page 69	Commands defined by IEEE 488.2 standard that are common to all instruments.
"Root (:) Commands" on page 94	Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.
":ACQuire Commands" on page 132	Set the parameters for acquiring and storing data.
":BUS<n> Commands" on page 147	Control all oscilloscope functions associated with the digital channels bus display.
":CALibrate Commands" on page 156	Utility commands for determining the state of the calibration factor protection switch.
":CHANnel<n> Commands" on page 164	Control all oscilloscope functions associated with individual analog channels or groups of channels.
":DIGital<n> Commands" on page 183	Control all oscilloscope functions associated with individual digital channels.
":DISPlay Commands" on page 190	Control how waveforms, graticule, and text are displayed and written on the screen.
":EXTernal Trigger Commands" on page 200	Control the input characteristics of the external trigger input.
":FUNction Commands" on page 210	Control functions in the measurement/storage module.
":HARDcopy Commands" on page 222	Set and query the selection of hardcopy device and formatting options.
":MARKer Commands" on page 232	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 243	Select automatic measurements to be made and control time markers.



3 Commands by Subsystem

Subsystem	Description
":POD Commands" on page 287	Control all oscilloscope functions associated with groups of digital channels.
":RECall Commands" on page 292	Recall previously saved oscilloscope setups and traces.
":SAVE Commands" on page 297	Save oscilloscope setups and traces, screen images, and data.
":SBUS Commands" on page 311	Control oscilloscope functions associated with the serial decode bus.
":SYSTem Commands" on page 338	Control basic system functions of the oscilloscope.
":TIMebase Commands" on page 346	Control all horizontal sweep functions.
":TRIGger Commands" on page 358	Control the trigger modes and parameters for each trigger type.
":WAVEform Commands" on page 476	Provide access to waveform data.

Command Types Three types of commands are used:

- **Common (*) Commands** – See ["Introduction to Common \(*\) Commands"](#) on page 71 for more information.
- **Root Level (:) Commands** – See ["Introduction to Root \(: \) Commands"](#) on page 96 for more information.
- **Subsystem Commands** – Subsystem commands are grouped together under a common node of the ["Command Tree"](#) on page 621, 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 71.

Table 36 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 73)	n/a	n/a
*ESE <mask> (see page 74)	*ESE? (see page 75)	<p><mask> ::= 0 to 255; an integer in NR1 format:</p> <pre> Bit Weight Name Enables ----- 7 128 PON Power On 6 64 URQ User Request 5 32 CME Command Error 4 16 EXE Execution Error 3 8 DDE Dev. Dependent Error 2 4 QYE Query Error 1 2 RQL Request Control 0 1 OPC Operation Complete </pre>
n/a	*ESR? (see page 76)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 76)	<p>AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX</p> <p><model> ::= the model number of the instrument</p> <p><serial number> ::= the serial number of the instrument</p> <p><X.XX.XX> ::= the software revision of the instrument</p>
n/a	*LRN? (see page 79)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 80)	*OPC? (see page 80)	ASCII "1" is placed in the output queue when all pending device operations have completed.

3 Commands by Subsystem

Table 36 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*OPT? (see page 81)	<pre> <return_value> ::= 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} </pre>
*RCL <value> (see page 82)	n/a	<pre> <value> ::= {0 1 2 3 4 5 6 7 8 9} </pre>
*RST (see page 83)	n/a	See *RST (Reset) (see page 83)
*SAV <value> (see page 86)	n/a	<pre> <value> ::= {0 1 2 3 4 5 6 7 8 9} </pre>
*SRE <mask> (see page 87)	*SRE? (see page 88)	<pre> <mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables ----- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger </pre>

Table 36 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	*STB? (see page 89)	<p><value> ::= 0 to 255; an integer in NR1 format, as shown in the following:</p> <pre> Bit Weight Name "1" Indicates ----- 7 128 OPER Operation status condition occurred. 6 64 RQS/ Instrument is MSS requesting service. 5 32 ESB Enabled event status condition occurred. 4 16 MAV Message available. 3 8 ---- (Not used.) 2 4 MSG Message displayed. 1 2 USR User event condition occurred. 0 1 TRG A trigger occurred. </pre>
*TRG (see page 91)	n/a	n/a
n/a	*TST? (see page 92)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 93)	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.

3 Commands by Subsystem

NOTE

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

*CLS (Clear Status)

C (see [page 616](#))

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 71
 - "[*STB \(Read Status Byte\)](#)" on page 89
 - "[*ESE \(Standard Event Status Enable\)](#)" on page 74
 - "[*ESR \(Standard Event Status Register\)](#)" on page 76
 - "[*SRE \(Service Request Enable\)](#)" on page 87
 - "[:SYSTem:ERRor](#)" on page 341

*ESE (Standard Event Status Enable)

C (see page 616)

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.

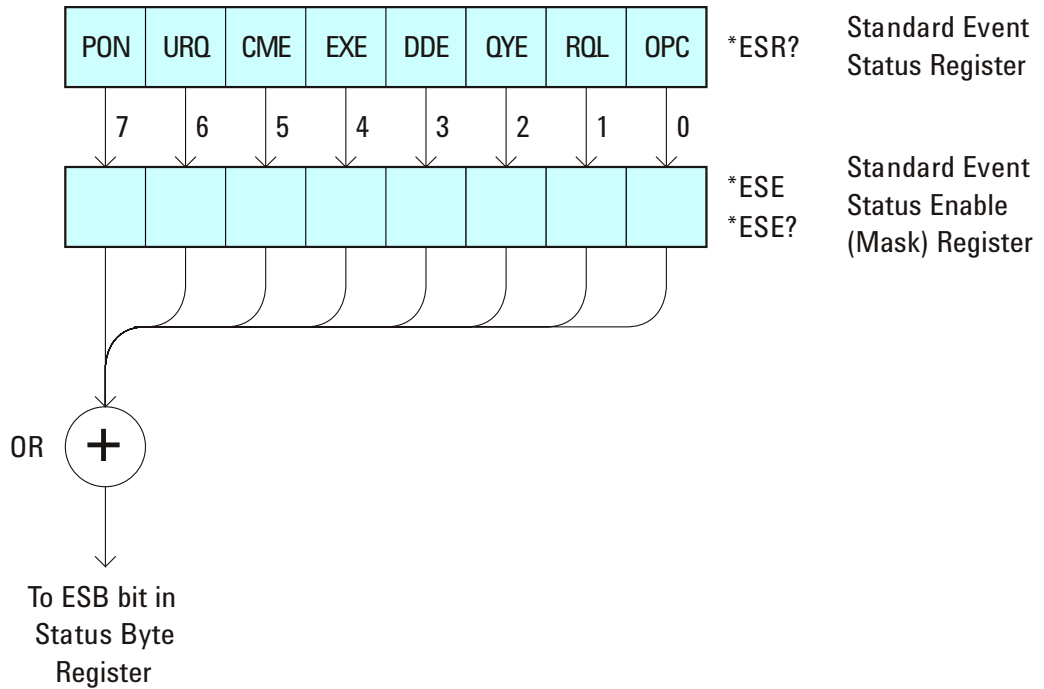


Table 37 Standard Event Status Enable (ESE)

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 71
 - ["*ESR \(Standard Event Status Register\)"](#) on page 76
 - ["*OPC \(Operation Complete\)"](#) on page 80
 - ["*CLS \(Clear Status\)"](#) on page 73

*ESR (Standard Event Status Register)

C (see page 616)

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.

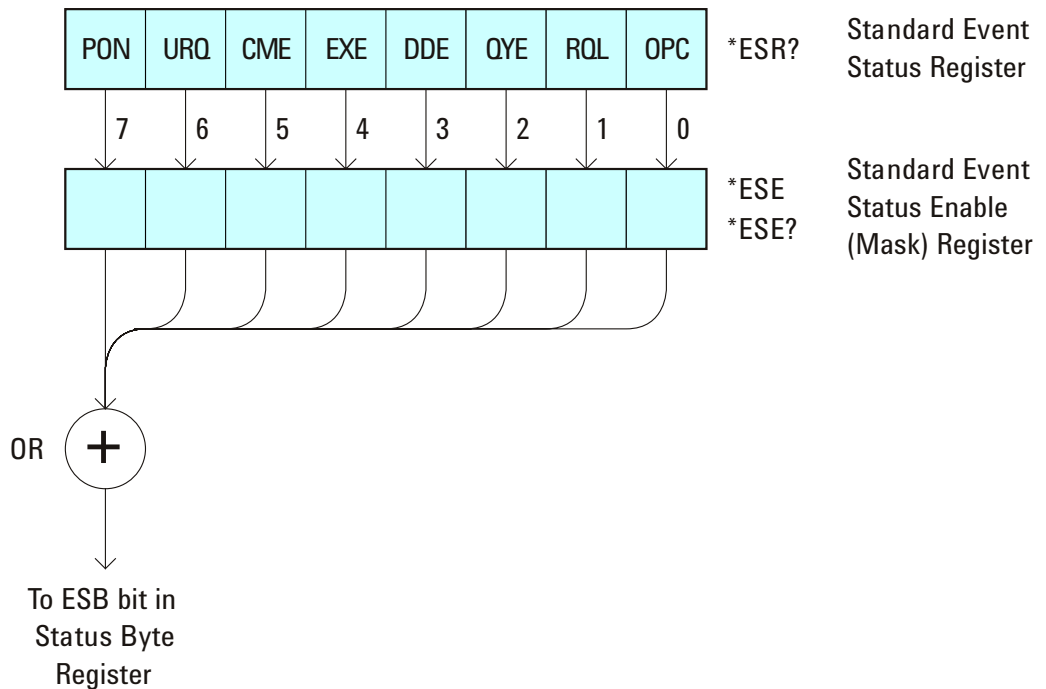


Table 38 Standard Event Status Register (ESR)

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 71
 - ["*ESE \(Standard Event Status Enable\)"](#) on page 74
 - ["*OPC \(Operation Complete\)"](#) on page 80
 - ["*CLS \(Clear Status\)"](#) on page 73
 - [":SYSTem:ERRor"](#) on page 341

***IDN (Identification Number)**

C (see [page 616](#))

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 71
 - ["*OPT \(Option Identification\)"](#) on page 81

*LRN (Learn Device Setup)

C (see [page 616](#))

Query Syntax

*LRN?

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTEM:SETup? (see [page 343](#)) 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 71
- "[*RCL \(Recall\)](#)" on page 82
- "[*SAV \(Save\)](#)" on page 86
- "[:SYSTEM:SETup](#)" on page 343

*OPC (Operation Complete)

C (see [page 616](#))

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 71
 - ["*ESE \(Standard Event Status Enable\)"](#) on page 74
 - ["*ESR \(Standard Event Status Register\)"](#) on page 76
 - ["*CLS \(Clear Status\)"](#) on page 73

*OPT (Option Identification)

C (see [page 616](#))

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 71
 - ["*IDN \(Identification Number\)"](#) on page 78

*RCL (Recall)

C (see [page 616](#))

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 71
 - ["*SAV \(Save\)"](#) on page 86

*RST (Reset)

C (see page 616)

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

3 Commands by Subsystem

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 71

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 6"](#) on page 692

***SAV (Save)**

C (see [page 616](#))

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 71
 - ["*RCL \(Recall\)"](#) on page 82

*SRE (Service Request Enable)

C (see page 616)

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.

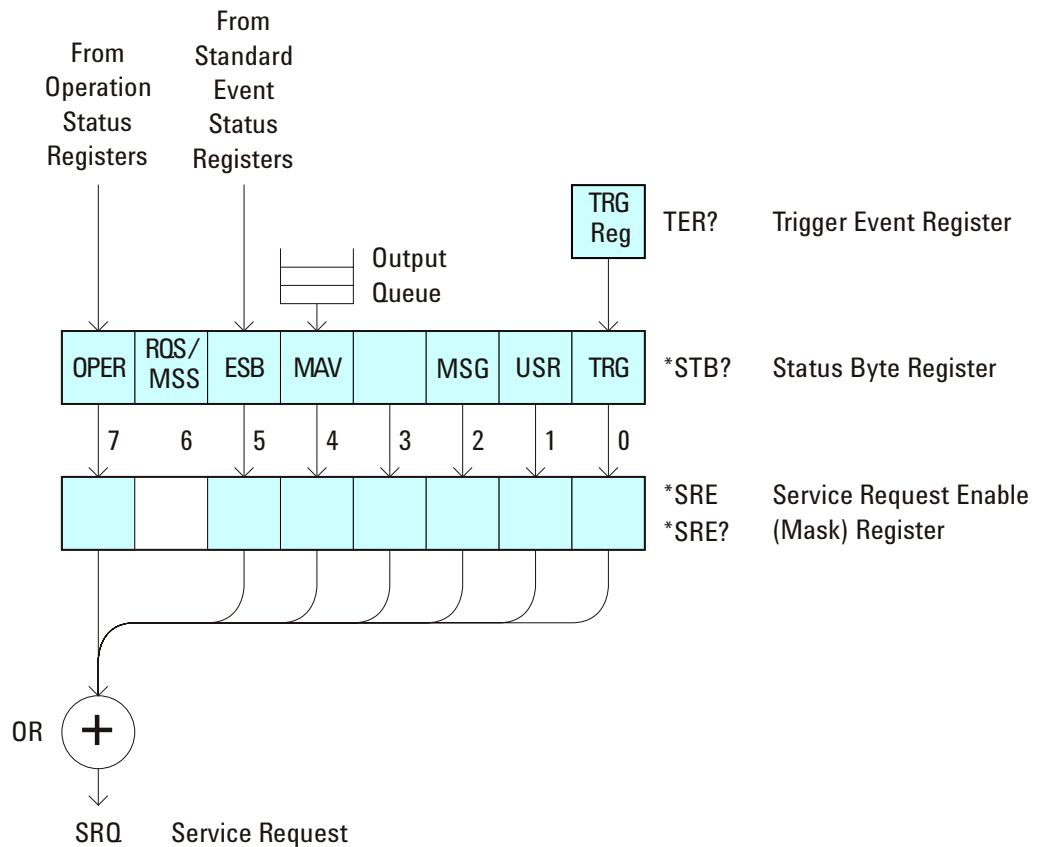


Table 39 Service Request Enable Register (SRE)

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 71
 - ["*STB \(Read Status Byte\)"](#) on page 89
 - ["*CLS \(Clear Status\)"](#) on page 73

*STB (Read Status Byte)

C (see page 616)

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

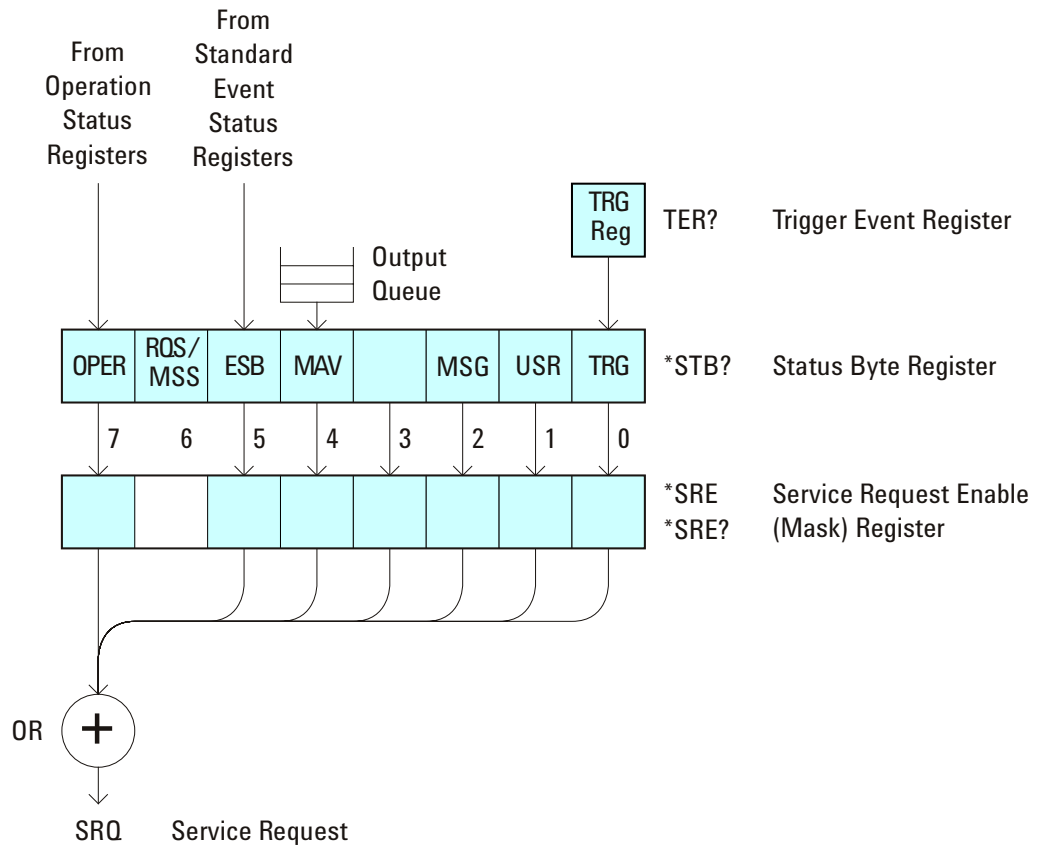


Table 40 Status Byte Register (STB)

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 71
 - ["*SRE \(Service Request Enable\)"](#) on page 87

*TRG (Trigger)

C (see [page 616](#))

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 71
 - [":DIGitize"](#) on page 105
 - [":RUN"](#) on page 125
 - [":STOP"](#) on page 129

***TST (Self Test)**

C (see [page 616](#))

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 71

*WAI (Wait To Continue)

C (see [page 616](#))

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 71

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 96.

Table 41 Root (:) Commands Summary

Command	Query	Options and Query Returns
:ACTivity (see page 97)	:ACTivity? (see page 97)	<return value> ::= <edges>,<levels> <edges> ::= presence of edges (32-bit integer in NR1 format) <levels> ::= logical highs or lows (32-bit integer in NR1 format)
n/a	:AER? (see page 98)	{0 1}; an integer in NR1 format
:AUToscale [<source>[,...,<source>]] (see page 99)	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 page 101)	:AUToscale:AMODE? (see page 101)	<value> ::= {NORMAL CURRENT}}
:AUToscale:CHANnels <value> (see page 102)	:AUToscale:CHANnels? (see page 102)	<value> ::= {ALL DISPLAYed}}
:BLANK [<source>] (see page 103)	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 page 104)	n/a	n/a

Table 41 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,...,<source>]] (see page 105)	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 page 107)	:HWEenable? (see page 107)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister:CONDition? (see page 109)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENT]? (see page 111)	<n> ::= 16-bit integer in NR1 format
:MERGE <pixel memory> (see page 113)	n/a	<pixel memory> ::= {PMEMory{0 1 2 3 4 5 6 7 8 9}}
:OPEE <n> (see page 114)	:OPEE? (see page 115)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister:CONDition? (see page 116)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERRegister[:EVENT]? (see page 118)	<n> ::= 16-bit integer in NR1 format
:OVLenable <mask> (see page 120)	:OVLenable? (see page 121)	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input ----- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see page 122)	<value> ::= integer in NR1 format. See OVLenable for <value>

3 Commands by Subsystem

Table 41 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:PRINT [<options>] (see page 124)	n/a	<options> ::= [<print option>][,...,<print option>] <print option> ::= {COLor GRAYscale PRINter0 BMP8bit BMP PNG NOFactoRs FACToRs} <print option> can be repeated up to 5 times.
:RUN (see page 125)	n/a	n/a
n/a	:SERial (see page 126)	<return value> ::= unquoted string containing serial number
:SINGle (see page 127)	n/a	n/a
n/a	:STATus? <display> (see page 128)	{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 page 129)	n/a	n/a
n/a	:TER? (see page 130)	{0 1}
:VIEW <source> (see page 131)	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 616](#))

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 96
 - [":POD<n>:THReshold"](#) on page 290
 - [":DIGital<n>:THReshold"](#) on page 189

:AER (Arm Event Register)

C (see [page 616](#))

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 96
- [":OPEE \(Operation Status Enable Register\)"](#) on page 114
- [":OPERRegister:CONDition \(Operation Status Condition Register\)"](#) on page 116
- [":OPERRegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 118
- ["*STB \(Read Status Byte\)"](#) on page 89
- ["*SRE \(Service Request Enable\)"](#) on page 87

:AUToscale

C (see [page 616](#))

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 102) 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*.

3 Commands by Subsystem

- See Also**
- ["Introduction to Root \(:\) Commands"](#) on page 96
 - [":AUToscale:CHANnels"](#) on page 102
 - [":AUToscale:AMODE"](#) on page 101

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 6"](#)
on page 692

:AUToscale:AMODE

N (see [page 616](#))

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 96
 - "[:AUToscale](#)" on page 99
 - "[:AUToscale:CHANnels](#)" on page 102
 - "[:ACQUIRE:TYPE](#)" on page 145
 - "[:ACQUIRE:MODE](#)" on page 138

:AUToscale:CHANnels

N (see [page 616](#))

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 96
 - [":AUToscale"](#) on page 99
 - [":AUToscale:AMODE"](#) on page 101
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103

:BLANK

N (see [page 616](#))

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 96
- [":CDISplay"](#) on page 104
- [":CHANnel<n>:DISPlay"](#) on page 169
- [":DIGital<n>:DISPlay"](#) on page 185
- [":FUNCTION:DISPlay"](#) on page 213
- [":POD<n>:DISPlay"](#) on page 288
- [":STATus"](#) on page 128
- [":VIEW"](#) on page 131

Example Code

- ["Example Code"](#) on page 131

:CDISplay

C (see [page 616](#))

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 96
 - [":DISPlay:CLEar"](#) on page 192

:DIGitize

C (see [page 616](#))

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 96
 - "[:RUN](#)" on page 125
 - "[:SINGLE](#)" on page 127
 - "[:STOP](#)" on page 129
 - "[:ACQUIRE Commands](#)" on page 132
 - "[:WAVEFORM Commands](#)" on page 476

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
```

3 Commands by Subsystem

```
' 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 6"](#)
on page 692

:HWEenable (Hardware Event Enable Register)

N (see page 616)

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.

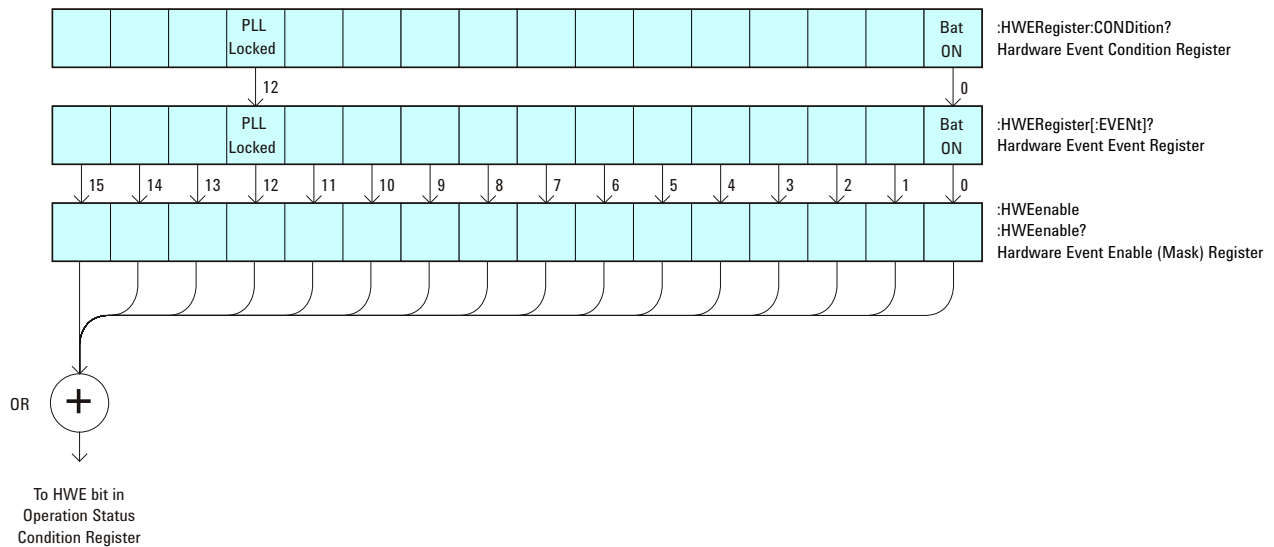


Table 42 Hardware Event Enable Register (HWEenable)

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.

3 Commands by Subsystem

- See Also**
- "Introduction to Root (:) Commands" on page 96
 - ":AER (Arm Event Register)" on page 98
 - ":CHANnel<n>:PROTection" on page 178
 - ":EXTeRnal:PROTection" on page 207
 - ":OPERegister[:EVENT] (Operation Status Event Register)" on page 118
 - ":OVLenable (Overload Event Enable Register)" on page 120
 - ":OVLRegister (Overload Event Register)" on page 122
 - "**STB (Read Status Byte)" on page 89
 - "**SRE (Service Request Enable)" on page 87

:HWERegister:CONDition (Hardware Event Condition Register)

N (see page 616)

Query Syntax :HWERegister:CONDition?

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

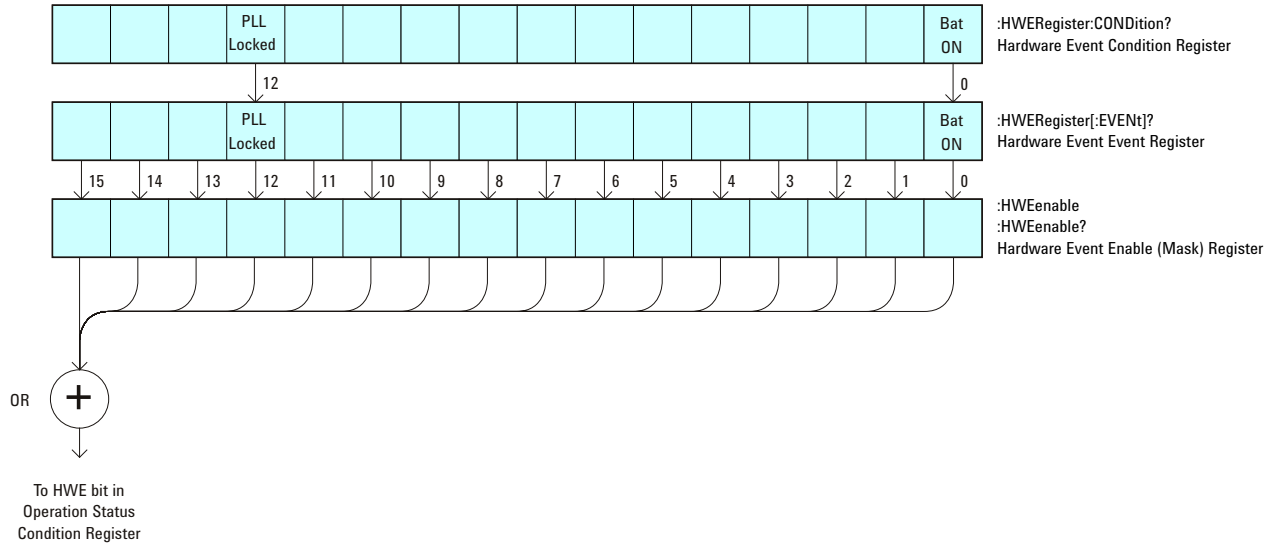


Table 43 Hardware Event Condition Register

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 96
 - ":CHANnel<n>:PROTection" on page 178
 - ":EXTErnal:PROTection" on page 207
 - ":OPEE (Operation Status Enable Register)" on page 114
 - ":OPERegister[:EVENT] (Operation Status Event Register)" on page 118

3 Commands by Subsystem

- ":OVLenable (Overload Event Enable Register)" on page 120
- ":OVLRegister (Overload Event Register)" on page 122
- "*STB (Read Status Byte)" on page 89
- "*SRE (Service Request Enable)" on page 87

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

N (see page 616)

Query Syntax :HWERegister[:EVENT]?

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

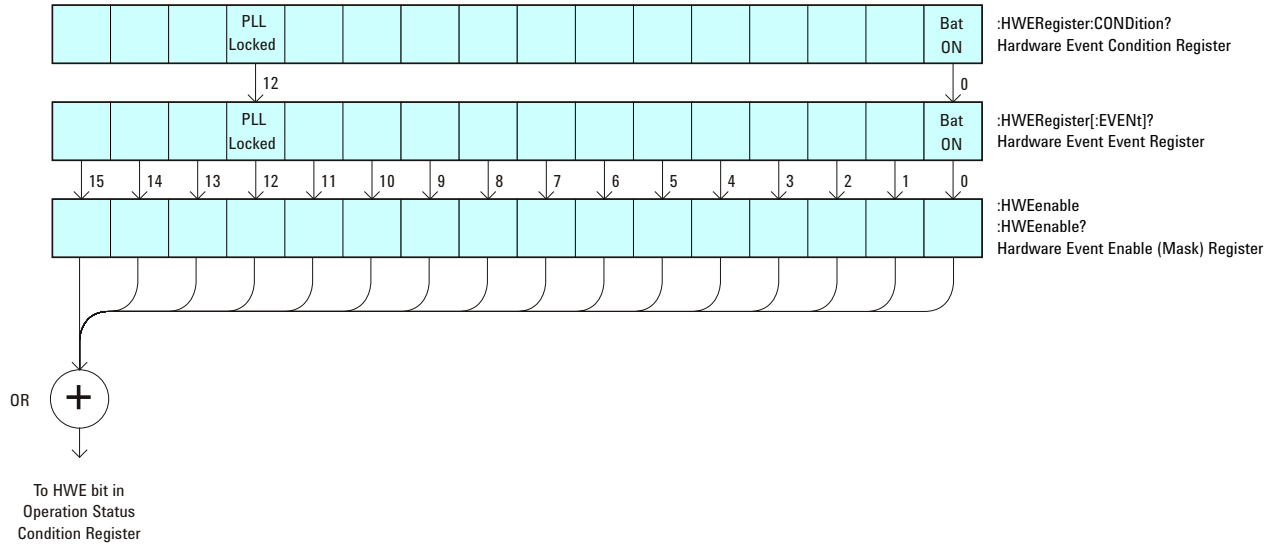


Table 44 Hardware Event Event Register

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 96
 - ":CHANnel<n>:PROTection" on page 178
 - ":EXTeRnal:PROTection" on page 207
 - ":OPEE (Operation Status Enable Register)" on page 114

3 Commands by Subsystem

- ":OPERRegister:CONDition (Operation Status Condition Register)" on page 116
- ":OVLenable (Overload Event Enable Register)" on page 120
- ":OVLRegister (Overload Event Register)" on page 122
- "*STB (Read Status Byte)" on page 89
- "*SRE (Service Request Enable)" on page 87

:MERGe

N (see [page 616](#))

Command Syntax :MERGe <pixel memory>

```
<pixel memory> ::= {PMEMemory0 | PMEMemory1 | PMEMemory2 | PMEMemory3
                    | PMEMemory4 | PMEMemory5 | PMEMemory6 | PMEMemory7
                    | PMEMemory8 | PMEMemory9}
```

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 PMEMemory0 through PMEMemory9. 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 96
 - ["*SAV \(Save\)"](#) on page 86
 - ["*RCL \(Recall\)"](#) on page 82
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103

:OPEE (Operation Status Enable Register)

C (see page 616)

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.

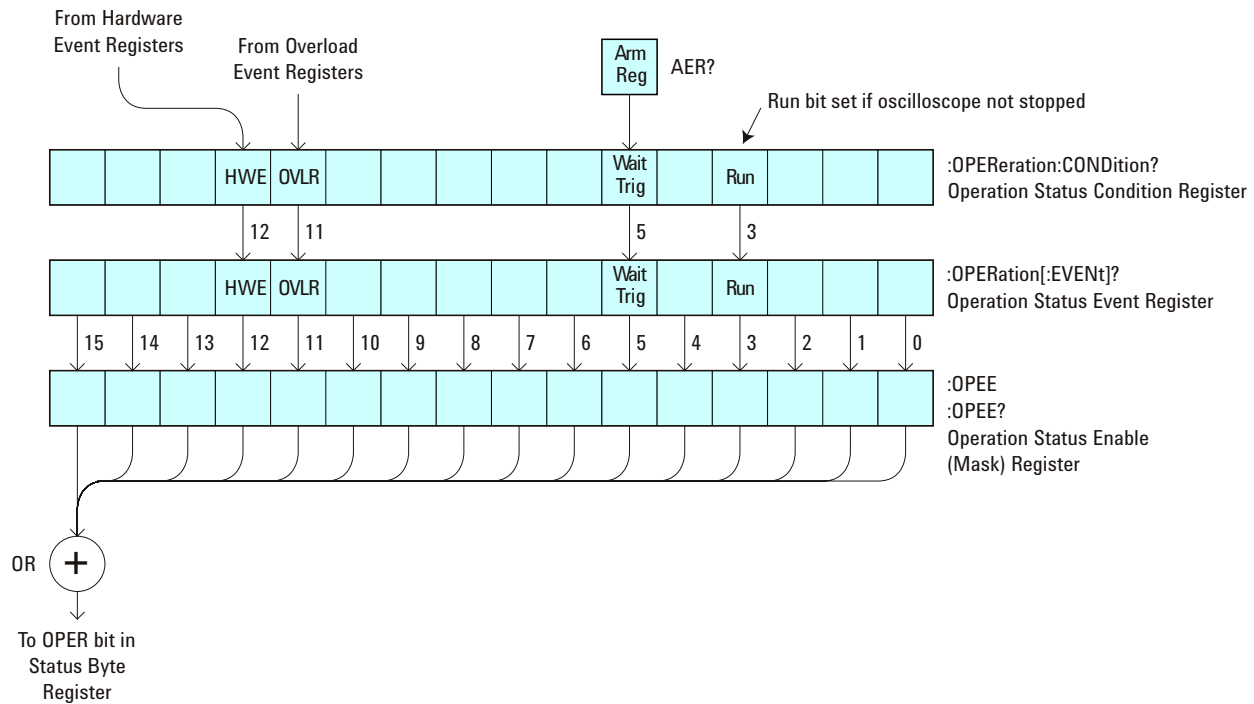


Table 45 Operation Status Enable Register (OPEE)

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 96
 - [":AER \(Arm Event Register\)"](#) on page 98
 - [":CHANnel<n>:PROTection"](#) on page 178
 - [":EXTeRnal:PROTection"](#) on page 207
 - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 118
 - [":OVLenable \(Overload Event Enable Register\)"](#) on page 120
 - [":OVLRegister \(Overload Event Register\)"](#) on page 122
 - ["*STB \(Read Status Byte\)"](#) on page 89
 - ["*SRE \(Service Request Enable\)"](#) on page 87

:OPERRegister:CONDition (Operation Status Condition Register)

C (see page 616)

Query Syntax :OPERRegister:CONDition?

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

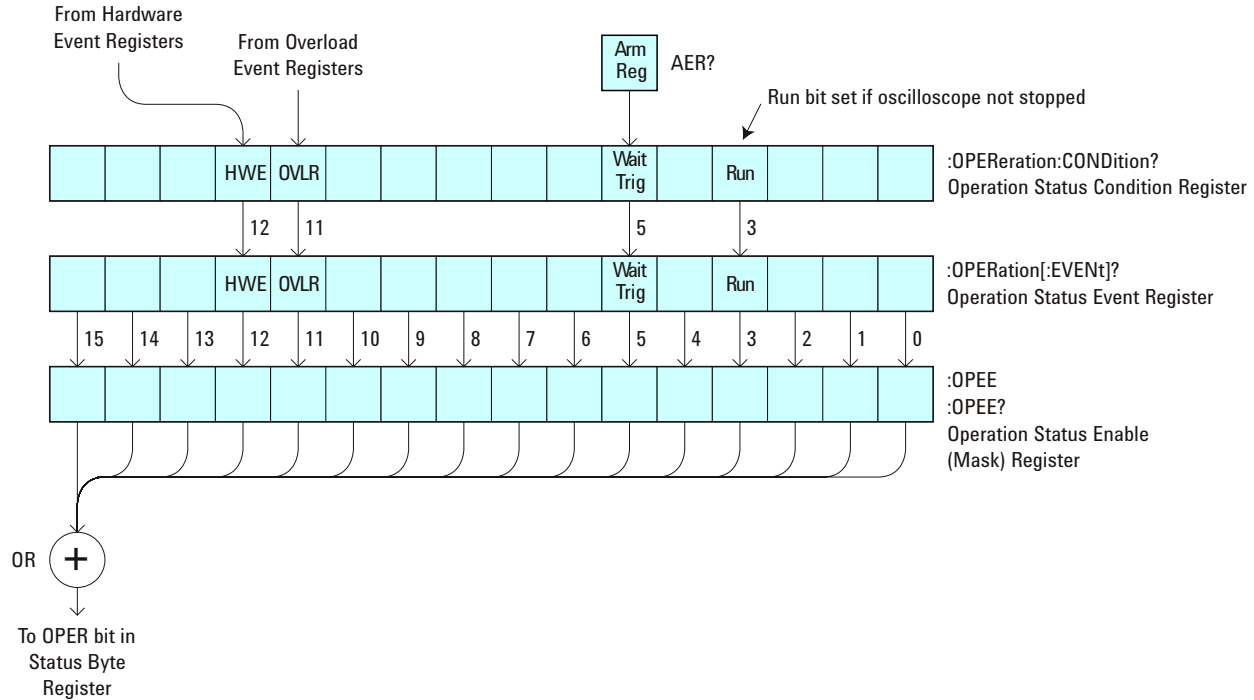


Table 46 Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred..
11	OVLRL	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 96](#)
 - [":CHANnel<n>:PROTection" on page 178](#)
 - [":EXTErnal:PROTection" on page 207](#)
 - [":OPEE \(Operation Status Enable Register\)" on page 114](#)
 - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)" on page 118](#)
 - [":OVLenable \(Overload Event Enable Register\)" on page 120](#)
 - [":OVLRegister \(Overload Event Register\)" on page 122](#)
 - ["*STB \(Read Status Byte\)" on page 89](#)
 - ["*SRE \(Service Request Enable\)" on page 87](#)
 - [":HWERegister\[:EVENT\] \(Hardware Event Event Register\)" on page 111](#)
 - [":HWEenable \(Hardware Event Enable Register\)" on page 107](#)

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

C (see page 616)

Query Syntax :OPERRegister[:EVENT]?

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

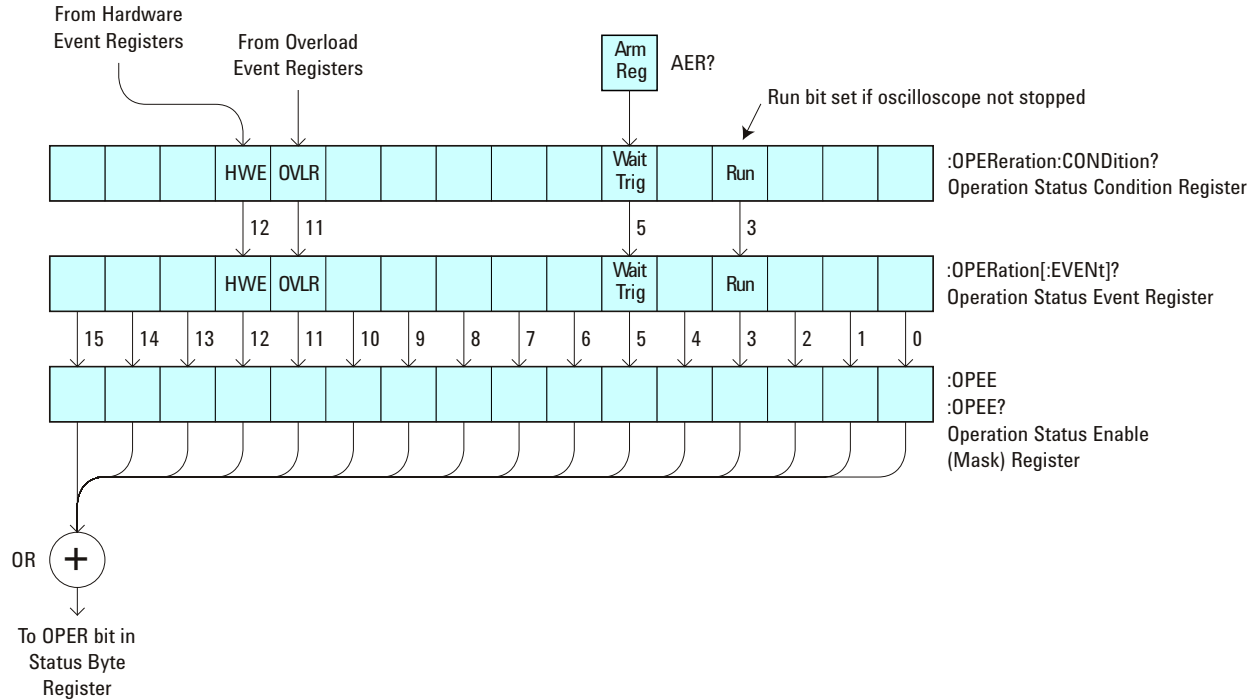


Table 47 Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred.
11	OVL	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 96](#)
 - [":CHANnel<n>:PROTection" on page 178](#)
 - [":EXTErnal:PROTection" on page 207](#)
 - [":OPEE \(Operation Status Enable Register\)" on page 114](#)
 - [":OPERegister:CONDition \(Operation Status Condition Register\)" on page 116](#)
 - [":OVLenable \(Overload Event Enable Register\)" on page 120](#)
 - [":OVLRegister \(Overload Event Register\)" on page 122](#)
 - ["*STB \(Read Status Byte\)" on page 89](#)
 - ["*SRE \(Service Request Enable\)" on page 87](#)
 - [":HWERegister\[:EVENT\] \(Hardware Event Event Register\)" on page 111](#)
 - [":HWEenable \(Hardware Event Enable Register\)" on page 107](#)

:OVLenable (Overload Event Enable Register)

C (see page 616)

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.

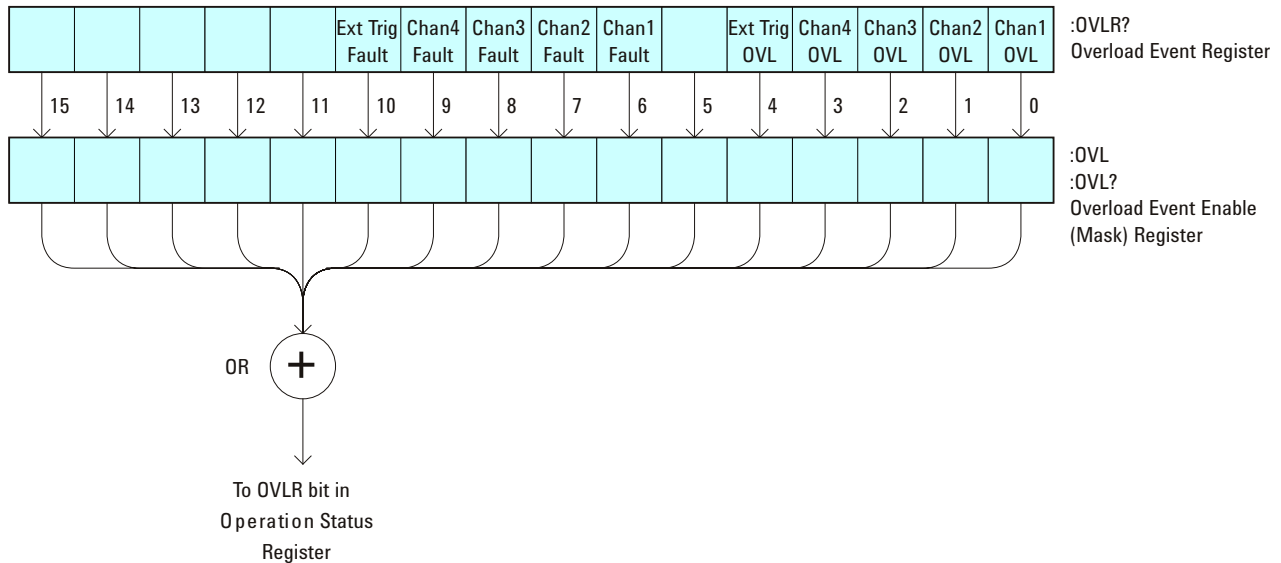


Table 48 Overload Event Enable Register (OVL)

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 96
 - [":CHANnel<n>:PROTection"](#) on page 178
 - [":EXTeRnal:PROTection"](#) on page 207
 - [":OPEE \(Operation Status Enable Register\)"](#) on page 114
 - [":OPERegister:CONDition \(Operation Status Condition Register\)"](#) on page 116
 - [":OPERegister\[:EVENT\] \(Operation Status Event Register\)"](#) on page 118
 - [":OVLRegister \(Overload Event Register\)"](#) on page 122
 - ["*STB \(Read Status Byte\)"](#) on page 89
 - ["*SRE \(Service Request Enable\)"](#) on page 87

:OVLRegister (Overload Event Register)

C (see page 616)

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.

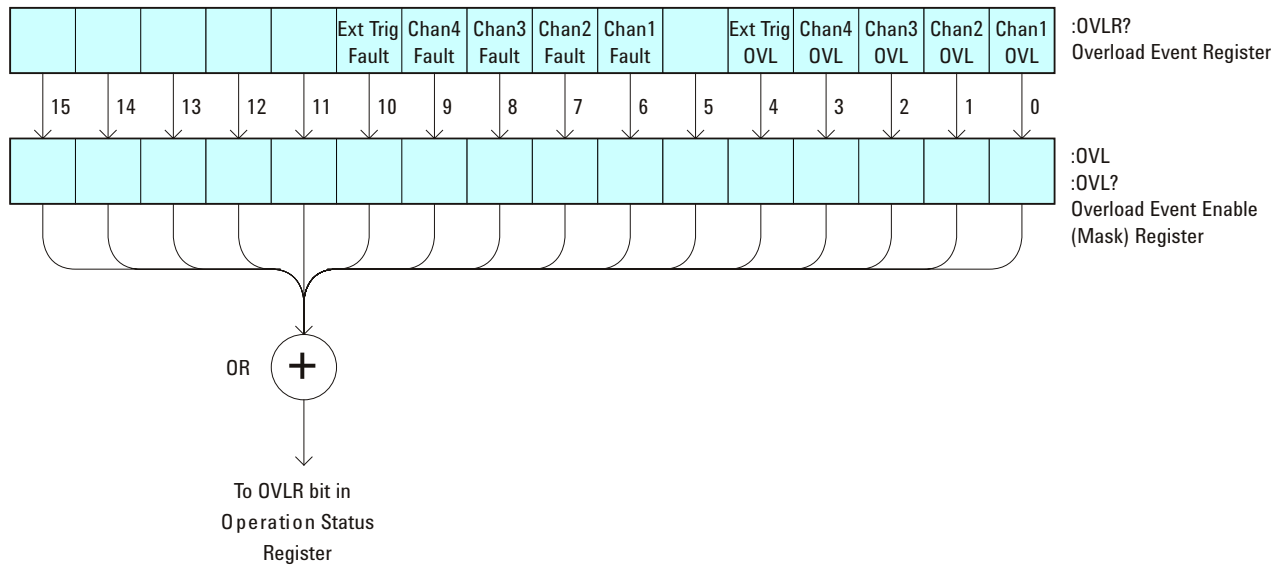


Table 49 Overload Event Register (OVL)

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 96
 - [":CHANnel<n>:PROTection"](#) on page 178
 - [":EXTErnal:PROTection"](#) on page 207
 - [":OPEE \(Operation Status Enable Register\)"](#) on page 114
 - [":OVLenable \(Overload Event Enable Register\)"](#) on page 120
 - ["*STB \(Read Status Byte\)"](#) on page 89
 - ["*SRE \(Service Request Enable\)"](#) on page 87

:PRINt

C (see [page 616](#))

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 559](#) for more information.

- See Also**
- "[Introduction to Root \(:\) Commands](#)" on [page 96](#)
 - "[Introduction to :HARDcopy Commands](#)" on [page 222](#)
 - "[:HARDcopy:FORMat](#)" on [page 559](#)
 - "[:HARDcopy:FACTors](#)" on [page 226](#)
 - "[:HARDcopy:GRAYscale](#)" on [page 560](#)
 - "[:DISPlay:DATA](#)" on [page 193](#)

:RUN

C (see [page 616](#))

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 96
 - [":SINGLE"](#) on page 127
 - [":STOP"](#) on page 129

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 6"](#) on page 692

:SERial

N (see [page 616](#))

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 96

:SINGle

C (see [page 616](#))

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 96
 - [":RUN"](#) on page 125
 - [":STOP"](#) on page 129

:STATus

N (see [page 616](#))

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 96
 - [":BLANK"](#) on page 103
 - [":CHANnel<n>:DISPlay"](#) on page 169
 - [":DIGital<n>:DISPlay"](#) on page 185
 - [":FUNCTION:DISPlay"](#) on page 213
 - [":POD<n>:DISPlay"](#) on page 288
 - [":VIEW"](#) on page 131

:STOP

C (see [page 616](#))

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 96
 - [":RUN"](#) on page 125
 - [":SINGLE"](#) on page 127

- Example Code**
- ["Example Code"](#) on page 125

:TER (Trigger Event Register)

C (see [page 616](#))

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 96
- ["*SRE \(Service Request Enable\)"](#) on page 87
- ["*STB \(Read Status Byte\)"](#) on page 89

:VIEW

N (see [page 616](#))

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 96
 - "[:BLANK](#)" on page 103
 - "[:CHANnel<n>:DISPlay](#)" on page 169
 - "[:DIGital<n>:DISPlay](#)" on page 185
 - "[:FUNction:DISPlay](#)" on page 213
 - "[:POD<n>:DISPlay](#)" on page 288
 - "[:STATus](#)" on page 128

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 6](#)" on page 692

:ACQUIRE Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQUIRE Commands" on page 132.

Table 50 :ACQUIRE Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQUIRE:AAlias? (see page 134)	{1 0}
:ACQUIRE:COMPLETE <complete> (see page 135)	:ACQUIRE:COMPLETE? (see page 135)	<complete> ::= 100; an integer in NR1 format
:ACQUIRE:COUNT <count> (see page 136)	:ACQUIRE:COUNT? (see page 136)	<count> ::= an integer from 1 to 65536 in NR1 format
:ACQUIRE:DAALIAS <mode> (see page 137)	:ACQUIRE:DAALIAS? (see page 137)	<mode> ::= {DISABLE AUTO}
:ACQUIRE:MODE <mode> (see page 138)	:ACQUIRE:MODE? (see page 138)	<mode> ::= {RTIME ETIME SEGMENTED}
n/a	:ACQUIRE:POINTS? (see page 139)	<# points> ::= an integer in NR1 format
:ACQUIRE:RSIGNAL <ref_signal_mode> (see page 140)	:ACQUIRE:RSIGNAL? (see page 140)	<ref_signal_mode> ::= {OFF OUT IN}
:ACQUIRE:SEGMENTED:COUNT <count> (see page 141)	:ACQUIRE:SEGMENTED:COUNT? (see page 141)	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
:ACQUIRE:SEGMENTED:INDEX <index> (see page 142)	:ACQUIRE:SEGMENTED:INDEX? (see page 142)	<index> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:ACQUIRE:SRATE? (see page 144)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQUIRE:TYPE <type> (see page 145)	:ACQUIRE:TYPE? (see page 145)	<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 1 (smoothing) to 65536. The `COUNT` value determines the number of averages that must be acquired.

Peak Detect

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

Real-time Mode

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

Equivalent-time Mode

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

Reporting the Setup

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

Return Format

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

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

:ACquire:AALias

N (see [page 616](#))

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 132
 - "[:ACquire:DAALias](#)" on page 137

:ACQUIRE:COMPLETE

C (see [page 616](#))

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 132
 - [":ACQUIRE:TYPE"](#) on page 145
 - [":DIGITIZE"](#) on page 105
 - [":WAVEFORM:POINTS"](#) on page 489

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 6"](#) on page 692

:ACquire:COUNT

C (see [page 616](#))

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 1 (smoothing) to 65536.

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 1 to 65536 in NR1 format

- See Also**
- ["Introduction to :ACquire Commands"](#) on page 132
 - [":ACquire:TYPE"](#) on page 145
 - [":DIGitize"](#) on page 105
 - [":WAVEform:COUNT"](#) on page 485

:ACQUIRE:DAALIAS

N (see [page 616](#))

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 132
 - [":ACQUIRE:AALIAS"](#) on page 134

:ACQUIRE:MODE

C (see [page 616](#))

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 132
 - [":ACQUIRE:TYPE"](#) on page 145

:ACQUIRE:POINTS

C (see [page 616](#))

Query Syntax :ACQUIRE:POINTS?

The :ACQUIRE:POINTS? query returns the number of data points that the hardware will acquire from the input signal. The number of points acquired is not directly controllable. To set the number of points to be transferred from the oscilloscope, use the command :WAVEFORM:POINTS. The :WAVEFORM:POINTS? query will return the number of points available to be transferred from the oscilloscope.

Return Format <points_argument><NL>

<points_argument> ::= an integer in NR1 format

- See Also**
- ["Introduction to :ACQUIRE Commands"](#) on page 132
 - [":DIGITIZE"](#) on page 105
 - [":WAVEFORM:POINTS"](#) on page 489

:ACQUIRE:RSIGNAL

N (see [page 616](#))

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 ± 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 351](#)
 - The *Agilent InfiniiVision 7000 Series Oscilloscope User's Guide* for information on using the 10 MHz reference clock.

:ACQUIRE:SEGMENTED:COUNT

N (see [page 616](#))

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 138
 - [":DIGITIZE"](#) on page 105
 - [":WAVEFORM:SEGMENTED:COUNT"](#) on page 496
 - ["Introduction to :ACQUIRE Commands"](#) on page 132

Example Code • ["Example Code"](#) on page 142

:ACQUIRE:SEGMENTED:INDEX

N (see [page 616](#))

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 138
 - [":ACQUIRE:SEGMENTED:COUNT"](#) on page 141
 - [":DIGITIZE"](#) on page 105
 - [":WAVEFORM:SEGMENTED:COUNT"](#) on page 496
 - [":WAVEFORM:SEGMENTED:TTAG"](#) on page 497
 - ["Introduction to :ACQUIRE Commands"](#) on page 132

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 616](#))

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 132
 - "[:ACQUIRE:POINTS](#)" on page 139

:ACquire:TYPE

C (see [page 616](#))

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 (smoothing) 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.

This command is functionally equivalent to :ACquire:TYPE AVERage and :ACquire:COUNT 1.

- 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 132

3 Commands by Subsystem

- [":ACQUIRE:COUNT"](#) on page 136
- [":ACQUIRE:MODE"](#) on page 138
- [":DIGITIZE"](#) on page 105
- [":WAVEFORM:TYPE"](#) on page 503
- [":WAVEFORM:PREAmble"](#) on page 493

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 6"](#)
on page 692

:BUS<n> Commands

Control all oscilloscope functions associated with buses made up of digital channels. See ["Introduction to :BUS<n> Commands"](#) on page 148.

Table 51 :BUS<n> Commands Summary

Command	Query	Options and Query Returns
:BUS<n>:BIT<m> {{0 OFF} {1 ON}} (see page 149)	:BUS<n>:BIT<m>? (see page 149)	{0 1} <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:BITS <channel_list>, {{0 OFF} {1 ON}} (see page 150)	:BUS<n>:BITS? (see page 150)	<channel_list>, {0 1} <channel_list> ::= (@<m>, <m>:<m> ...) where "," is separator and ":" is range <n> ::= 1 or 2; an integer in NR1 format <m> ::= 0-15; an integer in NR1 format
:BUS<n>:CLear (see page 152)	n/a	<n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:DISPlay {{0 OFF} {1 ON}} (see page 153)	:BUS<n>:DISPlay? (see page 153)	{0 1} <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:LABel <string> (see page 154)	:BUS<n>:LABel? (see page 154)	<string> ::= quoted ASCII string up to 16 characters <n> ::= 1 or 2; an integer in NR1 format
:BUS<n>:MASK <mask> (see page 155)	:BUS<n>:MASK? (see page 155)	<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

3 Commands by Subsystem

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 616](#))

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 148
 - [":BUS<n>:BITS"](#) on page 150
 - [":BUS<n>:CLEar"](#) on page 152
 - [":BUS<n>:DISPlay"](#) on page 153
 - [":BUS<n>:LABel"](#) on page 154
 - [":BUS<n>:MASK"](#) on page 155

Example Code

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

:BUS<n>:BITS

N (see [page 616](#))

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 148
 - [":BUS<n>:BIT<m>"](#) on page 149
 - [":BUS<n>:CLEar"](#) on page 152
 - [":BUS<n>:DISPlay"](#) on page 153
 - [":BUS<n>:LABel"](#) on page 154
 - [":BUS<n>:MASK"](#) on page 155

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 616](#))

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 148
 - [":BUS<n>:BIT<m>"](#) on page 149
 - [":BUS<n>:BITS"](#) on page 150
 - [":BUS<n>:DISPlay"](#) on page 153
 - [":BUS<n>:LABel"](#) on page 154
 - [":BUS<n>:MASK"](#) on page 155

:BUS<n>:DISPlay

N (see [page 616](#))

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 148
 - [":BUS<n>:BIT<m>"](#) on page 149
 - [":BUS<n>:BITS"](#) on page 150
 - [":BUS<n>:CLEar"](#) on page 152
 - [":BUS<n>:LABel"](#) on page 154
 - [":BUS<n>:MASK"](#) on page 155

:BUS<n>:LABel

N (see [page 616](#))

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 148
 - "[:BUS<n>:BIT<m>](#)" on page 149
 - "[:BUS<n>:BITS](#)" on page 150
 - "[:BUS<n>:CLEar](#)" on page 152
 - "[:BUS<n>:DISPlay](#)" on page 153
 - "[:BUS<n>:MASK](#)" on page 155
 - "[:CHANnel<n>:LABel](#)" on page 172
 - "[:DISPlay:LABList](#)" on page 196
 - "[:DIGital<n>:LABel](#)" on page 186

Example Code

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

:BUS<n>:MASK

N (see [page 616](#))

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 148
 - [":BUS<n>:BIT<m>"](#) on page 149
 - [":BUS<n>:BITS"](#) on page 150
 - [":BUS<n>:CLEar"](#) on page 152
 - [":BUS<n>:DISPlay"](#) on page 153
 - [":BUS<n>:LABel"](#) on page 154

:CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "[Introduction to :CALibrate Commands](#)" on page 156.

Table 52 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 157)	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string> (see page 158)	:CALibrate:LABel? (see page 158)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:START (see page 159)	n/a	n/a
n/a	:CALibrate:STATus? (see page 160)	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITCh? (see page 161)	{PROTeCted UNPRoTeCted}
n/a	:CALibrate:TEMPerature? (see page 162)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 163)	<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 616](#))

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 156

:CALibrate:LABel

N (see [page 616](#))

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 156

:CALibrate:START

N (see [page 616](#))

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 156
 - "[:CALibrate:SWITCh](#)" on page 161

:CALibrate:STATus

N (see [page 616](#))

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 156

:CALibrate:SWITCh

N (see [page 616](#))

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 156

:CALibrate:TEMPerature

N (see [page 616](#))

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 156

:CALibrate:TIME

N (see [page 616](#))

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 156

:CHANnel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See ["Introduction to :CHANnel<n> Commands"](#) on page 165.

Table 53 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {0 OFF} {1 ON}} (see page 167)	:CHANnel<n>:BWLimit? (see page 167)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUpling <coupling> (see page 168)	:CHANnel<n>:COUpling? (see page 168)	<coupling> ::= {AC DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {0 OFF} {1 ON}} (see page 169)	:CHANnel<n>:DISPlay? (see page 169)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 170)	:CHANnel<n>:IMPedance? (see page 170)	<impedance> ::= {ONEMeg FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {0 OFF} {1 ON}} (see page 171)	:CHANnel<n>:INVert? (see page 171)	{0 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see page 172)	:CHANnel<n>:LABel? (see page 172)	<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 page 173)	:CHANnel<n>:OFFSet? (see page 173)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 174)	:CHANnel<n>:PROBe? (see page 174)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 175)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 176)	:CHANnel<n>:PROBe:SKEW? (see page 176)	<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 page 177)	:CHANnel<n>:PROBe:STYPe? (see page 177)	<signal type> ::= {DIFFerential SINGLE} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see page 178)	:CHANnel<n>:PROTection? (see page 178)	{NORM TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 179)	:CHANnel<n>:RANGe? (see page 179)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALE <scale>[suffix] (see page 180)	:CHANnel<n>:SCALE? (see page 180)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see page 181)	:CHANnel<n>:UNITs? (see page 181)	<units> ::= {VOLT AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0 OFF} {1 ON}} (see page 182)	:CHANnel<n>:VERNier? (see page 182)	{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.

3 Commands by 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 616](#))

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 165

:CHANnel<n>:COUPling

C (see [page 616](#))

Command Syntax :CHANnel<n>:COUPling <coupling>

<coupling> ::= {AC | DC}

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

<n> ::= {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>

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

See Also • ["Introduction to :CHANnel<n> Commands"](#) on page 165

:CHANnel<n>:DISPlay

C (see [page 616](#))

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 165
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103
 - [":STATus"](#) on page 128
 - [":POD<n>:DISPlay"](#) on page 288
 - [":DIGital<n>:DISPlay"](#) on page 185

:CHANnel<n>:IMPedance

C (see [page 616](#))

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 Ω) and FIFTy (50 Ω).

NOTE

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

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 165

:CHANnel<n>:INVert

N (see [page 616](#))

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 165

:CHANnel<n>:LABel

N (see [page 616](#))

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 165
 - [":DISPlay:LABel"](#) on page 195
 - [":DIGital<n>:LABel"](#) on page 186
 - [":DISPlay:LABList"](#) on page 196
 - [":BUS<n>:LABel"](#) on page 154

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 6"](#) on page 692

:CHANnel<n>:OFFSet

C (see [page 616](#))

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 165
 - "[:CHANnel<n>:RANGe](#)" on page 179
 - "[:CHANnel<n>:SCALE](#)" on page 180
 - "[:CHANnel<n>:PROBe](#)" on page 174

:CHANnel<n>:PROBe

C (see [page 616](#))

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 165
 - [":CHANnel<n>:RANGe"](#) on page 179
 - [":CHANnel<n>:SCALE"](#) on page 180
 - [":CHANnel<n>:OFFSet"](#) on page 173

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 6"](#) on page 692

:CHANnel<n>:PROBe:ID

C (see [page 616](#))

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 165

:CHANnel<n>:PROBe:SKEW

C (see [page 616](#))

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 165

:CHANnel<n>:PROBe:STYPe

C (see [page 616](#))

Command Syntax**NOTE**

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

```
:CHANnel<n>:PROBe:STYPe <signal type>
<signal type> ::= {DIFFerential | SINGle}
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

When single-ended is selected, the :CHANnel<n>:OFFset command changes the offset value of the probe amplifier. When differential is selected, the :CHANnel<n>:OFFset command changes the offset value of the channel amplifier.

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

The :CHANnel<n>:PROBe:STYPe? query returns the current probe signal type setting for the selected channel.

Return Format <signal type><NL>

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

- See Also**
- "[Introduction to :CHANnel<n> Commands](#)" on page 165
 - "[:CHANnel<n>:OFFSet](#)" on page 173

:CHANnel<n>:PROTection

N (see [page 616](#))

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 170) 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 165
 - [":CHANnel<n>:COUPling"](#) on page 168
 - [":CHANnel<n>:IMPedance"](#) on page 170
 - [":CHANnel<n>:PROBe"](#) on page 174

:CHANnel<n>:RANGe

C (see [page 616](#))

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 165
 - [":CHANnel<n>:SCALE"](#) on page 180
 - [":CHANnel<n>:PROBe"](#) on page 174

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 6"](#) on page 692

:CHANnel<n>:SCALE

N (see [page 616](#))

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 165
 - [":CHANnel<n>:RANGE"](#) on page 179
 - [":CHANnel<n>:PROBE"](#) on page 174

:CHANnel<n>:UNITs

N (see [page 616](#))

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 165
 - [":CHANnel<n>:RANGe"](#) on page 179
 - [":CHANnel<n>:PROBe"](#) on page 174
 - [":EXTeRnal:UNITs"](#) on page 209

:CHANnel<n>:VERNier

N (see [page 616](#))

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 165

:DIGital<n> Commands

Control all oscilloscope functions associated with individual digital channels. See "[Introduction to :DIGital<n> Commands](#)" on page 183.

Table 54 :DIGital<n> Commands Summary

Command	Query	Options and Query Returns
:DIGital<n>:DISPlay {0 OFF} {1 ON} (see page 185)	:DIGital<n>:DISPlay? (see page 185)	{0 1} <n> ::= 0-15; an integer in NR1 format
:DIGital<n>:LABel <string> (see page 186)	:DIGital<n>:LABel? (see page 186)	<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 page 187)	:DIGital<n>:POSition? (see page 187)	<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 page 188)	:DIGital<n>:SIZE? (see page 188)	<value> ::= {SMALl MEDium LARGe}
:DIGital<n>:THReshold <value>[suffix] (see page 189)	:DIGital<n>:THReshold? (see page 189)	<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.

3 Commands by 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 616](#))

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 183
 - [":POD<n>:DISPlay"](#) on page 288
 - [":CHANnel<n>:DISPlay"](#) on page 169
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103
 - [":STATus"](#) on page 128

:DIGital<n>:LABel

N (see [page 616](#))

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 183
 - [":CHANnel<n>:LABel"](#) on page 172
 - [":DISPlay:LABList"](#) on page 196
 - [":BUS<n>:LABel"](#) on page 154

:DIGital<n>:POSition

N (see [page 616](#))

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 183

:DIGital<n>:SIZE

N (see [page 616](#))

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 183
 - "[:POD<n>:SIZE](#)" on page 289
 - "[:DIGital<n>:POSition](#)" on page 187

:DIGital<n>:THReshold

N (see [page 616](#))

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 183
 - ":POD<n>:THReshold" on page 290
 - ":TRIGger[:EDGE]:LEVel" on page 392

:DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See ["Introduction to :DISPlay Commands"](#) on page 190.

Table 55 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLear (see page 192)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>] [,][<palette>]<displa y data> (see page 193)	:DISPlay:DATA? [<format>][,][<area>] [,][<palette>] (see page 193)	<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 195)	:DISPlay:LABel? (see page 195)	{0 1}
:DISPlay:LABList <binary block> (see page 196)	:DISPlay:LABList? (see page 196)	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters
:DISPlay:PERsistence <value> (see page 197)	:DISPlay:PERsistence? (see page 197)	<value> ::= {MINimum INFinite}}
:DISPlay:SOURce <value> (see page 198)	:DISPlay:SOURce? (see page 198)	<value> ::= {PMEMory{0 1 2 3 4 5 6 7 8 9}}
:DISPlay:VECTors {{1 ON} {0 OFF}} (see page 199)	:DISPlay:VECTors? (see page 199)	{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 616](#))

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 190
 - [":CDISplay"](#) on page 104

:DISPlay:DATA

N (see page 616)

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 190
 - [":DISPlay:SOURce"](#) on page 198
 - [":HARDcopy:INKSaver"](#) on page 228
 - [":MERGe"](#) on page 113
 - [":PRINT"](#) on page 124
 - ["*RCL \(Recall\)"](#) on page 82
 - ["*SAV \(Save\)"](#) on page 86
 - [":VIEW"](#) on page 131

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 6"](#) on page 692

:DISPlay:LABel

N (see [page 616](#))

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 190
 - [":CHANnel<n>:LABel"](#) on page 172

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 6"](#) on page 692

:DISPlay:LABList

N (see [page 616](#))

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 190
 - "[:DISPlay:LABel](#)" on page 195
 - "[:CHANnel<n>:LABel](#)" on page 172
 - "[:DIGital<n>:LABel](#)" on page 186

:DISPlay:PERStence

N (see [page 616](#))

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 190
 - "[:DISPlay:CLEar](#)" on page 192
 - "[:CDISplay](#)" on page 104

:DISPlay:SOURce

N (see [page 616](#))

Command Syntax :DISPlay:SOURce <value>

```
<value> ::= {PMEemory0 | PMEemory1 | PMEemory2 | PMEemory3 | PMEemory4  
            | PMEemory5 | PMEemory6 | PMEemory7 | PMEemory8 | PMEemory9}
```

PMEemory0-9 ::= pixel memory 0 through 9

The :DISPlay:SOURce command specifies the default source and destination for the :DISPlay:DATA command and query. PMEemory0-9 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 190
 - [":DISPlay:DATA"](#) on page 193

:DISPlay:VECTors

N (see [page 616](#))

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 190

:EXternal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXternal Trigger Commands" on page 200.

Table 56 :EXternal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXternal:BWLimit <bwlimit> (see page 202)	:EXternal:BWLimit? (see page 202)	<bwlimit> ::= {0 OFF}
:EXternal:IMPedance <value> (see page 203)	:EXternal:IMPedance? (see page 203)	<impedance> ::= {ONEMeg FIFTy}
:EXternal:PROBe <attenuation> (see page 204)	:EXternal:PROBe? (see page 204)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXternal:PROBe:ID? (see page 205)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXternal:PROBe:STYPe <signal type> (see page 206)	:EXternal:PROBe:STYPe? (see page 206)	<signal type> ::= {DIFFerential SINGle}
:EXternal:PROTection[:CLEar] (see page 207)	:EXternal:PROTection? (see page 207)	{NORM TRIP}
:EXternal:RANGe <range>[<suffix>] (see page 208)	:EXternal:RANGe? (see page 208)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V mV}
:EXternal:UNITs <units> (see page 209)	:EXternal:UNITs? (see page 209)	<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 616](#))

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 200
 - ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:HFReject"](#) on page 362

:EXternal:IMPedance

C (see [page 616](#))

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 Ω) and FIFTy (50 Ω).

NOTE

You can set external trigger input impedance to FIFTy (50 Ω) 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 200
 - ["Introduction to :TRIGger Commands"](#) on page 358
 - [":CHANnel<n>:IMPedance"](#) on page 170

:EXtErnal:PROBe

C (see [page 616](#))

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 200
 - [":EXtErnal:RANGe"](#) on page 208
 - ["Introduction to :TRIGger Commands"](#) on page 358
 - [":CHANnel<n>:PROBe"](#) on page 174

:EXternal:PROBe:ID

C (see [page 616](#))

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 200

:EXtErnal:PROBe:STYPe

C (see [page 616](#))

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 200

:EXternal:PROtection

N (see [page 616](#))

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 203) 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 200
 - "[:EXternal:IMPedance](#)" on page 203
 - "[:EXternal:PROBE](#)" on page 204

:EXternal:RANGe

C (see [page 616](#))

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 200
 - [":EXternal:PROBe"](#) on page 204
 - ["Introduction to :TRIGger Commands"](#) on page 358

:EXternal:UNITs

N (see [page 616](#))

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 200
 - ["Introduction to :TRIGger Commands"](#) on page 358
 - [":EXternal:RANGe"](#) on page 208
 - [":EXternal:PROBe"](#) on page 204
 - [":CHANnel<n>:UNITs"](#) on page 181

:FUNCTION Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCTION Commands" on page 211.

Table 57 :FUNCTION Commands Summary

Command	Query	Options and Query Returns
:FUNCTION:CENter <frequency> (see page 212)	:FUNCTION:CENter? (see page 212)	<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 213)	:FUNCTION:DISPlay? (see page 213)	{0 1}
:FUNCTION:OFFSet <offset> (see page 214)	:FUNCTION:OFFSet? (see page 214)	<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 215)	:FUNCTION:OPERation? (see page 215)	<operation> ::= {SUBtract MULTiPLY INTegrate DIFFerentiate FFT SQRT}
:FUNCTION:RANGe <range> (see page 216)	:FUNCTION:RANGe? (see page 216)	<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 217)	:FUNCTION:REFerence? (see page 217)	<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 218)	:FUNCTION:SCALe? (see page 218)	<scale value> ::= integer in NR1 format <suffix> ::= {V dB}

Table 57 :FUNcTion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNcTion:SOURce <source> (see page 219)	:FUNcTion:SOURce? (see page 219)	<source> ::= {CHANnel<n> ADD SUBT MULT} <n> ::= 1-2 or 1-4 in NR1 format
:FUNcTion:SPAN (see page 220)	:FUNcTion:SPAN? (see page 220)	 ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNcTion:WINDow <window> (see page 221)	:FUNcTion:WINDow? (see page 221)	<window> ::= {RECTangular HANNing FLATtop}

Introduction to :FUNcTion Commands

The FUNcTion subsystem controls the math functions in the oscilloscope. Multiply (channel 1 x channel 2), subtract (channel 1 - channel 2), differentiate, integrate, and FFT (Fast Fourier Transform) operations are available. These math operations only use the analog (vertical) channels.

NOTE

To perform analog channel addition, set analog channel 2 to invert and select subtract (channel 1 - channel 2).

The SOURce, 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 SUBT;DISP 0;RANG +8.00E+00;OFFS +0.00000E+00
```

:FUNCTION:CENTer

N (see [page 616](#))

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 211
 - "[:FUNCTION:SPAN](#)" on page 220
 - "[:TIMEbase:RANGe](#)" on page 350
 - "[:TIMEbase:SCALE](#)" on page 353

:FUNCTION:DISPlay

N (see [page 616](#))

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 211
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103
 - [":STATus"](#) on page 128

:FUNCTION:OFFSet

N (see [page 616](#))

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 211
 - "[:FUNCTION:RANGE](#)" on page 216
 - "[:FUNCTION:REFerence](#)" on page 217
 - "[:FUNCTION:SCALE](#)" on page 218

:FUNCTION:OPERation

N (see [page 616](#))

Command Syntax :FUNCTION:OPERation <operation>

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

The :FUNCTION:OPERation command sets the desired operation for a function. (FFT = Fast Fourier Transform, SQRT = square root.)

Query Syntax :FUNCTION:OPERation?

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

Return Format <operation><NL>

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

See Also • ["Introduction to :FUNCTION Commands"](#) on page 211

:FUNCTION:RANGe

N (see [page 616](#))

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 211
 - ":FUNCTION:SCALE" on page 218

:FUNCTION:REFERENCE

N (see [page 616](#))

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 211
 - [":FUNCTION:OFFSET"](#) on page 214
 - [":FUNCTION:RANGE"](#) on page 216
 - [":FUNCTION:SCALE"](#) on page 218

:FUNCTION:SCALE

N (see [page 616](#))

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 211
 - [":FUNCTION:RANGE"](#) on page 216

:FUNCTION:SOURce

N (see [page 616](#))

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.

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 211
 - [":FUNCTION:OPERation"](#) on page 215

:FUNCTION:SPAN

N (see [page 616](#))

Command Syntax :FUNCTION: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 <NL>

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

- See Also**
- ["Introduction to :FUNCTION Commands"](#) on page 211
 - [":FUNCTION:CENTer"](#) on page 212
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:SCALE"](#) on page 353

:FUNCTION:WINDOW

N (see [page 616](#))

Command Syntax :FUNCTION:WINDOW <window>

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

- The RECTangular window is useful for transient signals, and signals where there are an integral number of cycles in the time record.
- The HANNing window is 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.
- The FLATtop window is best for making accurate amplitude measurements of frequency peaks.

The :FUNCTION:WINDOW command allows the selection of three 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.

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}

See Also • ["Introduction to :FUNCTION Commands"](#) on page 211

:HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "[Introduction to :HARDcopy Commands](#)" on page 222.

Table 58 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 224)	:HARDcopy:AREA? (see page 224)	<area> ::= SCREEN
:HARDcopy:APRinter <active_printer> (see page 225)	:HARDcopy:APRinter? (see page 225)	<active_printer> ::= {<index> <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 OFF} {1 ON}} (see page 226)	:HARDcopy:FACTors? (see page 226)	{0 1}
:HARDcopy:FFEed {{0 OFF} {1 ON}} (see page 227)	:HARDcopy:FFEed? (see page 227)	{0 1}
:HARDcopy:INKSaver {{0 OFF} {1 ON}} (see page 228)	:HARDcopy:INKSaver? (see page 228)	{0 1}
:HARDcopy:PALette <palette> (see page 229)	:HARDcopy:PALette? (see page 229)	<palette> ::= {COLor GRAYscale NONE}
n/a	:HARDcopy:PRinter:LIS T? (see page 230)	<list> ::= [<printer_spec>] ... [printer_spec] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y N} <name> ::= name of printer
:HARDcopy:STARt (see page 231)	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 616](#))

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 222
 - [":HARDcopy:START"](#) on page 231
 - [":HARDcopy:APRinter"](#) on page 225
 - [":HARDcopy:PRinter:LIST"](#) on page 230
 - [":HARDcopy:FACTors"](#) on page 226
 - [":HARDcopy:FFEed"](#) on page 227
 - [":HARDcopy:INKSaver"](#) on page 228
 - [":HARDcopy:PALETTE"](#) on page 229

:HARDcopy:APRinter

N (see [page 616](#))

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 222
 - [":HARDcopy:PRinter:LIST"](#) on page 230
 - [":HARDcopy:START"](#) on page 231

:HARDcopy:FACTors

N (see [page 616](#))

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 222
 - [":HARDcopy:START"](#) on page 231
 - [":HARDcopy:FFEed"](#) on page 227
 - [":HARDcopy:INKSaver"](#) on page 228
 - [":HARDcopy:PALETTE"](#) on page 229

:HARDcopy:FFEed

N (see [page 616](#))

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 222
 - "[:HARDcopy:START](#)" on page 231
 - "[:HARDcopy:FACTors](#)" on page 226
 - "[:HARDcopy:INKSaver](#)" on page 228
 - "[:HARDcopy:PALETTE](#)" on page 229

:HARDcopy:INKSaver

N (see [page 616](#))

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 222
 - [":HARDcopy:START"](#) on page 231
 - [":HARDcopy:FACTors"](#) on page 226
 - [":HARDcopy:FFEed"](#) on page 227
 - [":HARDcopy:PALETTE"](#) on page 229

:HARDcopy:PALETTE

N (see [page 616](#))

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 222
 - [":HARDcopy:START"](#) on page 231
 - [":HARDcopy:FACTors"](#) on page 226
 - [":HARDcopy:FFEd"](#) on page 227
 - [":HARDcopy:INKSaver"](#) on page 228

:HARDcopy:PRinter:LIST

N (see [page 616](#))

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 222
 - [":HARDcopy:APRinter"](#) on page 225
 - [":HARDcopy:START"](#) on page 231

:HARDcopy:START

N (see [page 616](#))

Command Syntax :HARDcopy:START

The :HARDcopy:START command starts a print job.

- See Also**
- "[Introduction to :HARDcopy Commands](#)" on page 222
 - "[:HARDcopy:APRinter](#)" on page 225
 - "[:HARDcopy:PRinter:LIST](#)" on page 230
 - "[:HARDcopy:FACTors](#)" on page 226
 - "[:HARDcopy:FFEed](#)" on page 227
 - "[:HARDcopy:INKSaver](#)" on page 228
 - "[:HARDcopy:PALETTE](#)" on page 229

: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 233.

Table 59 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 234)	:MARKer:MODE? (see page 234)	<mode> ::= {OFF MEASurement MANual}
:MARKer:X1Position <position>[suffix] (see page 235)	:MARKer:X1Position? (see page 235)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 236)	:MARKer:X1Y1source? (see page 236)	<source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 237)	:MARKer:X2Position? (see page 237)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s ms us ns ps Hz kHz MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 238)	:MARKer:X2Y2source? (see page 238)	<source> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 239)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see page 240)	:MARKer:Y1Position? (see page 240)	<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 page 241)	:MARKer:Y2Position? (see page 241)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V mV dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 242)	<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 616](#))

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 233
 - "[:MARKer:X1Y1source](#)" on page 236
 - "[:MARKer:X2Y2source](#)" on page 238
 - "[:MEASure:SOURce](#)" on page 270

:MARKer:X1Position

N (see [page 616](#))

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X2Position"](#) on page 237
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MEASure:TSTArt"](#) on page 569

:MARKer:X1Y1source

N (see [page 616](#))

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 234).

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MEASure:SOURce"](#) on page 270

:MARKer:X2Position

N (see [page 616](#))

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Position"](#) on page 235
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MEASure:TSTOp"](#) on page 570

:MARKer:X2Y2source

N (see [page 616](#))

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 234).

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MEASure:SOURce"](#) on page 270

:MARKer:XDELta

N (see [page 616](#))

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Position"](#) on page 235
 - [":MARKer:X2Position"](#) on page 237
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MARKer:X2Y2source"](#) on page 238

:MARKer:Y1Position

N (see [page 616](#))

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MARKer:Y2Position"](#) on page 241
 - [":MEASure:VSTArt"](#) on page 575

:MARKer:Y2Position

N (see [page 616](#))

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 233
 - [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MARKer:Y1Position"](#) on page 240
 - [":MEASure:VSTOp"](#) on page 576

:MARKer:YDELta

N (see [page 616](#))

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 233](#)
 - "[:MARKer:MODE](#)" on [page 234](#)
 - "[:MARKer:X1Y1source](#)" on [page 236](#)
 - "[:MARKer:X2Y2source](#)" on [page 238](#)
 - "[:MARKer:Y1Position](#)" on [page 240](#)
 - "[:MARKer:Y2Position](#)" on [page 241](#)

:MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 248.

Table 60 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 250)	n/a	n/a
:MEASure:COUNter [<source>] (see page 251)	:MEASure:COUNter? [<source>] (see page 251)	<source> ::= {CHANnel<n>} for DSO models <source> ::= {CHANnel<n> DIGital0,..,DIGital15 } 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 page 252)	:MEASure:DEFine? DELay (see page 253)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ -} <occurrence> ::= integer
:MEASure:DEFine THResholds, <threshold spec> (see page 252)	:MEASure:DEFine? THResholds (see page 253)	<threshold spec> ::= {STANdard} {<threshold mode>,<upper>,<middle>,<lower>} <threshold mode> ::= {PERCent ABSolute}
:MEASure:DELay [<source1>] [,<source2>] (see page 255)	:MEASure:DELay? [<source1>] [,<source2>] (see page 255)	<source1,2> ::= {CHANnel<n> FUNCTION MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 257)	:MEASure:DUTYcycle? [<source>] (see page 257)	<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

3 Commands by Subsystem

Table 60 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 258)	:MEASure:FALLtime? [<source>] (see page 258)	<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 259)	:MEASure:FREQuency? [<source>] (see page 259)	<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 260)	:MEASure:NWIDth? [<source>] (see page 260)	<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 261)	:MEASure:OVERshoot? [<source>] (see page 261)	<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 263)	:MEASure:PERiod? [<source>] (see page 263)	<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 264)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 264)	<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 265)	:MEASure:PREShoOt? [<source>] (see page 265)	<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 266)	:MEASure:PWIDth? [<source>] (see page 266)	<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 267)	:MEASure:RISEtime? [<source>] (see page 267)	<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 268)	:MEASure:SDEVIation? [<source>] (see page 268)	<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 269)	:MEASure:SHOW? (see page 269)	{1}
:MEASure:SOURce [<source1>] [,<source2>] (see page 270)	:MEASure:SOURce? (see page 270)	<source1,2> ::= {CHANnel<n> FUNctIon MATH} for DSO models <source1,2> ::= {CHANnel<n> DIGital0,...,DIGital15 FUNctIon MATH} for MSO models <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= {<source> NONE}

3 Commands by Subsystem

Table 60 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TEDGE? <slope><occurrence>[, <source>] (see page 272)	<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 page 274)	<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 276)	:MEASure:VAMPLitude? [<source>] (see page 276)	<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 277)	:MEASure:VAverage? [<source>] (see page 277)	<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 278)	:MEASure:VBASe? [<source>] (see page 278)	<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 279)	:MEASure:VMAX? [<source>] (see page 279)	<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 280)	:MEASure:VMIN? [<source>] (see page 280)	<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 281)	:MEASure:VPP? [<source>] (see page 281)	<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:VRMS [<source>] (see page 282)	:MEASure:VRMS? [<source>] (see page 282)	<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 283)	<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

3 Commands by Subsystem

Table 60 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VTOP [<source>] (see page 284)	:MEASure:VTOP? [<source>] (see page 284)	<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 285)	:MEASure:XMAX? [<source>] (see page 285)	<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 286)	:MEASure:XMIN? [<source>] (see page 286)	<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 616](#))

Command Syntax :MEASure:CLEar

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

See Also • ["Introduction to :MEASure Commands"](#) on page 248

:MEASure:COUNter

N (see [page 616](#))

Command Syntax :MEASure:COUNter [<source>]
 <source> ::= {<digital channels> | CHANnel<n>}
 <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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:FREQuency"](#) on page 259
 - [":MEASure:CLEar"](#) on page 250

:MEASure:DEFine

N (see page 616)

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 Command Syntax

<delay_spec> ::= <edge_spec1>,<edge_spec2>

<edge_spec1> ::= [<slope>]<occurrence>

<edge_spec2> ::= [<slope>]<occurrence>

<slope> ::= {+ | -}

<occurrence> ::= integer

This command defines the behavior of the :MEASure:DElay? query by specifying the start and stop edge to be used. <edge_spec1> specifies the slope and edge number on source1. <edge_spec2> specifies the slope and edge number on source2. The measurement is taken as:

$$\text{delay} = t(\text{<edge_spec2>}) - t(\text{<edge_spec1>})$$

NOTE

The :MEASure:DElay command and the front-panel delay measurement use an auto-edge selection method to determine the actual edge used for the measurement. The :MEASure:DEfine command has no effect on these delay measurements. The edges specified by the :MEASure:DEfine command only define the edges used by the :MEASure:DElay? query.

**:MEASure:DEfine
THResholds
Command Syntax**

```
:MEASure:DEfine THResholds,<threshold spec>
```

```
<threshold spec> ::= {STANdard  
| {<threshold mode>,<upper>,<middle>,<lower>}}
```

```
<threshold mode> ::= {PERCent | ABSolute}
```

for <threshold mode> = PERCent:

```
<upper>,<middle>,<lower> ::= A number specifying the upper, middle,  
and lower threshold percentage values  
between Vbase and Vtop in NR3 format.
```

for <threshold mode> = ABSolute:

```
<upper>,<middle>,<lower> ::= A number specifying the upper, middle,  
and lower threshold absolute values in  
NR3 format.
```

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.
- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGe or ":CHANnel<n>:SCALe" on page 180:CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

Query Syntax

```
:MEASure:DEfine? <meas_spec>
```

```
<meas_spec> ::= {DElay | THResholds}
```

The :MEASure:DEfine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

Return Format for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>
```

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and lower threshold percentage values between Vbase and Vtop in NR3 format.

for <meas_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>
```

<upper>, <middle>, <lower> ::= A number specifying the upper, middle, and lower threshold voltages in NR3 format.

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:DELay"](#) on page 255
 - [":MEASure:SOURce"](#) on page 270
 - [":CHANnel<n>:RANGe"](#) on page 179
 - [":CHANnel<n>:SCALE"](#) on page 180
 - [":CHANnel<n>:PROBE"](#) on page 174
 - [":CHANnel<n>:UNITs"](#) on page 181

:MEASure:DElay

N (see page 616)

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

3 Commands by Subsystem

Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THResholds command.

Return Format <value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:DEFine"](#) on page 252
 - [":MEASure:PHASe"](#) on page 264

:MEASure:DUTYcycle

C (see [page 616](#))

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 248
 - [":MEASure:PERiod"](#) on page 263
 - [":MEASure:PWIDth"](#) on page 266
 - [":MEASure:SOURce"](#) on page 270

Example Code

- ["Example Code"](#) on page 270

:MEASure:FALLtime

C (see [page 616](#))

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 248
 - "[:MEASure:RISetime](#)" on page 267
 - "[:MEASure:SOURce](#)" on page 270

:MEASure:FREQuency

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:PERiod"](#) on page 263

Example Code • ["Example Code"](#) on page 270

:MEASure:NWIDth

C (see [page 616](#))

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:

$$\text{width} = (\text{time at trailing rising edge} - \text{time at leading falling edge})$$

Return Format <value><NL>
<value> ::= negative pulse width in seconds in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:PWIDth"](#) on page 266
 - [":MEASure:PERiod"](#) on page 263

:MEASure:OVERshoot

C (see page 616)

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 248
 - ":MEASure:PREShoot" on page 265
 - ":MEASure:SOURce" on page 270
 - ":MEASure:VMAX" on page 279

3 Commands by Subsystem

- `":MEASure:VTOP"` on page 284
- `":MEASure:VBASe"` on page 278
- `":MEASure:VMIN"` on page 280

:MEASure:PERiod

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:NWIDTH"](#) on page 260
 - [":MEASure:PWIDth"](#) on page 266
 - [":MEASure:FREQuency"](#) on page 259

Example Code • ["Example Code"](#) on page 270

:MEASure:PHASe

N (see [page 616](#))

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 248
 - [":MEASure:DELAy"](#) on page 255
 - [":MEASure:PERiod"](#) on page 263
 - [":MEASure:SOURce"](#) on page 270

:MEASure:PREShoot

C (see page 616)

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 248
 - ":MEASure:SOURce" on page 270
 - ":MEASure:VMIN" on page 280
 - ":MEASure:VMAX" on page 279
 - ":MEASure:VTOP" on page 284
 - ":MEASure:VBASe" on page 278

:MEASure:PWIDth

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:NWIDth"](#) on page 260
 - [":MEASure:PERiod"](#) on page 263

:MEASure:RISetime

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:FALLtime"](#) on page 258

:MEASure:SDEVIation

N (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270

:MEASure:SHOW

N (see [page 616](#))

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 248

:MEASure:SOURce

C (see [page 616](#))

Command Syntax :MEASure:SOURce <source1>[,<source2>]
 <source1>,<source2> ::= {<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: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.

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 | NONE}

- See Also:**
- ["Introduction to :MEASure Commands" on page 248](#)
 - [":MARKer:MODE" on page 234](#)
 - [":MARKer:X1Y1source" on page 236](#)
 - [":MARKer:X2Y2source" on page 238](#)
 - [":MEASure:DELay" on page 255](#)
 - [":MEASure:PHASe" on page 264](#)

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 6"](#)
on page 692

:MEASure:TEDGe

N (see [page 616](#))

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 273](#).

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 6"](#) on page 692

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:TVALue"](#) on page 274
 - [":MEASure:VTIME"](#) on page 283

:MEASure:TVALue

C (see page 616)

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 248
 - [":MEASure:TEDGE"](#) on page 272
 - [":MEASure:VTIME"](#) on page 283

:MEASure:VAMPlitude

C (see [page 616](#))

Command Syntax :MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax :MEASure:VAMPlitude? [<source>]

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

$$\text{vertical amplitude} = V_{\text{top}} - V_{\text{base}}$$

Return Format <value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:VBASe"](#) on page 278
 - [":MEASure:VTOP"](#) on page 284
 - [":MEASure:VPP"](#) on page 281

:MEASure:VAverage

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270

:MEASure:VBASe

C (see [page 616](#))

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 248
 - "[:MEASure:SOURce](#)" on page 270
 - "[:MEASure:VTOP](#)" on page 284
 - "[:MEASure:VAMPLitude](#)" on page 276
 - "[:MEASure:VMIN](#)" on page 280

:MEASure:VMAX

C (see [page 616](#))

Command Syntax :MEASure:VMAX [<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: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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:VMIN"](#) on page 280
 - [":MEASure:VPP"](#) on page 281
 - [":MEASure:VTOP"](#) on page 284

:MEASure:VMIN

C (see [page 616](#))

Command Syntax :MEASure:VMIN [<source>]

<source> ::= {CHANnel<n> | FUNCTION | MATH}

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

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

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax :MEASure:VMIN? [<source>]

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

Return Format <value><NL>

<value> ::= minimum vertical value of the selected waveform in NR3 format

- See Also**
- ["Introduction to :MEASure Commands"](#) on page 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:VBASe"](#) on page 278
 - [":MEASure:VMAX"](#) on page 279
 - [":MEASure:VPP"](#) on page 281

:MEASure:VPP

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:VMAX"](#) on page 279
 - [":MEASure:VMIN"](#) on page 280
 - [":MEASure:VAMPLitude"](#) on page 276

:MEASure:VRMS

C (see [page 616](#))

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 248
 - "[:MEASure:SOURce](#)" on page 270

:MEASure:VTIME

N (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:TEDGe"](#) on page 272
 - [":MEASure:TVALue"](#) on page 274

:MEASure:VTOP

C (see [page 616](#))

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 248
 - [":MEASure:SOURce"](#) on page 270
 - [":MEASure:VMAX"](#) on page 279
 - [":MEASure:VAMPLitude"](#) on page 276
 - [":MEASure:VBASe"](#) on page 278

:MEASure:XMAX

N (see [page 616](#))

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 248
 - [":MEASure:XMIN"](#) on page 286
 - [":MEASure:TMAX"](#) on page 567

:MEASure:XMIn

N (see [page 616](#))

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 248](#)
 - "[:MEASure:XMAX](#)" on [page 285](#)
 - "[:MEASure:TMin](#)" on [page 568](#)

:POD Commands

Control all oscilloscope functions associated with groups of digital channels. See "[Introduction to :POD<n> Commands](#)" on page 287.

Table 61 :POD<n> Commands Summary

Command	Query	Options and Query Returns
:POD<n>:DISPlay {{0 OFF} {1 ON}} (see page 288)	:POD<n>:DISPlay? (see page 288)	{0 1} <n> ::= 1-2 in NR1 format
:POD<n>:SIZE <value> (see page 289)	:POD<n>:SIZE? (see page 289)	<value> ::= {SMALl MEDium LARGe}
:POD<n>:THReshold <type>[suffix] (see page 290)	:POD<n>:THReshold? (see page 290)	<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 616](#))

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 287
 - "[:DIGital<n>:DISPlay](#)" on page 185
 - "[:CHANnel<n>:DISPlay](#)" on page 169
 - "[:VIEW](#)" on page 131
 - "[:BLANK](#)" on page 103
 - "[:STATus](#)" on page 128

:POD<n>:SIZE

N (see [page 616](#))

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 287
 - "[:DIGital<n>:SIZE](#)" on page 188
 - "[:DIGital<n>:POSition](#)" on page 187

:POD<n>:THReshold

N (see [page 616](#))

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 287
 - [":DIGital<n>:THReshold"](#) on page 189
 - [":TRIGger\[:EDGE\]:LEVel"](#) on page 392

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 6"](#)
on page 692

:RECall Commands

Recall previously saved oscilloscope setups and traces. See "Introduction to :RECall Commands" on page 292.

Table 62 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILEname <base_name> (see page 293)	:RECall:FILEname? (see page 293)	<base_name> ::= quoted ASCII string
:RECall:IMAGe[:START] [<file_spec>] (see page 294)	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 page 295)	<path_info> ::= quoted ASCII string
:RECall:SETup[:START] [<file_spec>] (see page 296)	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 616](#))

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 292
 - [":RECall:IMAGe\[:START\]"](#) on page 294
 - [":RECall:SETup\[:START\]"](#) on page 296
 - [":SAVE:FILEname"](#) on page 299

:RECall:IMAGe[:START]

N (see [page 616](#))

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 292
 - [":RECall:FILENAME"](#) on page 293
 - [":SAVE:IMAGe\[:START\]"](#) on page 300

:RECall:PWD

N (see [page 616](#))

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 292
 - [":SAVE:PWD"](#) on page 306

:RECall:SETup[:START]

N (see [page 616](#))

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 292
 - "[:RECall:FILENAME](#)" on page 293
 - "[:SAVE:SETup\[:START\]](#)" on page 307

:SAVE Commands

Save oscilloscope setups and traces, screen images, and data. See "[Introduction to :SAVE Commands](#)" on page 298.

Table 63 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILENAME <base_name> (see page 299)	:SAVE:FILENAME? (see page 299)	<base_name> ::= quoted ASCII string
:SAVE:IMAGE[:START] [<file_spec>] (see page 300)</file_spec>	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:IMAGE:AREA <area> (see page 301)	:SAVE:IMAGE:AREA? (see page 301)	<area> ::= {GRATICule SCReen}
:SAVE:IMAGE:FACTors {0 OFF} {1 ON} (see page 302)	:SAVE:IMAGE:FACTors? (see page 302)	{0 1}
:SAVE:IMAGE:FORMat <format> (see page 303)	:SAVE:IMAGE:FORMat? (see page 303)	<format> ::= {TIFF {BMP BMP24bit} BMP8bit PNG NONE}
:SAVE:IMAGE:INKSaver {0 OFF} {1 ON} (see page 304)	:SAVE:IMAGE:INKSaver? (see page 304)	{0 1}
:SAVE:IMAGE:PALette <palette> (see page 305)	:SAVE:IMAGE:PALette? (see page 305)	<palette> ::= {COLor GRAYscale MONochrome}
n/a	:SAVE:PWD? (see page 306)	<path_info> ::= quoted ASCII string
:SAVE:SETup[:START] [<file_spec>] (see page 307)</file_spec>	n/a	<file_spec> ::= {<internal_loc> <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVEform[:START] [<file_name>] (see page 308)</file_name>	n/a	<file_name> ::= quoted ASCII string

3 Commands by Subsystem

Table 63 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WAVEform:FORMat <format> (see page 309)	:SAVE:WAVEform:FORMat ? (see page 309)	<format> ::= {ALB ASCiixy CSV BINary NONE}
:SAVE:WAVEform:LENGth <length> (see page 310)	:SAVE:WAVEform:LENGth ? (see page 310)	<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 616](#))

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 298
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
 - [":SAVE:SETup\[:START\]"](#) on page 307
 - [":SAVE:WAVEform\[:START\]"](#) on page 308
 - [":SAVE:PWD"](#) on page 306
 - [":RECall:FILEname"](#) on page 293

:SAVE:IMAGe[:START]

N (see [page 616](#))

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 298
 - "[:SAVE:IMAGe:AREA](#)" on page 301
 - "[:SAVE:IMAGe:FACTors](#)" on page 302
 - "[:SAVE:IMAGe:FORMat](#)" on page 303
 - "[:SAVE:IMAGe:INKSaver](#)" on page 304
 - "[:SAVE:IMAGe:PALette](#)" on page 305
 - "[:SAVE:FILEname](#)" on page 299
 - "[:RECall:IMAGe\[:START\]](#)" on page 294

:SAVE:IMAGe:AREA

N (see [page 616](#))

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 298
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
 - [":SAVE:IMAGe:FACTors"](#) on page 302
 - [":SAVE:IMAGe:FORMat"](#) on page 303
 - [":SAVE:IMAGe:INKSaver"](#) on page 304
 - [":SAVE:IMAGe:PALette"](#) on page 305

:SAVE:IMAGe:FACTors

N (see [page 616](#))

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 298
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
 - [":SAVE:IMAGe:AREA"](#) on page 301
 - [":SAVE:IMAGe:FORMat"](#) on page 303
 - [":SAVE:IMAGe:INKSaver"](#) on page 304
 - [":SAVE:IMAGe:PALette"](#) on page 305

:SAVE:IMAGe:FORMat

N (see [page 616](#))

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 298
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
 - [":SAVE:IMAGe:AREA"](#) on page 301
 - [":SAVE:IMAGe:FACTors"](#) on page 302
 - [":SAVE:IMAGe:INKSaver"](#) on page 304
 - [":SAVE:IMAGe:PALette"](#) on page 305
 - [":SAVE:WAVEform:FORMat"](#) on page 309

:SAVE:IMAGe:INKSaver

N (see [page 616](#))

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 298
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
 - [":SAVE:IMAGe:AREA"](#) on page 301
 - [":SAVE:IMAGe:FACTors"](#) on page 302
 - [":SAVE:IMAGe:FORMat"](#) on page 303
 - [":SAVE:IMAGe:PALette"](#) on page 305

:SAVE:IMAGe:PALette

N (see [page 616](#))

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 298
 - "[:SAVE:IMAGe\[:START\]](#)" on page 300
 - "[:SAVE:IMAGe:AREA](#)" on page 301
 - "[:SAVE:IMAGe:FACTors](#)" on page 302
 - "[:SAVE:IMAGe:FORMat](#)" on page 303
 - "[:SAVE:IMAGe:INKSaver](#)" on page 304

:SAVE:PWD

N (see [page 616](#))

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 298
 - [":SAVE:FILENAME"](#) on page 299
 - [":RECALL:PWD"](#) on page 295

:SAVE:SETup[:START]

N (see [page 616](#))

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 298
 - "[:SAVE:FILENAME](#)" on page 299
 - "[:RECall:SETup\[:START\]](#)" on page 296

:SAVE:WAVEform[:START]

N (see [page 616](#))

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 298
 - "[:SAVE:WAVEform:FORMat](#)" on page 309
 - "[:SAVE:WAVEform:LENGth](#)" on page 310
 - "[:SAVE:FILEname](#)" on page 299
 - "[:RECall:SETup\[:START\]](#)" on page 296

:SAVE:WAVEform:FORMat

N (see [page 616](#))

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 298
 - "[:SAVE:WAVEform\[:START\]](#)" on page 308
 - "[:SAVE:WAVEform:LENGth](#)" on page 310
 - "[:SAVE:IMAGe:FORMat](#)" on page 303

:SAVE:WAVEform:LENGth

N (see [page 616](#))

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 298
 - [":SAVE:WAVEform\[:START\]"](#) on page 308
 - [":WAVEform:POINTs"](#) on page 489
 - [":SAVE:WAVEform:FORMat"](#) on page 309

:SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See "Introduction to :SBUS Commands" on page 312.

Table 64 :SBUS Commands Summary

Command	Query	Options and Query Returns
:SBUS:BUSDoctor:ADDRe ss <value> (see page 314)	:SBUS:BUSDoctor:ADDRe ss? (see page 314)	<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 315)	:SBUS:BUSDoctor:BAUDr ate? (see page 315)	<baudrate> ::= {2500000 5000000 10000000}
:SBUS:BUSDoctor:CHANn el <channel> (see page 316)	:SBUS:BUSDoctor:CHANn el? (see page 316)	<channel> ::= {A B}
:SBUS:BUSDoctor:MODE <mode> (see page 317)	:SBUS:BUSDoctor:MODE? (see page 317)	<mode> ::= {ASYNchronous SYNchronous PC}
n/a	:SBUS:CAN:COUNT:ERRor ? (see page 318)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:OVERl oad? (see page 319)	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNT:RESet (see page 320)	n/a	n/a
n/a	:SBUS:CAN:COUNT:TOTal ? (see page 321)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNT:UTILi zation? (see page 322)	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0 OFF} {1 ON}} (see page 323)	:SBUS:DISPlay? (see page 323)	{0 1}
n/a	:SBUS:FLEXray:COUNT:N ULL? (see page 324)	<frame_count> ::= integer in NR1 format
:SBUS:FLEXray:COUNT:R ESet (see page 325)	n/a	n/a
n/a	:SBUS:FLEXray:COUNT:S YNC? (see page 326)	<frame_count> ::= integer in NR1 format

3 Commands by Subsystem

Table 64 :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:FLEXray:COUNT:TOTal? (see page 327)	<frame_count> ::= integer in NR1 format
:SBUS:IIC:ASize <size> (see page 328)	:SBUS:IIC:ASize? (see page 328)	<size> ::= {BIT7 BIT8}
:SBUS:LIN:PARity {{0 OFF} {1 ON}} (see page 329)	:SBUS:LIN:PARity? (see page 329)	{0 1}
:SBUS:MODE <mode> (see page 330)	:SBUS:MODE? (see page 330)	<mode> ::= {IIC SPI CAN LIN FLEXray UART}
:SBUS:SPI:WIDTh <word_width> (see page 331)	:SBUS:SPI:WIDTh? (see page 331)	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see page 332)	:SBUS:UART:BASE? (see page 332)	<base> ::= {ASCIi BINary HEX}
n/a	:SBUS:UART:COUNT:ERROr? (see page 333)	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNT:RESEt (see page 334)	n/a	n/a
n/a	:SBUS:UART:COUNT:RXFRames? (see page 335)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:UART:COUNT:TXFRames? (see page 336)	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see page 337)	:SBUS:UART:FRAMing? (see page 337)	<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 616](#))

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 587](#)

See Also • ["Introduction to :SBUS Commands"](#) on [page 312](#)
• [":TRIGger:FLEXray Commands"](#) on [page 396](#)

:SBUS:BUSDoctor:BAUDrate

N (see [page 616](#))

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 587](#)

See Also • "Introduction to :SBUS Commands" on [page 312](#)
 • ":TRIGger:FLEXray Commands" on [page 396](#)

:SBUS:BUSDoctor:CHANnel

N (see [page 616](#))

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 587](#)

See Also • "[Introduction to :SBUS Commands](#)" on [page 312](#)

• "[:TRIGger:FLEXray Commands](#)" on [page 396](#)

:SBUS:BUSDoctor:MODE

N (see [page 616](#))

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 587

See Also • "Introduction to :SBUS Commands" on page 312
 • ":TRIGger:FLEXray Commands" on page 396

:SBUS:CAN:COUNT:ERRor

N (see [page 616](#))

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 587

- See Also**
- [":SBUS:CAN:COUNT:RESet"](#) on page 320
 - ["Introduction to :SBUS Commands"](#) on page 312
 - [":SBUS:MODE"](#) on page 330
 - [":TRIGger:CAN Commands"](#) on page 369

:SBUS:CAN:COUNT:OVERload**N** (see [page 616](#))**Query Syntax** :SBUS:CAN:COUNT:OVERload?

Returns the overload frame count.

Return Format <frame_count><NL>

<frame_count> ::= integer in NR1 format

Errors • ["-241, Hardware missing"](#) on page 587

- See Also**
-
- [":SBUS:CAN:COUNT:RESet"](#)
- on page 320
-
-
- ["Introduction to :SBUS Commands"](#)
- on page 312
-
-
- [":SBUS:MODE"](#)
- on page 330
-
-
- [":TRIGger:CAN Commands"](#)
- on page 369

:SBUS:CAN:COUNt:RESet

N (see [page 616](#))

Command Syntax :SBUS:CAN:COUNt:RESet

Resets the frame counters.

Errors • "-241, Hardware missing" on [page 587](#)

- See Also**
- [":SBUS:CAN:COUNt:ERRor"](#) on [page 318](#)
 - [":SBUS:CAN:COUNt:OVERload"](#) on [page 319](#)
 - [":SBUS:CAN:COUNt:TOTal"](#) on [page 321](#)
 - [":SBUS:CAN:COUNt:UTILization"](#) on [page 322](#)
 - ["Introduction to :SBUS Commands"](#) on [page 312](#)
 - [":SBUS:MODE"](#) on [page 330](#)
 - [":TRIGger:CAN Commands"](#) on [page 369](#)

:SBUS:CAN:COUNT:TOTAL**N** (see [page 616](#))**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 587

- See Also**
-
- [":SBUS:CAN:COUNT:RESet"](#)
- on page 320
-
-
- ["Introduction to :SBUS Commands"](#)
- on page 312
-
-
- [":SBUS:MODE"](#)
- on page 330
-
-
- [":TRIGger:CAN Commands"](#)
- on page 369

:SBUS:CAN:COUNT:UTILization

N (see [page 616](#))

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 587

- See Also**
- [":SBUS:CAN:COUNT:RESet"](#) on page 320
 - ["Introduction to :SBUS Commands"](#) on page 312
 - [":SBUS:MODE"](#) on page 330
 - [":TRIGger:CAN Commands"](#) on page 369

:SBUS:DISPlay

N (see [page 616](#))

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 587

- See Also**
- ["Introduction to :SBUS Commands"](#) on page 312
 - [":CHANnel<n>:DISPlay"](#) on page 169
 - [":DIGital<n>:DISPlay"](#) on page 185
 - [":POD<n>:DISPlay"](#) on page 288
 - [":VIEW"](#) on page 131
 - [":BLANK"](#) on page 103
 - [":STATus"](#) on page 128

:SBUS:FLEXray:COUNT:NULL

N (see [page 616](#))

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 587](#)

- See Also**
- [":SBUS:FLEXray:COUNT:RESet"](#) on [page 325](#)
 - ["Introduction to :SBUS Commands"](#) on [page 312](#)
 - [":SBUS:MODE"](#) on [page 330](#)
 - [":TRIGger:FLEXray Commands"](#) on [page 396](#)

:SBUS:FLEXray:COUNT:RESet**N** (see [page 616](#))**Command Syntax** :SBUS:FLEXray:COUNT:RESet

Resets the FlexRay frame counters.

Errors • "-241, Hardware missing" on [page 587](#)

- See Also**
-
- [":SBUS:FLEXray:COUNT:NULL"](#)
- on
- [page 324](#)
-
-
- [":SBUS:FLEXray:COUNT:SYNC"](#)
- on
- [page 326](#)
-
-
- [":SBUS:FLEXray:COUNT:TOTAl"](#)
- on
- [page 327](#)
-
-
- ["Introduction to :SBUS Commands"](#)
- on
- [page 312](#)
-
-
- [":SBUS:MODE"](#)
- on
- [page 330](#)
-
-
- [":TRIGger:FLEXray Commands"](#)
- on
- [page 396](#)

:SBUS:FLEXray:COUNT:SYNC

N (see [page 616](#))

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 587](#)

- See Also**
- [":SBUS:FLEXray:COUNT:RESet"](#) on [page 325](#)
 - ["Introduction to :SBUS Commands"](#) on [page 312](#)
 - [":SBUS:MODE"](#) on [page 330](#)
 - [":TRIGger:FLEXray Commands"](#) on [page 396](#)

:SBUS:FLEXray:COUNT:TOTal**N** (see [page 616](#))**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 587

- See Also**
-
- [":SBUS:FLEXray:COUNT:RESet"](#)
- on page 325
-
-
- ["Introduction to :SBUS Commands"](#)
- on page 312
-
-
- [":SBUS:MODE"](#)
- on page 330
-
-
- [":TRIGger:FLEXray Commands"](#)
- on page 396

:SBUS:IIC:ASIZE

N (see [page 616](#))

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 587](#)

See Also • "[Introduction to :SBUS Commands](#)" on [page 312](#)
• "[:TRIGger:IIC Commands](#)" on [page 417](#)

:SBUS:LIN:PARity

N (see [page 616](#))

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 587](#)

See Also • "Introduction to :SBUS Commands" on [page 312](#)
 • ":TRIGger:LIN Commands" on [page 426](#)

:SBUS:MODE

N (see [page 616](#))

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 587](#)

See Also • ["Introduction to :SBUS Commands"](#) on [page 312](#)
• [":TRIGger:MODE"](#) on [page 364](#)
• [":TRIGger:IIC Commands"](#) on [page 417](#)
• [":TRIGger:SPI Commands"](#) on [page 442](#)
• [":TRIGger:CAN Commands"](#) on [page 369](#)
• [":TRIGger:LIN Commands"](#) on [page 426](#)
• [":TRIGger:FLEXray Commands"](#) on [page 396](#)
• [":TRIGger:UART Commands"](#) on [page 457](#)

:SBUS:SPI:WIDTH

N (see [page 616](#))

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 587

See Also • ["Introduction to :SBUS Commands"](#) on page 312
 • [":SBUS:MODE"](#) on page 330
 • [":TRIGger:SPI Commands"](#) on page 442

:SBUS:UART:BASE

N (see [page 616](#))

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 587](#)

See Also • "[Introduction to :SBUS Commands](#)" on [page 312](#)
• "[:TRIGger:UART Commands](#)" on [page 457](#)

:SBUS:UART:COUNT:ERRor**N** (see [page 616](#))**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 587](#)

- See Also**
-
- [":SBUS:UART:COUNT:RESet"](#)
- on
- [page 334](#)
-
-
- ["Introduction to :SBUS Commands"](#)
- on
- [page 312](#)
-
-
- [":SBUS:MODE"](#)
- on
- [page 330](#)
-
-
- [":TRIGger:UART Commands"](#)
- on
- [page 457](#)

:SBUS:UART:COUNt:RESet

N (see [page 616](#))

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 587
- See Also**
- [":SBUS:UART:COUNt:ERRor"](#) on page 333
 - [":SBUS:UART:COUNt:RXFRames"](#) on page 335
 - [":SBUS:UART:COUNt:TXFRames"](#) on page 336
 - ["Introduction to :SBUS Commands"](#) on page 312
 - [":SBUS:MODE"](#) on page 330
 - [":TRIGger:UART Commands"](#) on page 457

:SBUS:UART:COUNT:RXFRames**N** (see [page 616](#))**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 587](#)

- See Also**
-
- [":SBUS:UART:COUNT:RESet"](#)
- on
- [page 334](#)
-
-
- ["Introduction to :SBUS Commands"](#)
- on
- [page 312](#)
-
-
- [":SBUS:MODE"](#)
- on
- [page 330](#)
-
-
- [":TRIGger:UART Commands"](#)
- on
- [page 457](#)

:SBUS:UART:COUNt:TXFRames

N (see [page 616](#))

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 587](#)

- See Also**
- [":SBUS:UART:COUNt:RESet"](#) on [page 334](#)
 - ["Introduction to :SBUS Commands"](#) on [page 312](#)
 - [":SBUS:MODE"](#) on [page 330](#)
 - [":TRIGger:UART Commands"](#) on [page 457](#)

:SBUS:UART:FRAMing

N (see [page 616](#))

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 587](#)

See Also • ["Introduction to :SBUS Commands"](#) on [page 312](#)
 • [":TRIGger:UART Commands"](#) on [page 457](#)

:SYSTEM Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTEM Commands" on page 338.

Table 65 :SYSTEM Commands Summary

Command	Query	Options and Query Returns
:SYSTEM:DATE <date> (see page 339)	:SYSTEM:DATE? (see page 339)	<date> ::= <year>,<month>,<day> <year> ::= 4-digit year in NR1 format <month> ::= {1,...,12 JANuary FEBruary MARch APRil MAY JUNE JULy AUGust SEPTember OCTober NOVember DECember} <day> ::= {1,..31}
:SYSTEM:DSP <string> (see page 340)	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTEM:ERROR? (see page 341)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 585).
:SYSTEM:LOCK (see page 342)	:SYSTEM:LOCK? (see page 342)	<value> ::= {ON OFF}
:SYSTEM:SETup <setup_data> (see page 343)	:SYSTEM:SETup? (see page 343)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTEM:TIME <time> (see page 345)	:SYSTEM:TIME? (see page 345)	<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 616](#))

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 338
 - [":SYSTem:TIME"](#) on page 345

:SYSTem:DSP

N (see [page 616](#))

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 338

:SYSTem:ERRor

C (see [page 616](#))

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 585.

- See Also**
- "[Introduction to :SYSTem Commands](#)" on page 338
 - "[*ESR \(Standard Event Status Register\)](#)" on page 76
 - "[*CLS \(Clear Status\)](#)" on page 73

:SYSTem:LOCK

N (see [page 616](#))

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 GPIB.

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 338

:SYSTEM:SETup

C (see [page 616](#))

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 338
 - ["*LRN \(Learn Device Setup\)"](#) on page 79

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.
```

3 Commands by Subsystem

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"  
' command:  
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString  
CheckForInstrumentErrors
```

Example program from the start: "[VISA COM Example in Visual Basic 6](#)"
on page 692

:SYSTem:TIME

N (see [page 616](#))

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 338
 - "[:SYSTEM:DATE](#)" on page 339

:TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 347.

Table 66 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 348)	:TIMebase:MODE? (see page 348)	<value> ::= {MAIN WINDow XY ROLL}
:TIMebase:POSition <pos> (see page 349)	:TIMebase:POSition? (see page 349)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 350)	:TIMebase:RANGe? (see page 350)	<range_value> ::= 5 ns through 500 s in NR3 format
:TIMebase:REFClock {{0 OFF} {1 ON}} (see page 351)	:TIMebase:REFClock? (see page 351)	{0 1}
:TIMebase:REFerence {LEFT CENTer RIGHT} (see page 352)	:TIMebase:REFerence? (see page 352)	<return_value> ::= {LEFT CENTer RIGHT}
:TIMebase:SCALe <scale_value> (see page 353)	:TIMebase:SCALe? (see page 353)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0 OFF} {1 ON}} (see page 354)	:TIMebase:VERNier? (see page 354)	{0 1}
:TIMebase:WINDow:POSition <pos> (see page 355)	:TIMebase:WINDow:POSition? (see page 355)	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 356)	:TIMebase:WINDow:RANGe? (see page 356)	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCALe <scale_value> (see page 357)	:TIMebase:WINDow:SCALe? (see page 357)	<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 616](#))

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 347
 - ["*RST \(Reset\)"](#) on page 83
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:POSition"](#) on page 349
 - [":TIMEbase:REFerence"](#) on page 352

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 6"](#) on page 692

:TIMEbase:POSition

C (see [page 616](#))

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 347
 - [":TIMEbase:REFerence"](#) on page 352
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:SCALE"](#) on page 353
 - [":TIMEbase:WINDow:POSition"](#) on page 355
 - [":TIMEbase:DELay"](#) on page 579

:TIMEbase:RANGe

C (see [page 616](#))

Command Syntax :TIMEbase:RANGe <range_value>

<range_value> ::= 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>

<range_value> ::= 5 ns through 500 s in NR3 format

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 347
 - [":TIMEbase:MODE"](#) on page 348
 - [":TIMEbase:SCALE"](#) on page 353
 - [":TIMEbase:WINDow:RANGe"](#) on page 356

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 6"](#) on page 692

:TIMEbase:REFClock

N (see [page 616](#))

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 140).

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 140

:TIMEbase:REFErence

C (see [page 616](#))

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 347
 - [":TIMEbase:MODE"](#) on page 348

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 6"](#) on page 692

:TIMEbase:SCALE

N (see [page 616](#))

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 347
 - [":TIMEbase:RANGE"](#) on page 350
 - [":TIMEbase:WINDOW:SCALE"](#) on page 357
 - [":TIMEbase:WINDOW:RANGE"](#) on page 356

:TIMEbase:VERNier

N (see [page 616](#))

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 347

:TIMEbase:WINDow:POSition

C (see [page 616](#))

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>

<value> ::= position value in seconds

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 347
 - [":TIMEbase:MODE"](#) on page 348
 - [":TIMEbase:POSition"](#) on page 349
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:SCALE"](#) on page 353
 - [":TIMEbase:WINDow:RANGe"](#) on page 356
 - [":TIMEbase:WINDow:SCALE"](#) on page 357

:TIMEbase:WINDow:RANGe

C (see [page 616](#))

Command Syntax :TIMEbase:WINDow:RANGe <range value>

<range value> ::= range value in seconds in NR3 format

The :TIMEbase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the 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>

<value> ::= range value in seconds

- See Also**
- ["Introduction to :TIMEbase Commands"](#) on page 347
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:POSition"](#) on page 349
 - [":TIMEbase:SCALE"](#) on page 353

:TIMEbase:WINDow:SCALe

N (see [page 616](#))

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 347
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:POSition"](#) on page 349
 - [":TIMEbase:SCALe"](#) on page 353
 - [":TIMEbase:WINDow:RANGe"](#) on page 356

:TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "Introduction to :TRIGger Commands" on page 358
- "General :TRIGger Commands" on page 361
- ":TRIGger:CAN Commands" on page 369
- ":TRIGger:DURation Commands" on page 380
- ":TRIGger:EBURst Commands" on page 386
- ":TRIGger[:EDGE] Commands" on page 390
- ":TRIGger:FLEXray Commands" on page 396
- ":TRIGger:GLITch Commands" on page 408 (Pulse Width trigger)
- ":TRIGger:IIC Commands" on page 417
- ":TRIGger:LIN Commands" on page 426
- ":TRIGger:SEQuence Commands" on page 434
- ":TRIGger:SPI Commands" on page 442
- ":TRIGger:TV Commands" on page 451
- ":TRIGger:USB Commands" on page 471
- ":TRIGger:UART Commands" on page 457

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 368) 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 364).

- **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 page 362)	:TRIGger:HFReject? (see page 362)	{0 1}
:TRIGger:HOLDoff <holdoff_time> (see page 363)	:TRIGger:HOLDoff? (see page 363)	<holdoff_time> ::= 60 ns to 10 s in NR3 format
:TRIGger:MODE <mode> (see page 364)	:TRIGger:MODE? (see page 364)	<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 365)	:TRIGger:NREJect? (see page 365)	{0 1}
:TRIGger:PATtern <value>, <mask> [, <edge source>, <edge>] (see page 366)	:TRIGger:PATtern? (see page 367)	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnxxxxn"; 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 368)	:TRIGger:SWEep? (see page 368)	<sweep> ::= {AUTO NORMal}

:TRIGger:HFReject

C (see [page 616](#))

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 358
 - "[:TRIGger\[:EDGE\]:REJECT](#)" on page 393

:TRIGger:HOLDoff

C (see [page 616](#))

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 358

:TRIGger:MODE

C (see [page 616](#))

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 358
 - [":TRIGger:SWEep"](#) on page 368
 - [":TIMEbase:MODE"](#) on page 348

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 6"](#) on page 692

:TRIGger:NREJect

C (see [page 616](#))

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 358

:TRIGger:PATtern

C (see page 616)

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> ::= "0xnxxxxn"; 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
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	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 358
 - [":TRIGger:MODE"](#) on page 364

:TRIGger:SWEEp

C (see [page 616](#))

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 358

:TRIGger:CAN Commands

Table 68 :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PAITtern:DATA <value>, <mask> (see page 371)	:TRIGger:CAN:PAITtern:DATA? (see page 371)	<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:PAITtern:DATA:LENGth <length> (see page 372)	:TRIGger:CAN:PAITtern:DATA:LENGth? (see page 372)	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PAITtern:ID <value>, <mask> (see page 373)	:TRIGger:CAN:PAITtern:ID? (see page 373)	<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:PAITtern:ID:MODE <value> (see page 374)	:TRIGger:CAN:PAITtern:ID:MODE? (see page 374)	<value> ::= {STANdard EXTENDED} (with Option AMS)
:TRIGger:CAN:SAMPlepo int <value> (see page 375)	:TRIGger:CAN:SAMPlepo int? (see page 375)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:TRIGger:CAN:SIGNal:BAUDrate <baudrate> (see page 376)	:TRIGger:CAN:SIGNal:BAUDrate? (see page 376)	<baudrate> ::= {10000 20000 33300 50000 62500 83300 100000 125000 250000 500000 800000 1000000}

3 Commands by Subsystem

Table 68 :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see page 377)	:TRIGger:CAN:SOURce? (see page 377)	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 378)	:TRIGger:CAN:TRIGger? (see page 379)	<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 616](#))

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 587](#)

See Also

- "[Introduction to :TRIGger Commands](#)" on [page 358](#)
- "[:TRIGger:CAN:PATtern:DATA:LENGth](#)" on [page 372](#)
- "[:TRIGger:CAN:PATtern:ID](#)" on [page 373](#)

:TRIGger:CAN:PATtern:DATA:LENGth

N (see [page 616](#))

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 587](#)

See Also • ["Introduction to :TRIGger Commands"](#) on [page 358](#)
• [":TRIGger:CAN:PATtern:DATA"](#) on [page 371](#)
• [":TRIGger:CAN:SOURce"](#) on [page 377](#)

:TRIGger:CAN:PATtern:ID

N (see [page 616](#))

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 587](#)

See Also

- ["Introduction to :TRIGger Commands"](#) on [page 358](#)
- [":TRIGger:CAN:PATtern:ID:MODE"](#) on [page 374](#)
- [":TRIGger:CAN:PATtern:DATA"](#) on [page 371](#)

:TRIGger:CAN:PATtern:ID:MODE

N (see [page 616](#))

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 587](#)

See Also • ["Introduction to :TRIGger Commands"](#) on [page 358](#)
• [":TRIGger:MODE"](#) on [page 364](#)
• [":TRIGger:CAN:PATtern:DATA"](#) on [page 371](#)
• [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on [page 372](#)
• [":TRIGger:CAN:PATtern:ID"](#) on [page 373](#)

:TRIGger:CAN:SAMPlEpoint

N (see [page 616](#))

Command Syntax :TRIGger:CAN:SAMPlEpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger:CAN:SAMPlEpoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax :TRIGger:CAN:SAMPlEpoint?

The :TRIGger:CAN:SAMPlEpoint? query returns the current CAN sample point setting.

Return Format <value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:CAN:TRIGger"](#) on page 378

:TRIGger:CAN:SIGNal:BAUDrate

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:CAN:TRIGger"](#) on page 378
 - [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 581
 - [":TRIGger:CAN:SOURce"](#) on page 377

:TRIGger:CAN:SOURce

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:CAN:TRIGger"](#) on page 378
 - [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 581

:TRIGger:CAN:TRIGger

N (see page 616)

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 587

See Also

- ["Introduction to :TRIGger Commands"](#) on page 358
- [":TRIGger:MODE"](#) on page 364
- [":TRIGger:CAN:PATtern:DATA"](#) on page 371
- [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 372
- [":TRIGger:CAN:PATtern:ID"](#) on page 373
- [":TRIGger:CAN:PATtern:ID:MODE"](#) on page 374
- [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 581
- [":TRIGger:CAN:SOURce"](#) on page 377

:TRIGger:DURation Commands

Table 69 :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see page 381)	:TRIGger:DURation:GREaterthan? (see page 381)	<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 page 382)	:TRIGger:DURation:LESSthan? (see page 382)	<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 page 383)	:TRIGger:DURation:PATTERN? (see page 383)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnxxxxxx" n ::= {0,...,9 A,...,F}
:TRIGger:DURation:QUALifier <qualifier> (see page 384)	:TRIGger:DURation:QUALifier? (see page 384)	<qualifier> ::= {GREaterthan LESSthan INRange OUTRange TIMEout}
:TRIGger:DURation:RANGE <greater than time>[suffix], <less than time>[suffix] (see page 385)	:TRIGger:DURation:RANGE? (see page 385)	<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 616](#))

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 :TRIGger:DURation:GREaterthan?

The :TRIGger:DURation:GREaterthan? query returns the minimum duration time for the defined pattern.

Return Format <greater than time><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:DURation:PATtern"](#) on page 383
 - [":TRIGger:DURation:QUALifier"](#) on page 384
 - [":TRIGger:MODE"](#) on page 364

:TRIGger:DURation:LESSthan

N (see [page 616](#))

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 :TRIGger:DURation:LESSthan?

The :TRIGger:DURation:LESSthan? query returns the duration time for the defined pattern.

Return Format <less than time><NL>

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 358
 - "[:TRIGger:DURation:PATtern](#)" on page 383
 - "[:TRIGger:DURation:QUALifier](#)" on page 384
 - "[:TRIGger:MODE](#)" on page 364

:TRIGger:DURation:PATtern

N (see page 616)

Command Syntax :TRIGger:DURation:PATtern <value>, <mask>
 <value> ::= integer or <string>
 <mask> ::= integer or <string>
 <string> ::= "0xnmmnnn"; 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
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	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 358
 - ":TRIGger:PATtern" on page 366

:TRIGger:DURation:QUALifier

N (see [page 616](#))

Command Syntax :TRIGger:DURation:QUALifier <qualifier>

<qualifier> ::= {GREaterthan | LESSthan | INRange | OTRange | 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 OTRange 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 358
 - [":TRIGger:DURation:GREaterthan"](#) on page 381
 - [":TRIGger:DURation:LESSthan"](#) on page 382
 - [":TRIGger:DURation:RANGE"](#) on page 385

:TRIGger:EBURst Commands

Table 70 :TRIGger:EBURst Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EBURst:COUNT <count> (see page 387)	:TRIGger:EBURst:COUNT? (see page 387)	<count> ::= integer in NR1 format
:TRIGger:EBURst:IDLE <time_value> (see page 388)	:TRIGger:EBURst:IDLE? (see page 388)	<time_value> ::= time in seconds in NR3 format
:TRIGger:EBURst:SLOPe <slope> (see page 389)	:TRIGger:EBURst:SLOPe? (see page 389)	<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 616](#))

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 358
 - "[:TRIGger:EBURst:SLOPe](#)" on page 389
 - "[:TRIGger:EBURst:IDLE](#)" on page 388

:TRIGger:EBURst:IDLE

N (see [page 616](#))

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 358](#)
 - "[:TRIGger:EBURst:SLOPe](#)" on [page 389](#)
 - "[:TRIGger:EBURst:COUnT](#)" on [page 387](#)

:TRIGger:EBURst:SLOPe

N (see [page 616](#))

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 358
 - "[:TRIGger:EBURst:IDLE](#)" on page 388
 - "[:TRIGger:EBURst:COUNT](#)" on page 387

:TRIGger[:EDGE] Commands

Table 71 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC DC LF} (see page 391)	:TRIGger[:EDGE]:COUPling? (see page 391)	{AC DC LF}
:TRIGger[:EDGE]:LEVel <level> [, <source>] (see page 392)	:TRIGger[:EDGE]:LEVel ? [<source>] (see page 392)	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 page 393)	:TRIGger[:EDGE]:REJect? (see page 393)	{OFF LF HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 394)	:TRIGger[:EDGE]:SLOPe ? (see page 394)	<polarity> ::= {POSitive NEGative EITHER ALTernate}
:TRIGger[:EDGE]:SOURC e <source> (see page 395)	:TRIGger[:EDGE]:SOURC e? (see page 395)	<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

C (see [page 616](#))

Command Syntax :TRIGger[:EDGE]:COUPLing <coupling>
 <coupling> ::= {AC | DC | LFReject}

The :TRIGger[:EDGE]:COUPLing command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LFReject.

- AC coupling places a high-pass filter (10 Hz for analog channels, and 3.5 Hz for all External trigger inputs) in the trigger path, removing dc offset voltage from the trigger waveform. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset.
- LFReject coupling places a 50 KHz high-pass filter in the trigger path.
- DC coupling allows dc and ac signals into the trigger path.

NOTE

The :TRIGger[:EDGE]:COUPLing and the :TRIGger[:EDGE]:REJect selections are coupled. Changing the setting of the :TRIGger[:EDGE]:REJect can change the COUPLing setting.

Query Syntax :TRIGger[:EDGE]:COUPLing?

The :TRIGger[:EDGE]:COUPLing? query returns the current coupling selection.

Return Format <coupling><NL>
 <coupling> ::= {AC | DC | LFR}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger\[:EDGE\]:REJect"](#) on page 393

:TRIGger[:EDGE]:LEVel

C (see [page 616](#))

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 358
 - [":TRIGger\[:EDGE\]:SOURce"](#) on page 395
 - [":POD<n>:THReshold"](#) on page 290
 - [":DIGital<n>:THReshold"](#) on page 189

:TRIGger[:EDGE]:REJect

C (see [page 616](#))

Command Syntax :TRIGger[:EDGE]:REJect <reject>
 <reject> ::= {OFF | LFReject | HFReject}

The :TRIGger[:EDGE]:REJect command turns the low-frequency or high-frequency reject filter on or off. You can turn on one of these filters at a time.

- The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use the high frequency reject filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.
- The low frequency reject filter adds a 50 kHz high-pass filter in series with the trigger waveform to remove any unwanted low frequency components from a trigger waveform, such as power line frequencies, that can interfere with proper triggering.

NOTE

The :TRIGger[:EDGE]:REJect and the :TRIGger[:EDGE]:COUpling selections are coupled. Changing the setting of the :TRIGger[:EDGE]:COUpling can change the COUpling setting.

Query Syntax :TRIGger[:EDGE]:REJect?

The :TRIGger[:EDGE]:REJect? query returns the current status of the reject filter.

Return Format <reject><NL>
 <reject> ::= {OFF | LFR | HFR}

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 358
 - "[:TRIGger:HFReject](#)" on page 362
 - "[:TRIGger\[:EDGE\]:COUpling](#)" on page 391

:TRIGger[:EDGE]:SLOPe

C (see [page 616](#))

Command Syntax :TRIGger[:EDGE]:SLOPe <slope>
<slope> ::= {NEGative | POSitive | EITHer | ALTErnate}

The :TRIGger[:EDGE]:SLOPe command specifies the slope of the edge for the trigger. The SLOPe command is not valid in TV trigger mode. Instead, use :TRIGger:TV:POLarity to set the polarity in TV trigger mode.

Query Syntax :TRIGger[:EDGE]:SLOPe?

The :TRIGger[:EDGE]:SLOPe? query returns the current trigger slope.

Return Format <slope><NL>
<slope> ::= {NEG | POS | EITH | ALT}

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:TV:POLarity"](#) on page 454

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 6"](#) on page 692

:TRIGger[:EDGE]:SOURce

C (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364

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 6"](#) on page 692

:TRIGger:FLEXray Commands

Table 72 :TRIGger:FLEXray Commands Summary

Command	Query	Options and Query Returns
:TRIGger:FLEXray:ERRor:TYPE <error_type> (see page 397)	:TRIGger:FLEXray:ERRor:TYPE? (see page 397)	<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 page 399)	:TRIGger:FLEXray:FRAMe:CCBase? (see page 399)	<cycle_count_base> ::= integer from 0-63
:TRIGger:FLEXray:FRAMe:CCRepetition <cycle_count_repetition> (see page 400)	:TRIGger:FLEXray:FRAMe:CCRepetition? (see page 400)	<cycle_count_repetition> ::= {ALL <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:FRAMe:ID <frame_id> (see page 401)	:TRIGger:FLEXray:FRAMe:ID? (see page 401)	<frame_id> ::= {ALL <frame #>} <frame #> ::= integer from 1-2047
:TRIGger:FLEXray:FRAMe:TYPE <frame_type> (see page 402)	:TRIGger:FLEXray:FRAMe:TYPE? (see page 402)	<frame_type> ::= {NORMAL STARTup NULL SYNC NSTArtup NNULl NSYNc ALL}
:TRIGger:FLEXray:TIME:CBASE <cycle_base> (see page 403)	:TRIGger:FLEXray:TIME:CBASE? (see page 403)	<cycle_base> ::= integer from 0-63
:TRIGger:FLEXray:TIME:CREPpetition <cycle_repetition> (see page 404)	:TRIGger:FLEXray:TIME:CREPpetition? (see page 404)	<cycle_repetition> ::= {ALL <rep #>} <rep #> ::= integer from 2-64
:TRIGger:FLEXray:TIME:SEGMENT <segment_type> (see page 405)	:TRIGger:FLEXray:TIME:SEGMENT? (see page 405)	<segment_type> ::= {STATic DYNamic SYMbol IDLE}
:TRIGger:FLEXray:TIME:SLOT <slot_type>, <slot_id> (see page 406)	:TRIGger:FLEXray:TIME:SLOT? (see page 406)	<slot_type> ::= {ALL EMPTY} <slot_id> ::= {ALL <slot #>} <slot #> ::= integer from 1-2047
:TRIGger:FLEXray:TRIGger <condition> (see page 407)	:TRIGger:FLEXray:TRIGger? (see page 407)	<condition> ::= {FRAME TIME ERRor}

:TRIGger:FLEXray:ERRor:TYPE

N (see page 616)

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}
```

3 Commands by Subsystem

- Errors**
- "-241, Hardware missing" on page 587
- See Also**
- "Introduction to :TRIGger Commands" on page 358
 - ":TRIGger:FLEXray:TRIGger" on page 407

:TRIGger:FLEXray:FRAME:CCBase

N (see [page 616](#))

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 587](#)

See Also • "Introduction to :TRIGger Commands" on [page 358](#)
 • ":TRIGger:FLEXray:TRIGger" on [page 407](#)

:TRIGger:FLEXray:FRAME:CCRepetition

N (see [page 616](#))

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 587

See Also • ["Introduction to :TRIGger Commands"](#) on page 358
• [":TRIGger:FLEXray:TRIGger"](#) on page 407

:TRIGger:FLEXray:FRAME:ID

N (see [page 616](#))

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 587](#)

See Also • ["Introduction to :TRIGger Commands"](#) on [page 358](#)
 • [":TRIGger:MODE"](#) on [page 364](#)
 • [":TRIGger:FLEXray:TRIGger"](#) on [page 407](#)

:TRIGger:FLEXray:FRAME:TYPE

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:FLEXray:TRIGger"](#) on page 407

:TRIGger:FLEXray:TIME:CBASe

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:FLEXray:TRIGger"](#) on page 407

:TRIGger:FLEXray:TIME:CREPetition

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:FLEXray:TRIGger](#)" on page 407

:TRIGger:FLEXray:TIME:SEGment

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:FLEXray:TRIGger"](#) on page 407

:TRIGger:FLEXray:TIME:SLOT

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:FLEXray:TRIGger"](#) on page 407

:TRIGger:FLEXray:TRIGger

N (see [page 616](#))

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 587](#)

- See Also**
- "Introduction to :TRIGger Commands" on [page 358](#)
 - ":TRIGger:MODE" on [page 364](#)
 - ":TRIGger:FLEXray:ERRor:TYPE" on [page 397](#)
 - ":TRIGger:FLEXray:FRAMe:CCBase" on [page 399](#)
 - ":TRIGger:FLEXray:FRAMe:CCRepetition" on [page 400](#)
 - ":TRIGger:FLEXray:FRAMe:ID" on [page 401](#)
 - ":TRIGger:FLEXray:FRAMe:TYPE" on [page 402](#)
 - ":TRIGger:FLEXray:TIME:CBase" on [page 403](#)
 - ":TRIGger:FLEXray:TIME:CREpetition" on [page 404](#)
 - ":TRIGger:FLEXray:TIME:SEGment" on [page 405](#)
 - ":TRIGger:FLEXray:TIME:SLOT" on [page 406](#)

:TRIGger:GLITch Commands

Table 73 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREAt erthan <greater than time>[suffix] (see page 410)	:TRIGger:GLITch:GREAt erthan? (see page 410)	<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 page 411)	:TRIGger:GLITch:LESSt han? (see page 411)	<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 page 412)	:TRIGger:GLITch:LEVel ? (see page 412)	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 page 413)	:TRIGger:GLITch:POLAr ity? (see page 413)	<polarity> ::= {POSitive NEGative}
:TRIGger:GLITch:QUALi fier <qualifier> (see page 414)	:TRIGger:GLITch:QUALi fier? (see page 414)	<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 page 415)	:TRIGger:GLITCh:RANGe? (see page 415)	<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 page 416)	:TRIGger:GLITCh:SOURce? (see page 416)	<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 616](#))

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 :TRIGger:GLITch:GREaterthan?

The :TRIGger:GLITch:GREaterthan? query returns the minimum pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format <greater_than_time><NL>.

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 358
 - "[:TRIGger:GLITch:SOURce](#)" on page 416
 - "[:TRIGger:GLITch:QUALifier](#)" on page 414
 - "[:TRIGger:MODE](#)" on page 364

:TRIGger:GLITCh:LESSthan

N (see [page 616](#))

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 358
 - [":TRIGger:GLITCh:SOURce"](#) on page 416
 - [":TRIGger:GLITCh:QUALifier"](#) on page 414
 - [":TRIGger:MODE"](#) on page 364

:TRIGger:GLITch:LEVel

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:GLITch:SOURce"](#) on page 416

:TRIGger:GLITch:POLarity

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:GLITch:SOURce"](#) on page 416

:TRIGger:GLITch:QUALifier

N (see [page 616](#))

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 358
 - "[:TRIGger:GLITch:SOURce](#)" on page 416
 - "[:TRIGger:MODE](#)" on page 364

:TRIGger:GLITch:SOURce

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:GLITch:LEVel"](#) on page 412
 - [":TRIGger:GLITch:POLarity"](#) on page 413
 - [":TRIGger:GLITch:QUALifier"](#) on page 414
 - [":TRIGger:GLITch:RANGe"](#) on page 415

Example Code • ["Example Code"](#) on page 395

:TRIGger:IIC Commands

Table 74 :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATtern:ADDRESS <value> (see page 418)	:TRIGger:IIC:PATtern:ADDRESS? (see page 418)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC:PATtern:DATA <value> (see page 419)	:TRIGger:IIC:PATtern:DATA? (see page 419)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC:PATtern:DATA2 <value> (see page 420)	:TRIGger:IIC:PATtern:DATA2? (see page 420)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,...,9 A,...,F}
:TRIGger:IIC[:SOURce]:CLOCK <source> (see page 421)	:TRIGger:IIC[:SOURce]:CLOCK? (see page 421)	<source> ::= {CHANnel<n> EXTErnal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see page 422)	:TRIGger:IIC[:SOURce]:DATA? (see page 422)	<source> ::= {CHANnel<n> EXTErnal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see page 423)	:TRIGger:IIC:TRIGger:QUALifier? (see page 423)	<value> ::= {EQUal NOTequal LESSthan GREATERthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see page 424)	:TRIGger:IIC:TRIGger[:TYPE]? (see page 424)	<type> ::= {START STOP READ7 READEprom WRITe7 WRITe10 NACKnowledge ANACKnowledge R7Data2 W7Data2 REStart}

:TRIGger:IIC:PATtern:ADDRESS

N (see [page 616](#))

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 358
 - [":TRIGger:IIC:PATtern:DATA"](#) on page 419
 - [":TRIGger:IIC:PATtern:DATA2"](#) on page 420
 - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 424

:TRIGger:IIC:PATtern:DATA

N (see [page 616](#))

Command Syntax :TRIGger:IIC:PATtern:DATA <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC:PATtern:DATA command sets IIC data. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax :TRIGger:IIC:PATtern:DATA?

The :TRIGger:IIC:PATtern:DATA? query returns the current pattern for IIC data.

Return Format <value><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:IIC:PATtern:ADDRess"](#) on page 418
 - [":TRIGger:IIC:PATtern:DATA2"](#) on page 420
 - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 424

:TRIGger:IIC:PATtern:DATA2

N (see [page 616](#))

Command Syntax :TRIGger:IIC:PATtern:DATA2 <value>

<value> ::= integer or <string>

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

The :TRIGger:IIC:PATtern:DATA2 command sets IIC data 2. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax :TRIGger:IIC:PATtern:DATA2?

The :TRIGger:IIC:PATtern:DATA2? query returns the current pattern for IIC data 2.

Return Format <value><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:IIC:PATtern:ADDRESS"](#) on page 418
 - [":TRIGger:IIC:PATtern:DATA"](#) on page 419
 - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 424

:TRIGger:IIC:SOURce:CLOCK

N (see [page 616](#))

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 358
 - [":TRIGger:IIC:SOURce:DATA"](#) on page 422

:TRIGger:IIC:SOURce:DATA

N (see [page 616](#))

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 358
 - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 421

:TRIGger:IIC:TRIGger:QUALifier

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 424

:TRIGger:IIC:TRIGger[:TYPE]

N (see [page 616](#))

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 618](#)).

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 358
 - [":TRIGger:MODE"](#) on page 364

- `":TRIGger:IIC:PATtern:ADDRes"` on page 418
- `":TRIGger:IIC:PATtern:DATA"` on page 419
- `":TRIGger:IIC:PATtern:DATA2"` on page 420
- `":TRIGger:IIC:TRIGger:QUALifier"` on page 423
- "Long Form to Short Form Truncation Rules" on page 618

:TRIGger:LIN Commands

Table 75 :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see page 427)	:TRIGger:LIN:ID? (see page 427)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,...,9 A,...,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 1} for binary <string> ::= "0xnn" where n ::= {0,...,9 A,...,F} for hexadecimal
:TRIGger:LIN:SAMplepo int <value> (see page 428)	:TRIGger:LIN:SAMplepo int? (see page 428)	<value> ::= {60 62.5 68 70 75 80 87.5} in NR3 format
:TRIGger:LIN:SIGNAL:B AUDrate <baudrate> (see page 429)	:TRIGger:LIN:SIGNAL:B AUDrate? (see page 429)	<baudrate> ::= {2400 9600 19200}
:TRIGger:LIN:SOURce <source> (see page 430)	:TRIGger:LIN:SOURce? (see page 430)	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std> (see page 431)	:TRIGger:LIN:STANdard ? (see page 431)	<std> ::= {LIN13 LIN20}
:TRIGger:LIN:SYNCbrea k <value> (see page 432)	:TRIGger:LIN:SYNCbrea k? (see page 432)	<value> ::= integer = {11 12 13}
:TRIGger:LIN:TRIGger <condition> (see page 433)	:TRIGger:LIN:TRIGger? (see page 433)	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak ID} (with Option AMS)

:TRIGger:LIN:ID

N (see [page 616](#))

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 587

See Also • ["Introduction to :TRIGger Commands"](#) on page 358
• [":TRIGger:MODE"](#) on page 364
• [":TRIGger:LIN:TRIGger"](#) on page 433
• [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 582
• [":TRIGger:LIN:SOURce"](#) on page 430

:TRIGger:LIN:SAMPlEpoint

N (see [page 616](#))

Command Syntax :TRIGger:LIN:SAMPlEpoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

The :TRIGger:LIN:SAMPlEpoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

NOTE

The sample point values are not limited by the baud rate.

Query Syntax :TRIGger:LIN:SAMPlEpoint?

The :TRIGger:LIN:SAMPlEpoint? query returns the current LIN sample point setting.

Return Format <value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format

- See Also**
- "[Introduction to :TRIGger Commands](#)" on [page 358](#)
 - "[:TRIGger:MODE](#)" on [page 364](#)
 - "[:TRIGger:LIN:TRIGger](#)" on [page 433](#)

:TRIGger:LIN:SIGNal:BAUDrate

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:LIN:TRIGger"](#) on page 433
 - [":TRIGger:LIN:SIGNal:DEFinition"](#) on page 582
 - [":TRIGger:LIN:SOURce"](#) on page 430

:TRIGger:LIN:SOURce

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:LIN:TRIGger"](#) on page 433
 - [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 582

:TRIGger:LIN:STANdard

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:LIN:SIGNAL:DEFinition](#)" on page 582
 - "[:TRIGger:LIN:SOURce](#)" on page 430

:TRIGger:LIN:SYNCbreak

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:LIN:SIGNal:DEFinition](#)" on page 582
 - "[:TRIGger:LIN:SOURce](#)" on page 430

:TRIGger:LIN:TRIGger

N (see [page 616](#))

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 587

See Also • ["Introduction to :TRIGger Commands"](#) on page 358
 • [":TRIGger:MODE"](#) on page 364
 • [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 582
 • [":TRIGger:LIN:SOURce"](#) on page 430

:TRIGger:SEQuence Commands

Table 76 :TRIGger:SEQuence Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SEQuence:COU Nt <count> (see page 435)	:TRIGger:SEQuence:COU Nt? (see page 435)	<count> ::= integer in NR1 format
:TRIGger:SEQuence:EDG E{1 2} <source>, <slope> (see page 436)	:TRIGger:SEQuence:EDG E{1 2}? (see page 436)	<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 page 437)	:TRIGger:SEQuence:FIN D? (see page 437)	<value> ::= {PATtern1,ENTERed PATtern1,EXITed EDGE1 PATtern1,AND,EDGE1}
:TRIGger:SEQuence:PAT Tern{1 2} <value>, <mask> (see page 438)	:TRIGger:SEQuence:PAT Tern{1 2}? (see page 438)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnxxxxn" n ::= {0,...,9 A,...,F}
:TRIGger:SEQuence:RES et <value> (see page 439)	:TRIGger:SEQuence:RES et? (see page 439)	<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 page 440)	:TRIGger:SEQuence:TIM er? (see page 440)	<time_value> ::= time from 100 ns to 10 seconds in NR3 format
:TRIGger:SEQuence:TRI Gger <value> (see page 441)	:TRIGger:SEQuence:TRI Gger? (see page 441)	<value> ::= {PATtern2,ENTERed PATtern2,EXITed EDGE2 PATtern2,AND,EDGE2 EDGE2,COUNT EDGE2,COUNT,NREFind}

:TRIGger:SEQuence:COUNT

N (see [page 616](#))

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 358
 - [":TRIGger:SEQuence:TRIGger"](#) on page 441
 - [":TRIGger:SEQuence:EDGE"](#) on page 436

:TRIGger:SEQuence:EDGE

N (see [page 616](#))

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 358
 - [":TRIGger:SEQuence:FIND"](#) on page 437
 - [":TRIGger:SEQuence:TRIGger"](#) on page 441
 - [":TRIGger:SEQuence:RESet"](#) on page 439
 - [":TRIGger:SEQuence:COUNT"](#) on page 435

:TRIGger:SEQuence:FIND

N (see [page 616](#))

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 358
 - [":TRIGger:SEQuence:PATTern"](#) on page 438
 - [":TRIGger:SEQuence:EDGE"](#) on page 436
 - [":TRIGger:SEQuence:TRIGger"](#) on page 441
 - [":TRIGger:SEQuence:RESet"](#) on page 439

:TRIGger:SEQuence:PAATtern

N (see page 616)

Command Syntax :TRIGger:SEQuence:PAATtern{1 | 2} <value>,<mask>
 <value> ::= integer or <string>
 <mask> ::= integer or <string>
 <string> ::= "0xnmmmmn" where n ::= {0,...,9 | A,...,F}

The :TRIGger:SEQuence:PAATtern<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
4 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 through 19 - analog channels 1 through 4.
2 analog + 16 digital channels (mixed-signal)	Bits 0 through 15 - digital channels 0 through 15. Bits 16 and 17 - analog channels 1 and 2.
4 analog channels only	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels only	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:PAATtern{1 | 2}?

The :TRIGger:SEQuence:PAATtern<n>? query returns the current settings of the specified pattern resource.

Return Format <value>,<mask><NL>

- See Also**
- "Introduction to :TRIGger Commands" on page 358
 - ":TRIGger:SEQuence:FIND" on page 437
 - ":TRIGger:SEQuence:TRIGger" on page 441
 - ":TRIGger:SEQuence:RESet" on page 439

:TRIGger:SEQuence:RESet

N (see [page 616](#))

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 358
 - [":TRIGger:SEQuence:PATTErn"](#) on page 438
 - [":TRIGger:SEQuence:EDGE"](#) on page 436
 - [":TRIGger:SEQuence:TIMer"](#) on page 440
 - [":TRIGger:SEQuence:FIND"](#) on page 437
 - [":TRIGger:SEQuence:TRIGger"](#) on page 441

:TRIGger:SEQuence:TIMer

N (see [page 616](#))

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 358
 - "[:TRIGger:SEQuence:RESet](#)" on page 439

:TRIGger:SEQuence:TRIGger

N (see page 616)

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 358
 - ":TRIGger:SEQuence:PATTern" on page 438
 - ":TRIGger:SEQuence:EDGE" on page 436
 - ":TRIGger:SEQuence:COUNt" on page 435
 - ":TRIGger:SEQuence:FIND" on page 437
 - ":TRIGger:SEQuence:RESet" on page 439
 - ":TRIGger:SEQuence:RESet" on page 439

:TRIGger:SPI Commands

Table 77 :TRIGger:SPI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCK:SL OPe <slope> (see page 443)	:TRIGger:SPI:CLOCK:SL OPe? (see page 443)	<slope> ::= {NEGative POSitive}
:TRIGger:SPI:CLOCK:TI Meout <time_value> (see page 444)	:TRIGger:SPI:CLOCK:TI Meout? (see page 444)	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 445)	:TRIGger:SPI:FRAMing? (see page 445)	<value> ::= {CHIPselect NOTChipselect TIMEout}
:TRIGger:SPI:PATtern: DATA <value>, <mask> (see page 446)	:TRIGger:SPI:PATtern: DATA? (see page 446)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnxxxxxx" where n ::= {0,...,9 A,...,F}
:TRIGger:SPI:PATtern: WIDTh <width> (see page 447)	:TRIGger:SPI:PATtern: WIDTh? (see page 447)	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:C LOCK <source> (see page 448)	:TRIGger:SPI:SOURce:C LOCK? (see page 448)	<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 449)	:TRIGger:SPI:SOURce:D ATA? (see page 449)	<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 450)	:TRIGger:SPI:SOURce:F RAME? (see page 450)	<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 616](#))

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 358
 - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 444
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448

:TRIGger:SPI:CLOCK:TIMEout

N (see [page 616](#))

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 358
 - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 443
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448
 - [":TRIGger:SPI:FRAMing"](#) on page 445

:TRIGger:SPI:FRAMing

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:SPI:CLOCK:TIMEout](#)" on page 444
 - "[:TRIGger:SPI:SOURce:FRAME](#)" on page 450

:TRIGger:SPI:PATtern:DATA

N (see [page 616](#))

Command Syntax :TRIGger:SPI:PATtern:DATA <value>,<mask>
<value> ::= integer or <string>
<mask> ::= integer or <string>
<string> ::= "0xnnnnnn" where n ::= {0,...,9 | A,...,F}

The :TRIGger:SPI:PATtern:DATA command defines the SPI data pattern resource according to the value and the mask. This pattern, along with the data width, control the data pattern searched for in the data stream.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

Query Syntax :TRIGger:SPI:PATtern:DATA?

The :TRIGger:SPI:PATtern:DATA? query returns the current settings of the specified SPI data pattern resource.

Return Format <value> , <mask><NL>

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 447
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449

:TRIGger:SPI:PATtern:WIDTh

N (see [page 616](#))

Command Syntax :TRIGger:SPI:PATtern:WIDTh <width>
 <width> ::= integer from 4 to 32 in NR1 format

The :TRIGger:SPI:PATtern:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 32 bits.

Query Syntax :TRIGger:SPI:PATtern:WIDTh?

The :TRIGger:SPI:PATtern:WIDTh? query returns the current SPI data pattern width setting.

Return Format <width><NL>
 <width> ::= integer from 4 to 32 in NR1 format

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:SPI:PATtern:DATA"](#) on page 446
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449

:TRIGger:SPI:SOURce:CLOCK

N (see [page 616](#))

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 358
 - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 443
 - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 444
 - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 450
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449

:TRIGger:SPI:SOURce:DATA

N (see [page 616](#))

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 358
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448
 - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 450
 - [":TRIGger:SPI:PATtern:DATA"](#) on page 446
 - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 447

:TRIGger:SPI:SOURce:FRAME

N (see [page 616](#))

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 358
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449
 - [":TRIGger:SPI:FRAMing"](#) on page 445

:TRIGger:TV Commands

Table 78 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 452)	:TRIGger:TV:LINE? (see page 452)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 453)	:TRIGger:TV:MODE? (see page 453)	<tv mode> ::= {FIELD1 FIELD2 AFIELDS ALINES LINE VERTICAL LFIELD1 LFIELD2 LALTERNATE LVERTICAL}
:TRIGger:TV:POLarity <polarity> (see page 454)	:TRIGger:TV:POLarity? (see page 454)	<polarity> ::= {POSitive NEGative}
:TRIGger:TV:SOURce <source> (see page 455)	:TRIGger:TV:SOURce? (see page 455)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard> (see page 456)	:TRIGger:TV:STANdard? (see page 456)	<standard> ::= {GENeric NTSC PALM PAL SECam {P480L60HZ P480} {P720L60HZ P720} {P1080L24HZ P1080} P1080L25HZ {I1080L50HZ I1080} I1080L60HZ}

:TRIGger:TV:LINE

N (see page 616)

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 358
 - ":TRIGger:TV:STANdard" on page 456
 - ":TRIGger:TV:MODE" on page 453

:TRIGger:TV:MODE

N (see [page 616](#))

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 358
 - [":TRIGger:TV:STANdard"](#) on page 456
 - [":TRIGger:MODE"](#) on page 364

:TRIGger:TV:POLarity

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:TV:SOURce"](#) on page 455

:TRIGger:TV:SOURce

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:TV:POLarity"](#) on page 454

- Example Code**
- ["Example Code"](#) on page 395

:TRIGger:TV:STANdard

N (see [page 616](#))

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 page 459)	:TRIGger:UART:BAUDrate? (see page 459)	<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 page 460)	:TRIGger:UART:BITorder? (see page 460)	<bitorder> ::= {LSBFirst MSBFirst}
:TRIGger:UART:BURSt <value> (see page 461)	:TRIGger:UART:BURSt? (see page 461)	<value> ::= {OFF 1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see page 462)	:TRIGger:UART:DATA? (see page 462)	<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 463)	:TRIGger:UART:IDLE? (see page 463)	<time_value> ::= time from 10 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 464)	:TRIGger:UART:PARity? (see page 464)	<parity> ::= {EVEN ODD NONE}
:TRIGger:UART:POLarity <polarity> (see page 465)	:TRIGger:UART:POLarity? (see page 465)	<polarity> ::= {HIGH LOW}
:TRIGger:UART:QUALifier <value> (see page 466)	:TRIGger:UART:QUALifier? (see page 466)	<value> ::= {EQUAL NOTequal GREaterthan LESSthan}
:TRIGger:UART:SOURce:RX <source> (see page 467)	:TRIGger:UART:SOURce:RX? (see page 467)	<source> ::= {CHANnel<n> EXTernal} for DSO models <source> ::= {CHANnel<n> DIGital0,...,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

3 Commands by Subsystem

Table 80 :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:SOURce: TX <source> (see page 468)	:TRIGger:UART:SOURce: TX? (see page 468)	<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 469)	:TRIGger:UART:TYPE? (see page 469)	<value> ::= {RSTArt RSTop RDATA RD1 RD0 RDX PARityerror TSTArt TSTOp TDATA TD1 TD0 TDX}
:TRIGger:UART:WIDTh <width> (see page 470)	:TRIGger:UART:WIDTh? (see page 470)	<width> ::= {5 6 7 8 9}

:TRIGger:UART:BAUDrate

N (see [page 616](#))

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 = {600 to 1000000 | 1382000 | 1843000 | 2765000}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:BITorder

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469
 - [":TRIGger:UART:SOURce:RX"](#) on page 467
 - [":TRIGger:UART:SOURce:TX"](#) on page 468

:TRIGger:UART:BURSt

N (see [page 616](#))

Command Syntax :TRIGger:UART:BURSt <value>
 <value> ::= {OFF | 1 to 4096 in NR1 format}

The :TRIGger:UART:BURSt command selects the burst value (Nth occurrence of trigger event 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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:IDLE"](#) on page 463
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:DATA

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:IDLE

N (see [page 616](#))

Command Syntax :TRIGger:UART:IDLE <time_value>

<time_value> ::= time from 10 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 10 us to 10 s in NR3 format

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:UART:BURSt](#)" on page 461
 - "[:TRIGger:UART:TYPE](#)" on page 469

:TRIGger:UART:PARity

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:POLarity

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:QUALifier

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469

:TRIGger:UART:SOURce:RX

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469
 - [":TRIGger:UART:BITorder"](#) on page 460

:TRIGger:UART:SOURce:TX

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:TYPE"](#) on page 469
 - [":TRIGger:UART:BITorder"](#) on page 460

:TRIGger:UART:TYPE

N (see [page 616](#))

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 | RST | RDAT | RD1 | RD0 | RDX | PAR | TSTA |
             TSTO | TDAT | TD1 | TD0 | TDX}
```

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:UART:DATA"](#) on page 462
 - [":TRIGger:UART:QUALifier"](#) on page 466
 - [":TRIGger:UART:WIDTH"](#) on page 470

:TRIGger:UART:WIDTH

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:UART:TYPE](#)" on page 469

:TRIGger:USB Commands

Table 81 :TRIGger:USB Commands Summary

Command	Query	Options and Query Returns
:TRIGger:USB:SOURce:D MINus <source> (see page 472)	:TRIGger:USB:SOURce:D MINus? (see page 472)	<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 473)	:TRIGger:USB:SOURce:D PLus? (see page 473)	<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 474)	:TRIGger:USB:SPEEd? (see page 474)	<value> ::= {LOW FULL}
:TRIGger:USB:TRIGger <value> (see page 475)	:TRIGger:USB:TRIGger? (see page 475)	<value> ::= {SOP EOP ENTersuspend EXITsuspend RESet}

:TRIGger:USB:SOURce:DMINus

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:USB:SOURce:DPLus"](#) on page 473
 - [":TRIGger:USB:TRIGger"](#) on page 475

:TRIGger:USB:SOURce:DPLus

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:USB:SOURce:DMINus"](#) on page 472
 - [":TRIGger:USB:TRIGger"](#) on page 475

:TRIGger:USB:SPEEd

N (see [page 616](#))

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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:USB:SOURce:DMINus](#)" on page 472
 - "[:TRIGger:USB:SOURce:DPLus](#)" on page 473
 - "[:TRIGger:USB:TRIGger](#)" on page 475

:TRIGger:USB:TRIGger

N (see [page 616](#))

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 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:USB:SPEed"](#) on page 474

:WAVEform Commands

Provide access to waveform data. See "Introduction to :WAVEform Commands" on page 478.

Table 82 :WAVEform Commands Summary

Command	Query	Options and Query Returns
:WAVEform:BYTeorder <value> (see page 484)	:WAVEform:BYTeorder? (see page 484)	<value> ::= {LSBFirst MSBFirst}
n/a	:WAVEform:COUNT? (see page 485)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVEform:DATA? (see page 486)	<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 488)	:WAVEform:FORMat? (see page 488)	<value> ::= {WORD BYTE ASCII}
:WAVEform:POINTs <# points> (see page 489)	:WAVEform:POINTs? (see page 489)	<# points> ::= {100 250 500 1000 <points_mode>} if waveform points mode is NORMAl <# points> ::= {100 250 500 1000 2000 ... 8000000 in 1-2-5 sequence <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMAl MAXimum RAW}
:WAVEform:POINTs:MODE <points_mode> (see page 491)	:WAVEform:POINTs:MODE ? (see page 492)	<points_mode> ::= {NORMAl MAXimum RAW}

Table 82 :WAVEform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVEform:PREAmble? (see page 493)	<p><preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1></p> <p><format> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for BYTE format • 1 for WORD format • 2 for ASCII format <p><type> ::= an integer in NR1 format:</p> <ul style="list-style-type: none"> • 0 for NORMAL type • 1 for PEAK detect type • 2 for AVERAGE type • 3 for HRESolution type <p><count> ::= Average count, or 1 if PEAK detect type or NORMAL; an integer in NR1 format</p>
n/a	:WAVEform:SEGmented:COUNT? (see page 496)	<count> ::= an integer from 2 to 2000 in NR1 format (with Option SGM)
n/a	:WAVEform:SEGmented:TAG? (see page 497)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVEform:SOURce <source> (see page 498)	:WAVEform:SOURce? (see page 498)	<p><source> ::= {CHANnel<n> FUNCTION MATH SBUS} for DSO models</p> <p><source> ::= {CHANnel<n> POD{1 2} BUS{1 2} FUNCTION MATH SBUS} for MSO models</p> <p><n> ::= 1-2 or 1-4 in NR1 format</p>
:WAVEform:SOURce:SUBSource <subsource> (see page 502)	:WAVEform:SOURce:SUBSource? (see page 502)	<subsource> ::= {NONE RX TX}
n/a	:WAVEform:TYPE? (see page 503)	<return_mode> ::= {NORM PEAK AVER HRES}
:WAVEform:UNSigned {{0 OFF} {1 ON}} (see page 504)	:WAVEform:UNSigned? (see page 504)	{0 1}
:WAVEform:VIEW <view> (see page 505)	:WAVEform:VIEW? (see page 505)	<view> ::= {MAIN}

Table 82 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:XINCrement? (see page 506)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 507)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 508)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 509)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 510)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 511)	<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 486](#)) and :WAVeform:PREamble (see [page 493](#)). 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 145](#)): NORMal, AVERage, PEAK, and HRESolution. Digital channels are always

acquired using NORMal. When the data is acquired using the :DIGitize command (see [page 105](#)) or :RUN command (see [page 125](#)), 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 GPIB 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 486](#)) 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 139](#)).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVEform:POINTS command (see [page 489](#)). 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 GPIB 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 489](#)). 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 136](#)). 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 489](#)). 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 489](#)). 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 145](#)), the value returned by the :WAVEFORM:XINCrement query (see [page 506](#)) 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. This mode is the same as the AVERage mode with :ACQUIRE:COUNT 1.

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 488](#)), 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 145](#)), 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 488](#)). 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 504](#)) 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 488](#)) provides access to the waveform data as real Y-axis values without using Y origin, Y reference, and Y increment to convert the binary data. Values are transferred as ASCII digits in floating point format separated by commas. In ASCII format, holes are represented by the value $9.9e+37$. The setting of `:WAVEform:BYTeorder` (see [page 484](#)) and `:WAVEform:UNSIGNED` (see [page 504](#)) have no effect when the format is ASCII.

Data Format for Transfer - WORD format

WORD format (see `:WAVEform:FORMat` on [page 488](#)) 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 489](#)). 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 484](#)) 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 488](#)) 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 GPIB 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 484](#)) 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 504](#)) 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 488](#)) is WORD, every other data byte will be 0. The setting of :WAVeform:BYTeorder (see [page 484](#)) 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 147](#)) are used to select the digital channels for a bus.

Reporting the Setup

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

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS
NONE
```

:WAVeform:BYTeorder

C (see [page 616](#))

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 478
 - [":WAVeform:DATA"](#) on page 486
 - [":WAVeform:FORMat"](#) on page 488
 - [":WAVeform:PREamble"](#) on page 493

- Example Code**
- ["Example Code"](#) on page 499
 - ["Example Code"](#) on page 494

:WAVeform:COUNT

C (see [page 616](#))

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 478
 - [":ACQUIRE:COUNT"](#) on page 136
 - [":ACQUIRE:TYPE"](#) on page 145

:WAVeform:DATA

C (see [page 616](#))

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 478
 - [":WAVeform:UNSigned"](#) on page 504
 - [":WAVeform:BYTeorder"](#) on page 484
 - [":WAVeform:FORMat"](#) on page 488
 - [":WAVeform:POINts"](#) on page 489
 - [":WAVeform:PREamble"](#) on page 493
 - [":WAVeform:SOURce"](#) on page 498
 - [":WAVeform:TYPE"](#) on page 503

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 6"](#)
on page 692

:WAVeform:FORMat

C (see [page 616](#))

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 478
 - [":WAVeform:BYTeorder"](#) on page 484
 - [":WAVeform:SOURce"](#) on page 498
 - [":WAVeform:DATA"](#) on page 486
 - [":WAVeform:PREamble"](#) on page 493

Example Code • ["Example Code"](#) on page 499

:WAVeform:POINts

C (see [page 616](#))

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 491](#)) 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 491](#)) 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 478
- [":ACQUIRE:POINTS"](#) on page 139
- [":WAVEform:DATA"](#) on page 486
- [":WAVEform:SOURce"](#) on page 498
- [":WAVEform:VIEW"](#) on page 505
- [":WAVEform:PREAmble"](#) on page 493
- [":WAVEform:POINTS:MODE"](#) on page 491

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 6"](#) on page 692

:WAVeform:POINts:MODE

N (see [page 616](#))

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 129](#)) or the :DIGitize command (see [page 105](#)) 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.

3 Commands by Subsystem

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 478
 - [":ACQuire:POINts"](#) on page 139
 - [":WAVeform:DATA"](#) on page 486
 - [":WAVeform:VIEW"](#) on page 505
 - [":WAVeform:PREAmble"](#) on page 493
 - [":WAVeform:POINts"](#) on page 489
 - [":TIMEbase:MODE"](#) on page 348
 - [":ACQuire:TYPE"](#) on page 145
 - [":ACQuire:COUNt"](#) on page 136

:WAVeform:PREamble

C (see [page 616](#))

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>

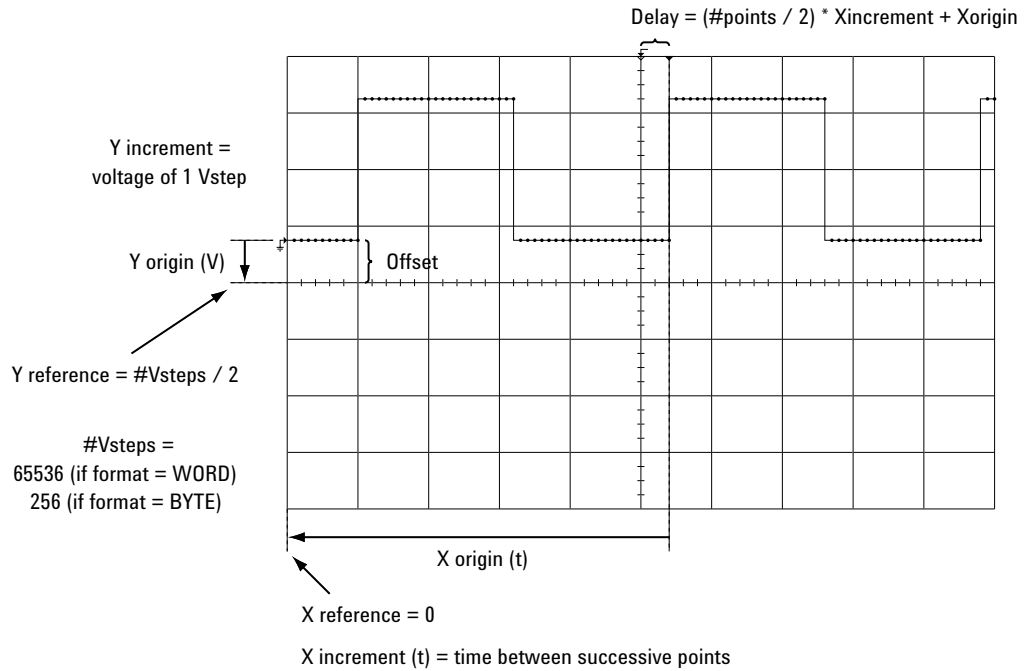
```
<preamble_block> ::= <format 16-bit NR1>,
                    <type 16-bit NR1>,
                    <points 32-bit NR1>,
                    <count 32-bit NR1>,
                    <xincrement 64-bit floating point NR3>,
                    <xorigin 64-bit floating point NR3>,
                    <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, 2 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).

3 Commands by Subsystem



- See Also**
- ["Introduction to :WAVEform Commands"](#) on page 478
 - [":ACQUIRE:COUNT"](#) on page 136
 - [":ACQUIRE:POINTS"](#) on page 139
 - [":ACQUIRE:TYPE"](#) on page 145
 - [":DIGitize"](#) on page 105
 - [":WAVEform:COUNT"](#) on page 485
 - [":WAVEform:DATA"](#) on page 486
 - [":WAVEform:FORMat"](#) on page 488
 - [":WAVEform:POINTS"](#) on page 489
 - [":WAVEform:TYPE"](#) on page 503
 - [":WAVEform:XINCrement"](#) on page 506
 - [":WAVEform:XORigin"](#) on page 507
 - [":WAVEform:XREFerence"](#) on page 508
 - [":WAVEform:YINCrement"](#) on page 509
 - [":WAVEform:YORigin"](#) on page 510
 - [":WAVEform:YREFerence"](#) on page 511

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, 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)

```

Example program from the start: ["VISA COM Example in Visual Basic 6"](#)
on page 692

:WAVeform:SEGmented:COUNT

N (see [page 616](#))

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 138
 - [":ACQUIRE:SEGmented:COUNT"](#) on page 141
 - [":DIGitize"](#) on page 105
 - ["Introduction to :WAVeform Commands"](#) on page 478

Example Code • ["Example Code"](#) on page 142

:WAVeform:SEGmented:TTAG**N** (see [page 616](#))**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 142
 - ["Introduction to :WAVeform Commands"](#) on page 478

Example Code • ["Example Code"](#) on page 142

:WAVeform:SOURce

C (see [page 616](#))

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 488).

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 478](#)
 - [":DIGitize" on page 105](#)
 - [":WAVEform:FORMat" on page 488](#)
 - [":WAVEform:BYTeorder" on page 484](#)
 - [":WAVEform:DATA" on page 486](#)
 - [":WAVEform:PREamble" on page 493](#)

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.

```

3 Commands by Subsystem

```
' 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 6"](#)
on page 692

:WAVeform:SOURce:SUBSource

C (see [page 616](#))

Command Syntax :WAVeform:SOURce:SUBSource <subsource>

<subsource> ::= {{NONE | RX} | TX}

If the :WAVeform:SOURce is SBUS (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

Currently, only UART serial decode lets you get "TX" data. The default, NONE, specifies "RX" data. (RX is an alias for NONE.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS:MODE is not UART, the only valid subsource is NONE.

Query Syntax :WAVeform:SOURce:SUBSource?

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

Return Format <subsource><NL>

<subsource> ::= {NONE | TX}

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 478
 - [":WAVeform:SOURce"](#) on page 498

:WAVeform:TYPE

C (see [page 616](#))

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 478
 - [":ACQUIRE:TYPE"](#) on page 145
 - [":WAVeform:DATA"](#) on page 486
 - [":WAVeform:PREamble"](#) on page 493
 - [":WAVeform:SOURce"](#) on page 498

:WAVeform:UNSigned

C (see [page 616](#))

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 478
 - [":WAVeform:SOURce"](#) on page 498

:WAVeform:VIEW

C (see [page 616](#))

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 478
 - "[:WAVeform:POINTS](#)" on page 489

:WAVeform:XINCrement

C (see [page 616](#))

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 478
 - "[:WAVeform:PREamble](#)" on page 493

Example Code

- "[Example Code](#)" on page 494

:WAVeform:XORigin

C (see [page 616](#))

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 478
 - [":WAVeform:PREamble"](#) on page 493
 - [":WAVeform:XREFerence"](#) on page 508

- Example Code**
- ["Example Code"](#) on page 494

:WAVeform:XREFerence

C (see [page 616](#))

Query Syntax :WAVeform:XREFerence?

The :WAVeform:XREFerence? query returns the x-reference value for the currently specified source. This value specifies the index of the data point associated with the x-origin data value. In this product, the x-reference point is the first point displayed and XREFerence is always 0.

Return Format <value><NL>

<value> ::= x-reference value = 0 in 32-bit NR1 format

- See Also**
- "[Introduction to :WAVeform Commands](#)" on page 478
 - "[:WAVeform:PREamble](#)" on page 493
 - "[:WAVeform:XORigin](#)" on page 507

Example Code • "[Example Code](#)" on page 494

:WAVeform:YINCrement

C (see [page 616](#))

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 478
 - [":WAVeform:PREamble"](#) on page 493

Example Code

- ["Example Code"](#) on page 494

:WAVeform:YORigin

C (see [page 616](#))

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 478
 - "[:WAVeform:PREamble](#)" on page 493
 - "[:WAVeform:YREFerence](#)" on page 511

- Example Code**
- "[Example Code](#)" on page 494

:WAVeform:YREFerence

C (see [page 616](#))

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.

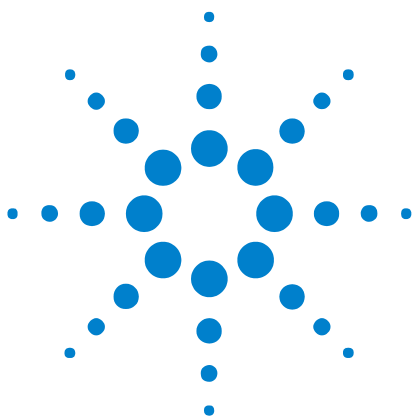
Return Format <value><NL>

<value> ::= y-reference value in the current preamble in 32-bit NR1 format

- See Also**
- ["Introduction to :WAVeform Commands"](#) on page 478
 - [":WAVeform:PREamble"](#) on page 493
 - [":WAVeform:YORigin"](#) on page 510

Example Code • ["Example Code"](#) on page 494

3 Commands by Subsystem



4 Commands A-Z

A	513
B	514
C	515
D	517
E	519
F	519
G	521
H	521
I	521
L	522
M	523
N	525
O	525
P	525
Q	527
R	527
S	528
T	531
U	536
V	537
W	537
X	538
Y	538

- A**
- AALias, `":ACquire:AALias"` on page 134
 - ACKnowledge, `":TRIGger:CAN:ACKnowledge"` on page 580
 - `":ACquire:AALias"` on page 134
 - `":ACquire:COMplete"` on page 135
 - `":ACquire:COUNt"` on page 136
 - `":ACquire:DAALias"` on page 137



- ":ACQUIRE:MODE" on page 138
- ":ACQUIRE:POINTS" on page 139
- ":ACQUIRE:RSIGNAL" on page 140
- ":ACQUIRE:SEGMENTED:COUNT" on page 141
- ":ACQUIRE:SEGMENTED:INDEX" on page 142
- ":ACQUIRE:SRATE" on page 144
- ":ACQUIRE:TYPE" on page 145
- ":ACTIVITY" on page 97
- ADDRESS Commands:
 - ":SBUS:BUSDOCTOR:ADDRESS" on page 314
 - ":TRIGGER:IIC:PATTERN:ADDRESS" on page 418
- ":AER (Arm Event Register)" on page 98
- APrinter, ":HARDcopy:APRinter" on page 225
- AREA Commands:
 - ":HARDcopy:AREA" on page 224
 - ":SAVE:IMAGe:AREA" on page 301
- ASIZE, ":SBUS:IIC:ASIZE" on page 328
- ":AUToscale" on page 99
 - ":AUToscale:AMODE" on page 101
 - ":AUToscale:CHANnels" on page 102
- B** • BASE, ":SBUS:UART:BASE" on page 332
- BAUDrate Commands:
 - ":SBUS:BUSDOCTOR:BAUDrate" on page 315
 - ":TRIGGER:CAN:SIGNal:BAUDrate" on page 376
 - ":TRIGGER:LIN:SIGNal:BAUDrate" on page 429
 - ":TRIGGER:UART:BAUDrate" on page 459
- BIT<m>, ":BUS<n>:BIT<m>" on page 149
- BITorder, ":TRIGGER:UART:BITorder" on page 460
- BITS, ":BUS<n>:BITS" on page 150
- ":BLANK" on page 103
- BURSt, ":TRIGGER:UART:BURSt" on page 461
- ":BUS<n>:BIT<m>" on page 149
- ":BUS<n>:BITS" on page 150
- ":BUS<n>:CLEar" on page 152
- ":BUS<n>:DISPlay" on page 153

- ":BUS<n>:LABel" on page 154
 - ":BUS<n>:MASK" on page 155
 - BUSDoctor Commands:
 - ":SBUS:BUSDoctor:ADDResS" on page 314
 - ":SBUS:BUSDoctor:BAUDrate" on page 315
 - ":SBUS:BUSDoctor:CHANnel" on page 316
 - ":SBUS:BUSDoctor:MODE" on page 317
 - BWLimit Commands:
 - ":CHANnel<n>:BWLimit" on page 167
 - ":EXternal:BWLimit" on page 202
 - BYTeorder, ":WAVEform:BYTeorder" on page 484
- C**
- ":CALibrate:DATE" on page 157
 - ":CALibrate:LABel" on page 158
 - ":CALibrate:STARt" on page 159
 - ":CALibrate:STATus" on page 160
 - ":CALibrate:SWITCh" on page 161
 - ":CALibrate:TEMPerature" on page 162
 - ":CALibrate:TIME" on page 163
 - CAN Commands:
 - ":SBUS:CAN:COUNT:ERRor" on page 318
 - ":SBUS:CAN:COUNT:OVERload" on page 319
 - ":SBUS:CAN:COUNT:RESet" on page 320
 - ":SBUS:CAN:COUNT:TOTal" on page 321
 - ":SBUS:CAN:COUNT:UTILization" on page 322
 - ":TRIGger:CAN Commands" on page 369
 - CBASe, ":TRIGger:FLEXray:TIME:CBASe" on page 403
 - CCBASe, ":TRIGger:FLEXray:FRAME:CCBase" on page 399
 - CCRepetition, ":TRIGger:FLEXray:FRAME:CCRepetition" on page 400
 - CRepetition, ":TRIGger:FLEXray:TIME:CREPetition" on page 404
 - ":CDISplay" on page 104
 - CENTer, ":FUNCTion:CENTer" on page 212
 - CHANnel, ":SBUS:BUSDoctor:CHANnel" on page 316
 - ":CHANnel:ACTivity" on page 544
 - ":CHANnel:LABel" on page 545
 - ":CHANnel:THReshold" on page 546

- [":CHANnel2:SKEW"](#) on page 547
- [":CHANnel<n>:BWLimit"](#) on page 167
- [":CHANnel<n>:COUPling"](#) on page 168
- [":CHANnel<n>:DISPlay"](#) on page 169
- [":CHANnel<n>:IMPedance"](#) on page 170
- [":CHANnel<n>:INPut"](#) on page 548
- [":CHANnel<n>:INVert"](#) on page 171
- [":CHANnel<n>:LABel"](#) on page 172
- [":CHANnel<n>:OFFSet"](#) on page 173
- [":CHANnel<n>:PMODE"](#) on page 549
- [":CHANnel<n>:PROBe"](#) on page 174
- [":CHANnel<n>:PROBe:ID"](#) on page 175
- [":CHANnel<n>:PROBe:SKEW"](#) on page 176
- [":CHANnel<n>:PROBe:STYPe"](#) on page 177
- [":CHANnel<n>:PROTection"](#) on page 178
- [":CHANnel<n>:RANGe"](#) on page 179
- [":CHANnel<n>:SCALe"](#) on page 180
- [":CHANnel<n>:UNITs"](#) on page 181
- [":CHANnel<n>:VERNier"](#) on page 182
- **CLEar Commands:**
 - [":BUS<n>:CLEar"](#) on page 152
 - [":DISPlay:CLEar"](#) on page 192
 - [":MEASure:CLEar"](#) on page 250
- **CLOCK Commands:**
 - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 421
 - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 443
 - [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 444
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448
- ["*CLS \(Clear Status\)"](#) on page 73
- **COMplete**, [":ACQuire:COMplete"](#) on page 135
- **CONNect**, [":DISPlay:CONNect"](#) on page 550
- **COUNt Commands:**
 - [":ACQuire:COUNt"](#) on page 136
 - [":ACQuire:SEGMENTed:COUNt"](#) on page 141
 - [":SBUS:CAN:COUNt:ERRor"](#) on page 318

- [":SBUS:CAN:COUNt:OVERload"](#) on page 319
- [":SBUS:CAN:COUNt:RESet"](#) on page 320
- [":SBUS:CAN:COUNt:TOTal"](#) on page 321
- [":SBUS:CAN:COUNt:UTILization"](#) on page 322
- [":SBUS:FLEXray:COUNt:NULL"](#) on page 324
- [":SBUS:FLEXray:COUNt:RESet"](#) on page 325
- [":SBUS:FLEXray:COUNt:SYNC"](#) on page 326
- [":SBUS:FLEXray:COUNt:TOTal"](#) on page 327
- [":SBUS:UART:COUNt:ERRor"](#) on page 333
- [":SBUS:UART:COUNt:RESet"](#) on page 334
- [":SBUS:UART:COUNt:RXFRames"](#) on page 335
- [":SBUS:UART:COUNt:TXFRames"](#) on page 336
- [":TRIGger:EBURst:COUNt"](#) on page 387
- [":WAVEform:COUNt"](#) on page 485
- [":WAVEform:SEGmented:COUNt"](#) on page 496
- COUNter, [":MEASure:COUNter"](#) on page 251
- COUPling Commands:
 - [":CHANnel<n>:COUPling"](#) on page 168
 - [":TRIGger\[:EDGE\]:COUPling"](#) on page 391
- D**
 - DAALias, [":ACQuire:DAALias"](#) on page 137
 - DATA Commands:
 - [":DISPlay:DATA"](#) on page 193
 - [":TRIGger:CAN:PATtern:DATA"](#) on page 371
 - [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 372
 - [":TRIGger:IIC:PATtern:DATA"](#) on page 419
 - [":TRIGger:IIC:PATtern:DATA2"](#) on page 420
 - [":TRIGger:IIC:SOURce:DATA"](#) on page 422
 - [":TRIGger:SPI:PATtern:DATA"](#) on page 446
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449
 - [":TRIGger:UART:DATA"](#) on page 462
 - [":WAVEform:DATA"](#) on page 486
 - DATE Commands:
 - [":CALibrate:DATE"](#) on page 157
 - [":SYSTem:DATE"](#) on page 339
 - DEFine, [":MEASure:DEFine"](#) on page 252

- DEFinition Commands:
 - [":TRIGger:CAN:SIGNal:DEFinition"](#) on page 581
 - [":TRIGger:LIN:SIGNal:DEFinition"](#) on page 582
- DELay Commands:
 - [":MEASure:DELay"](#) on page 255
 - [":TIMEbase:DELay"](#) on page 579
- DESTination, [":HARDcopy:DESTination"](#) on page 556
- DEvice, [":HARDcopy:DEvice"](#) on page 557
- [":DIGital<n>:DISPlay"](#) on page 185
- [":DIGital<n>:LABel"](#) on page 186
- [":DIGital<n>:POSition"](#) on page 187
- [":DIGital<n>:SIZE"](#) on page 188
- [":DIGital<n>:THReshold"](#) on page 189
- [":DIGitize"](#) on page 105
- DISPlay Commands:
 - [":BUS<n>:DISPlay"](#) on page 153
 - [":CHANnel<n>:DISPlay"](#) on page 169
 - [":DIGital<n>:DISPlay"](#) on page 185
 - [":FUNction:DISPlay"](#) on page 213
 - [":POD<n>:DISPlay"](#) on page 288
 - [":SBUS:DISPlay"](#) on page 323
- [":DISPlay:CLEar"](#) on page 192
- [":DISPlay:CONNect"](#) on page 550
- [":DISPlay:DATA"](#) on page 193
- [":DISPlay:LABel"](#) on page 195
- [":DISPlay:LABList"](#) on page 196
- [":DISPlay:ORDer"](#) on page 551
- [":DISPlay:PERsistence"](#) on page 197
- [":DISPlay:SOURce"](#) on page 198
- [":DISPlay:VECTors"](#) on page 199
- DMINus, [":TRIGger:USB:SOURce:DMINus"](#) on page 472
- DPLus, [":TRIGger:USB:SOURce:DPLus"](#) on page 473
- DSP, [":SYSTem:DSP"](#) on page 340
- DURation, [":TRIGger:DURation Commands"](#) on page 380
- DUTYcycle, [":MEASure:DUTYcycle"](#) on page 257

- E**
 - EBURst, ":TRIGger:EBURst Commands" on page 386
 - EDGE, ":TRIGger[:EDGE] Commands" on page 390
 - ":ERASe" on page 552
 - ERRor Commands:
 - ":SBUS:CAN:COUNt:ERRor" on page 318
 - ":SBUS:UART:COUNt:ERRor" on page 333
 - ":SYSTem:ERRor" on page 341
 - ":TRIGger:FLEXray:ERRor:TYPE" on page 397
 - "*ESE (Standard Event Status Enable)" on page 74
 - "*ESR (Standard Event Status Register)" on page 76
 - ":EXTernal:BWLimit" on page 202
 - ":EXTernal:IMPedance" on page 203
 - ":EXTernal:INPut" on page 553
 - ":EXTernal:PMODE" on page 554
 - ":EXTernal:PROBe" on page 204
 - ":EXTernal:PROBe:ID" on page 205
 - ":EXTernal:PROBe:STYPe" on page 206
 - ":EXTernal:PROTection" on page 207
 - ":EXTernal:RANGe" on page 208
 - ":EXTernal:UNITs" on page 209
- F**
 - FACTors Commands:
 - ":HARDcopy:FACTors" on page 226
 - ":SAVE:IMAGe:FACTors" on page 302
 - FALLtime, ":MEASure:FALLtime" on page 258
 - FFEed, ":HARDcopy:FFEed" on page 227
 - FILEname Commands:
 - ":HARDcopy:FILEname" on page 558
 - ":RECall:FILEname" on page 293
 - ":SAVE:FILEname" on page 299
 - FIND, ":TRIGger:SEQuence:FIND" on page 437
 - FLEXray Commands:
 - ":SBUS:FLEXray:COUNt:NULL" on page 324
 - ":SBUS:FLEXray:COUNt:RESet" on page 325
 - ":SBUS:FLEXray:COUNt:SYNC" on page 326
 - ":SBUS:FLEXray:COUNt:TOTal" on page 327

- [":TRIGger:FLEXray:ERRor:TYPE"](#) on page 397
- [":TRIGger:FLEXray:FRAMe:CCBase"](#) on page 399
- [":TRIGger:FLEXray:FRAMe:CCRepetition"](#) on page 400
- [":TRIGger:FLEXray:FRAMe:ID"](#) on page 401
- [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 402
- [":TRIGger:FLEXray:TIME:CBASe"](#) on page 403
- [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 404
- [":TRIGger:FLEXray:TIME:SEGMENT"](#) on page 405
- [":TRIGger:FLEXray:TIME:SLOT"](#) on page 406
- [":TRIGger:FLEXray:TRIGger"](#) on page 407
- **FORMat Commands:**
 - [":HARDcopy:FORMat"](#) on page 559
 - [":SAVE:IMAGe:FORMat"](#) on page 303
 - [":SAVE:WAVEform:FORMat"](#) on page 309
 - [":WAVEform:FORMat"](#) on page 488
- **FRAMe Commands:**
 - [":TRIGger:FLEXray:FRAMe:CCBase"](#) on page 399
 - [":TRIGger:FLEXray:FRAMe:CCRepetition"](#) on page 400
 - [":TRIGger:FLEXray:FRAMe:ID"](#) on page 401
 - [":TRIGger:FLEXray:FRAMe:TYPE"](#) on page 402
 - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 450
- **FRAMing Commands:**
 - [":SBUS:UART:FRAMing"](#) on page 337
 - [":TRIGger:SPI:FRAMing"](#) on page 445
- **FREQuency, ":MEASure:FREQuency"** on page 259
- [":FUNCTion:CENTer"](#) on page 212
- [":FUNCTion:DISPlay"](#) on page 213
- [":FUNCTion:OFFSet"](#) on page 214
- [":FUNCTion:OPERation"](#) on page 215
- [":FUNCTion:RANGe"](#) on page 216
- [":FUNCTion:REFerence"](#) on page 217
- [":FUNCTion:SCALE"](#) on page 218
- [":FUNCTion:SOURce"](#) on page 219
- [":FUNCTion:SPAN"](#) on page 220
- [":FUNCTion:VIEW"](#) on page 555

- [":FUNCTION:WINDOW"](#) on page 221
- G**
 - [GLITCh \(Pulse Width\), ":TRIGGER:GLITCh Commands"](#) on page 408
 - [GRAYscale, ":HARDcopy:GRAYscale"](#) on page 560
 - GREATERthan Commands:
 - [":TRIGGER:DURATION:GREATERthan"](#) on page 381
 - [":TRIGGER:GLITCh:GREATERthan"](#) on page 410
- H**
 - [":HARDcopy:AREA"](#) on page 224
 - [":HARDcopy:APRinter"](#) on page 225
 - [":HARDcopy:DESTination"](#) on page 556
 - [":HARDcopy:DEVIce"](#) on page 557
 - [":HARDcopy:FACTors"](#) on page 226
 - [":HARDcopy:FFEed"](#) on page 227
 - [":HARDcopy:FILEname"](#) on page 558
 - [":HARDcopy:FORMat"](#) on page 559
 - [":HARDcopy:GRAYscale"](#) on page 560
 - [":HARDcopy:IGColors"](#) on page 561
 - [":HARDcopy:INKSaver"](#) on page 228
 - [":HARDcopy:PALette"](#) on page 229
 - [":HARDcopy:PDRiver"](#) on page 562
 - [":HARDcopy:PRinter:LIST"](#) on page 230
 - [":HARDcopy:START"](#) on page 231
 - [HFReject, ":TRIGGER:HFReject"](#) on page 362
 - [HOLDoff, ":TRIGGER:HOLDoff"](#) on page 363
- I**
 - ID Commands:
 - [":TRIGGER:CAN:PATTERN:ID"](#) on page 373
 - [":TRIGGER:CAN:PATTERN:ID:MODE"](#) on page 374
 - [":TRIGGER:FLEXray:FRAME:ID"](#) on page 401
 - IDLE Commands:
 - [":TRIGGER:EBURst:IDLE"](#) on page 388
 - [":TRIGGER:UART:IDLE"](#) on page 463
 - ["*IDN \(Identification Number\)"](#) on page 78
 - IIC Commands:
 - [":SBUS:IIC:ASIZE"](#) on page 328
 - [":TRIGGER:IIC Commands"](#) on page 417

- IGCOLORS Commands:
 - [":HARDcopy:IGColors"](#) on page 561
 - [":SAVE:IMAGe:INKSaver"](#) on page 304
- IMAGe Commands:
 - [":RECall:IMAGe\[:START\]"](#) on page 294
 - [":SAVE:IMAGe:AREA"](#) on page 301
 - [":SAVE:IMAGe:FACTors"](#) on page 302
 - [":SAVE:IMAGe:FORMat"](#) on page 303
 - [":SAVE:IMAGe:INKSaver"](#) on page 304
 - [":SAVE:IMAGe:PALette"](#) on page 305
 - [":SAVE:IMAGe\[:START\]"](#) on page 300
- IMPedance Commands:
 - [":CHANnel<n>:IMPedance"](#) on page 170
 - [":EXTernal:IMPedance"](#) on page 203
- INDEX, [":ACQUIRE:SEGmented:INDEX"](#) on page 142
- INKSaver, [":HARDcopy:INKSaver"](#) on page 228
- INVert, [":CHANnel<n>:INVert"](#) on page 171
- L**
 - LABEL Commands:
 - [":BUS<n>:LABEL"](#) on page 154
 - [":CALibrate:LABEL"](#) on page 158
 - [":CHANnel:LABEL"](#) on page 545
 - [":CHANnel<n>:LABEL"](#) on page 172
 - [":DIGital<n>:LABEL"](#) on page 186
 - [":DISPlay:LABEL"](#) on page 195
 - LABList, [":DISPlay:LABList"](#) on page 196
 - LENGth Commands:
 - [":SAVE:WAVEform:LENGth"](#) on page 310
 - [":TRIGger:CAN:PATtern:DATA:LENGth"](#) on page 372
 - LESSthan Commands:
 - [":TRIGger:DURation:LESSthan"](#) on page 382
 - [":TRIGger:GLITCh:LESSthan"](#) on page 411
 - LEVel Commands:
 - [":TRIGger\[:EDGE\]:LEVel"](#) on page 392
 - [":TRIGger:GLITCh:LEVel"](#) on page 412
 - LIN Commands:

- [":SBUS:LIN:PARity"](#) on page 329
 - [":TRIGger:LIN Commands"](#) on page 426
 - [LINE, ":TRIGger:TV:LINE"](#) on page 452
 - [LIST, ":HARDcopy:Printer:LIST"](#) on page 230
 - [LOCK, ":SYSTem:LOCK"](#) on page 342
 - [LOWer, ":MEASure:LOWer"](#) on page 563
 - ["*LRN \(Learn Device Setup\)"](#) on page 79
- M**
- [":MARKer:MODE"](#) on page 234
 - [":MARKer:X1Position"](#) on page 235
 - [":MARKer:X1Y1source"](#) on page 236
 - [":MARKer:X2Position"](#) on page 237
 - [":MARKer:X2Y2source"](#) on page 238
 - [":MARKer:XDELta"](#) on page 239
 - [":MARKer:Y1Position"](#) on page 240
 - [":MARKer:Y2Position"](#) on page 241
 - [":MARKer:YDELta"](#) on page 242
 - [MASK, ":BUS<n>:MASK"](#) on page 155
 - [":MEASure:CLEar"](#) on page 250
 - [":MEASure:COUNter"](#) on page 251
 - [":MEASure:DEFine"](#) on page 252
 - [":MEASure:DELay"](#) on page 255
 - [":MEASure:DUTYcycle"](#) on page 257
 - [":MEASure:FALLtime"](#) on page 258
 - [":MEASure:FREQuency"](#) on page 259
 - [":MEASure:LOWer"](#) on page 563
 - [":MEASure:NWIDth"](#) on page 260
 - [":MEASure:OVERshoot"](#) on page 261
 - [":MEASure:PERiod"](#) on page 263
 - [":MEASure:PHASe"](#) on page 264
 - [":MEASure:PREShoot"](#) on page 265
 - [":MEASure:PWIDth"](#) on page 266
 - [":MEASure:RISetime"](#) on page 267
 - [":MEASure:SCRatch"](#) on page 564
 - [":MEASure:SDEVIation"](#) on page 268
 - [":MEASure:SHOW"](#) on page 269

- `":MEASure:SOURce"` on page 270
- `":MEASure:TDELta"` on page 565
- `":MEASure:TEDGe"` on page 272
- `":MEASure:THResholds"` on page 566
- `":MEASure:TMAX"` on page 567
- `":MEASure:TMIN"` on page 568
- `":MEASure:TSTArt"` on page 569
- `":MEASure:TSTOp"` on page 570
- `":MEASure:TVALue"` on page 274
- `":MEASure:TVOLt"` on page 571
- `":MEASure:UPPer"` on page 573
- `":MEASure:VAMPLitude"` on page 276
- `":MEASure:VAVerage"` on page 277
- `":MEASure:VBASe"` on page 278
- `":MEASure:VDELta"` on page 574
- `":MEASure:VMAX"` on page 279
- `":MEASure:VMIN"` on page 280
- `":MEASure:VPP"` on page 281
- `":MEASure:VRMS"` on page 282
- `":MEASure:VSTArt"` on page 575
- `":MEASure:VSTOp"` on page 576
- `":MEASure:VTIME"` on page 283
- `":MEASure:VTOp"` on page 284
- `":MEASure:XMAX"` on page 285
- `":MEASure:XMIN"` on page 286
- `":MERGe"` on page 113
- MODE Commands:
 - `":ACQuire:MODE"` on page 138
 - `":MARKer:MODE"` on page 234
 - `":SBUS:BUSDoctor:MODE"` on page 317
 - `":SBUS:MODE"` on page 330
 - `":TIMEbase:MODE"` on page 348
 - `":TRIGger:CAN:PATtern:ID:MODE"` on page 374
 - `":TRIGger:MODE"` on page 364
 - `":TRIGger:TV:MODE"` on page 453

- ":WAVeform:POINts:MODE" on page 491
- N**
 - NREJect, ":TRIGger:NREJect" on page 365
 - NULL, ":SBUS:FLEXray:COUNt:NULL" on page 324
 - NWIDth, ":MEASure:NWIDth" on page 260
- O**
 - OFFSet Commands:
 - ":CHANnel<n>:OFFSet" on page 173
 - ":FUNction:OFFSet" on page 214
 - "*OPC (Operation Complete)" on page 80
 - ":OPEE (Operation Status Enable Register)" on page 114
 - OPERation, ":FUNction:OPERation" on page 215
 - ":OPERegister:CONDition (Operation Status Condition Register)" on page 116
 - ":OPERegister[:EVENT] (Operation Status Event Register)" on page 118
 - "*OPT (Option Identification)" on page 81
 - ORDER, ":DISPlay:ORDER" on page 551
 - OVERload, ":SBUS:CAN:COUNt:OVERload" on page 319
 - OVERshoot, ":MEASure:OVERshoot" on page 261
 - ":OVLenable (Overload Event Enable Register)" on page 120
 - ":OVLRegister (Overload Event Register)" on page 122
- P**
 - PALette Commands:
 - ":HARDcopy:PALette" on page 229
 - ":SAVE:IMAGe:PALette" on page 305
 - PARity Commands:
 - ":SBUS:LIN:PARity" on page 329
 - ":TRIGger:UART:PARity" on page 464
 - PATTern Commands:
 - ":TRIGger:CAN:PATTern:DATA" on page 371
 - ":TRIGger:CAN:PATTern:DATA:LENGth" on page 372
 - ":TRIGger:CAN:PATTern:ID" on page 373
 - ":TRIGger:CAN:PATTern:ID:MODE" on page 374
 - ":TRIGger:DURation:PATTern" on page 383
 - ":TRIGger:IIC:PATTern:ADDReSS" on page 418
 - ":TRIGger:IIC:PATTern:DATA" on page 419
 - ":TRIGger:IIC:PATTern:DATA2" on page 420

- [":TRIGger:PATtern"](#) on page 366
- [":TRIGger:SEQuence:PATtern"](#) on page 438
- [":TRIGger:SPI:PATtern:DATA"](#) on page 446
- [":TRIGger:SPI:PATtern:WIDTh"](#) on page 447
- PDRiver, [":HARDcopy:PDRiver"](#) on page 562
- PERiod, [":MEASure:PERiod"](#) on page 263
- PERsistence, [":DISPlay:PERsistence"](#) on page 197
- PHASe, [":MEASure:PHASe"](#) on page 264
- PMODE, [":CHANnel<n>:PMODE"](#) on page 549
- [":POD<n>:DISPlay"](#) on page 288
- [":POD<n>:SIZE"](#) on page 289
- [":POD<n>:THReshold"](#) on page 290
- POINTs Commands:
 - [":ACQuire:POINTs"](#) on page 139
 - [":WAVeform:POINTs"](#) on page 489
 - [":WAVeform:POINTs:MODE"](#) on page 491
- POLarity Commands:
 - [":TRIGger:GLITch:POLarity"](#) on page 413
 - [":TRIGger:TV:POLarity"](#) on page 454
 - [":TRIGger:UART:POLarity"](#) on page 465
- POSition Commands:
 - [":DIGital<n>:POSition"](#) on page 187
 - [":TIMEbase:POSition"](#) on page 349
 - [":TIMEbase:WINDow:POSition"](#) on page 355
- PREamble, [":WAVeform:PREamble"](#) on page 493
- PREShoot, [":MEASure:PREShoot"](#) on page 265
- [":PRINT"](#) on page 124
- [":PRINT?"](#) on page 577
- PRINter, [":HARDcopy:PRinter:LIST"](#) on page 230
- PROBe Commands:
 - [":CHANnel<n>:PROBe"](#) on page 174
 - [":EXTernal:PROBe"](#) on page 204
- PROTection Commands:
 - [":CHANnel<n>:PROTection"](#) on page 178
 - [":EXTernal:PROTection"](#) on page 207

- Pulse Width (GLITch), [":TRIGger:GLITch Commands"](#) on page 408
- PWD Commands:
 - [":RECall:PWD"](#) on page 295
 - [":SAVE:PWD"](#) on page 306
- PWIDth, [":MEASure:PWIDth"](#) on page 266
- Q**
 - QUALifier Commands:
 - [":TRIGger:DURation:QUALifier"](#) on page 384
 - [":TRIGger:GLITch:QUALifier"](#) on page 414
 - [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 423
 - [":TRIGger:UART:QUALifier"](#) on page 466
- R**
 - RANGe Commands:
 - [":CHANnel<n>:RANGe"](#) on page 179
 - [":EXTernal:RANGe"](#) on page 208
 - [":FUNCTion:RANGe"](#) on page 216
 - [":TIMEbase:RANGe"](#) on page 350
 - [":TIMEbase:WINDow:RANGe"](#) on page 356
 - [":TRIGger:DURation:RANGe"](#) on page 385
 - [":TRIGger:GLITch:RANGe"](#) on page 415
 - ["*RCL \(Recall\)"](#) on page 82
 - [":RECall:FILEname"](#) on page 293
 - [":RECall:IMAGe\[:START\]"](#) on page 294
 - [":RECall:PWD"](#) on page 295
 - [":RECall:SETup\[:START\]"](#) on page 296
 - REFClock, [":TIMEbase:REFClock"](#) on page 351
 - REFerence Commands:
 - [":FUNCTion:REFerence"](#) on page 217
 - [":TIMEbase:REFerence"](#) on page 352
 - REJect, [":TRIGger\[:EDGE\]:REJect"](#) on page 393
 - RESet Commands:
 - [":SBUS:CAN:COUNt:RESet"](#) on page 320
 - [":SBUS:FLEXray:COUNt:RESet"](#) on page 325
 - [":SBUS:UART:COUNt:RESet"](#) on page 334
 - [":TRIGger:SEQuence:RESet"](#) on page 439
 - RISetime, [":MEASure:RISetime"](#) on page 267

- "Root (:) Commands" on page 94
 - RSiGnal, ":ACQuire:RSiGnal" on page 140
 - "*RST (Reset)" on page 83
 - ":RUN" on page 125
 - RX, ":TRIGger:UART:SOURce:RX" on page 467
 - RXFRames, ":SBUS:UART:COUNt:RXFRames" on page 335
- S**
- SAMPlEpoint Commands:
 - ":TRIGger:CAN:SAMPlEpoint" on page 375
 - ":TRIGger:LIN:SAMPlEpoint" on page 428
 - "*SAV (Save)" on page 86
 - ":SAVE:FILEname" on page 299
 - ":SAVE:IMAGe:AREA" on page 301
 - ":SAVE:IMAGe:FACTors" on page 302
 - ":SAVE:IMAGe:FORMat" on page 303
 - ":SAVE:IMAGe:INKSaver" on page 304
 - ":SAVE:IMAGe:PALette" on page 305
 - ":SAVE:IMAGe[:STARt]" on page 300
 - ":SAVE:PWD" on page 306
 - ":SAVE:SETup[:STARt]" on page 307
 - ":SAVE:WAVEform:FORMat" on page 309
 - ":SAVE:WAVEform:LENGth" on page 310
 - ":SAVE:WAVEform[:STARt]" on page 308
 - ":SBUS:BUSDoctor:ADDRes" on page 314
 - ":SBUS:BUSDoctor:BAUDrate" on page 315
 - ":SBUS:BUSDoctor:CHANnel" on page 316
 - ":SBUS:BUSDoctor:MODE" on page 317
 - ":SBUS:CAN:COUNt:ERRor" on page 318
 - ":SBUS:CAN:COUNt:OVERload" on page 319
 - ":SBUS:CAN:COUNt:RESet" on page 320
 - ":SBUS:CAN:COUNt:TOTal" on page 321
 - ":SBUS:CAN:COUNt:UTILization" on page 322
 - ":SBUS:DISPlay" on page 323
 - ":SBUS:FLEXray:COUNt:NULL" on page 324
 - ":SBUS:FLEXray:COUNt:RESet" on page 325
 - ":SBUS:FLEXray:COUNt:SYNC" on page 326

- ":SBUS:FLEXray:COUNT:TOTAL" on page 327
- ":SBUS:IIC:ASIZE" on page 328
- ":SBUS:LIN:PARity" on page 329
- ":SBUS:MODE" on page 330
- ":SBUS:SPI:WIDTH" on page 331
- ":SBUS:UART:BASE" on page 332
- ":SBUS:UART:COUNT:ERRor" on page 333
- ":SBUS:UART:COUNT:RESet" on page 334
- ":SBUS:UART:COUNT:RXFRames" on page 335
- ":SBUS:UART:COUNT:TXFRames" on page 336
- ":SBUS:UART:FRAMing" on page 337
- SCALE Commands:
 - ":CHANnel<n>:SCALE" on page 180
 - ":FUNction:SCALE" on page 218
 - ":TIMEbase:SCALE" on page 353
 - ":TIMEbase:WINDow:SCALE" on page 357
- SCRatch, ":MEASure:SCRatch" on page 564
- SDEViation, ":MEASure:SDEViation" on page 268
- ":SERial" on page 126
- SEGment, ":TRIGger:FLEXray:TIME:SEGment" on page 405
- SEGmented Commands:
 - ":ACquire:SEGmented:COUNT" on page 141
 - ":ACquire:SEGmented:INDEX" on page 142
 - ":WAVEform:SEGmented:COUNT" on page 496
 - ":WAVEform:SEGmented:TTAG" on page 497
- SETup Commands:
 - ":RECall:SETup[:START]" on page 296
 - ":SAVE:SETup[:START]" on page 307
 - ":SYSTem:SETup" on page 343
- SEQUENCE, ":TRIGger:SEQUence Commands" on page 434
- SHOW, ":MEASure:SHOW" on page 269
- SLOT, ":TRIGger:FLEXray:TIME:SLOT" on page 406
- SIGNAL Commands:
 - ":TRIGger:CAN:SIGNAL:BAUDrate" on page 376
 - ":TRIGger:CAN:SIGNAL:DEFinition" on page 581

- [":TRIGger:LIN:SIGNal:BAUDrate"](#) on page 429
- [":TRIGger:LIN:SIGNal:DEFinition"](#) on page 582
- [":SINGLE"](#) on page 127
- SIZE Commands:
 - [":DIGital<n>:SIZE"](#) on page 188
 - [":POD<n>:SIZE"](#) on page 289
- SKEW, [":CHANnel<n>:PROBe:SKEW"](#) on page 176
- SLOPe Commands:
 - [":TRIGger:EBURst:SLOPe"](#) on page 389
 - [":TRIGger\[:EDGE\]:SLOPe"](#) on page 394
 - [":TRIGger:SPI:CLOCK:SLOPe"](#) on page 443
- SOURce Commands:
 - [":DISPlay:SOURce"](#) on page 198
 - [":FUNCTion:SOURce"](#) on page 219
 - [":MEASure:SOURce"](#) on page 270
 - [":TRIGger:CAN:SOURce"](#) on page 377
 - [":TRIGger:GLITCh:SOURce"](#) on page 416
 - [":TRIGger:IIC:SOURce:CLOCK"](#) on page 421
 - [":TRIGger:IIC:SOURce:DATA"](#) on page 422
 - [":TRIGger:LIN:SOURce"](#) on page 430
 - [":TRIGger:SPI:SOURce:CLOCK"](#) on page 448
 - [":TRIGger:SPI:SOURce:DATA"](#) on page 449
 - [":TRIGger:SPI:SOURce:FRAMe"](#) on page 450
 - [":TRIGger:TV:SOURce"](#) on page 455
 - [":TRIGger:UART:SOURce:RX"](#) on page 467
 - [":TRIGger:UART:SOURce:TX"](#) on page 468
 - [":TRIGger:USB:SOURce:DMINus"](#) on page 472
 - [":TRIGger:USB:SOURce:DPLus"](#) on page 473
 - [":WAVEform:SOURce"](#) on page 498
 - [":WAVEform:SOURce:SUBSource"](#) on page 502
- SPAN, [":FUNCTion:SPAN"](#) on page 220
- SPEEd, [":TRIGger:USB:SPEEd"](#) on page 474
- SPI Commands:
 - [":SBUS:SPI:WIDTh"](#) on page 331
 - [":TRIGger:SPI Commands"](#) on page 442

- SRATe, ":ACQuire:SRATe" on page 144
 - "*SRE (Service Request Enable)" on page 87
 - STANdard Commands:
 - ":TRIGger:LIN:STANdard" on page 431
 - ":TRIGger:TV:STANdard" on page 456
 - STARt Commands:
 - ":CALibrate:STARt" on page 159
 - ":HARDcopy:STARt" on page 231
 - ":RECall:IMAGe[:STARt]" on page 294
 - ":RECall:SETup[:STARt]" on page 296
 - ":SAVE:IMAGe[:STARt]" on page 300
 - ":SAVE:SETup[:STARt]" on page 307
 - ":SAVE:WAVEform[:STARt]" on page 308
 - STATus Commands:
 - ":CALibrate:STATus" on page 160
 - ":STATus" on page 128
 - "*STB (Read Status Byte)" on page 89
 - ":STOP" on page 129
 - SUBSource, ":WAVEform:SOURce:SUBSource" on page 502
 - SWEep, ":TRIGger:SWEep" on page 368
 - SWITCh, ":CALibrate:SWITCh" on page 161
 - SYNC, ":SBUS:FLEXray:COUNT:SYNC" on page 326
 - SYNCbreak, ":TRIGger:LIN:SYNChreak" on page 432
 - ":SYSTem:DATE" on page 339
 - ":SYSTem:DSP" on page 340
 - ":SYSTem:ERRor" on page 341
 - ":SYSTem:LOCK" on page 342
 - ":SYSTem:SETup" on page 343
 - ":SYSTem:TIME" on page 345
- T**
- TDELta, ":MEASure:TDELta" on page 565
 - TEDGe, ":MEASure:TEDGe" on page 272
 - TEMPerature, ":CALibrate:TEMPerature" on page 162
 - ":TER (Trigger Event Register)" on page 130
 - THReshold Commands:
 - ":CHANnel:THReshold" on page 546

- [":DIGital<n>:THReshold"](#) on page 189
- [":MEASure:THResholds"](#) on page 566
- [":POD<n>:THReshold"](#) on page 290
- [":TRIGger:THReshold"](#) on page 583
- THResholds, [":MEASure:THResholds"](#) on page 566
- TIME Commands:
 - [":CALibrate:TIME"](#) on page 163
 - [":SYSTem:TIME"](#) on page 345
 - [":TRIGger:FLEXray:TIME:CBASe"](#) on page 403
 - [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 404
 - [":TRIGger:FLEXray:TIME:SEGment"](#) on page 405
 - [":TRIGger:FLEXray:TIME:SLOT"](#) on page 406
- [":TIMEbase:DELay"](#) on page 579
- [":TIMEbase:MODE"](#) on page 348
- [":TIMEbase:POSition"](#) on page 349
- [":TIMEbase:RANGe"](#) on page 350
- [":TIMEbase:REFClock"](#) on page 351
- [":TIMEbase:REFerence"](#) on page 352
- [":TIMEbase:SCALE"](#) on page 353
- [":TIMEbase:VERNier"](#) on page 354
- [":TIMEbase:WINDow:POSition"](#) on page 355
- [":TIMEbase:WINDow:RANGe"](#) on page 356
- [":TIMEbase:WINDow:SCALE"](#) on page 357
- TIMEout, [":TRIGger:SPI:CLOCK:TIMEout"](#) on page 444
- TIMer, [":TRIGger:SEQuence:TIMer"](#) on page 440
- TMAX, [":MEASure:TMAX"](#) on page 567
- TMIN, [":MEASure:TMIN"](#) on page 568
- TOTal Commands:
 - [":SBUS:CAN:COUNt:TOTal"](#) on page 321
 - [":SBUS:FLEXray:COUNt:TOTal"](#) on page 327
- ["*TRG \(Trigger\)"](#) on page 91
- TRIGger Commands:
 - [":TRIGger:CAN:TRIGger"](#) on page 378
 - [":TRIGger:FLEXray:TRIGger"](#) on page 407
 - [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 423

- ":TRIGger:IIC:TRIGger[:TYPE]" on page 424
- ":TRIGger:LIN:TRIGger" on page 433
- ":TRIGger:SEQuence:TRIGger" on page 441
- ":TRIGger:USB:TRIGger" on page 475
- ":TRIGger:HFReject" on page 362
- ":TRIGger:HOLDoff" on page 363
- ":TRIGger:MODE" on page 364
- ":TRIGger:NREJect" on page 365
- ":TRIGger:PATtern" on page 366
- ":TRIGger:SWEep" on page 368
- ":TRIGger:THReshold" on page 583
- ":TRIGger:CAN:ACKNowledge" on page 580
- ":TRIGger:CAN:PATtern:DATA" on page 371
- ":TRIGger:CAN:PATtern:DATA:LENGth" on page 372
- ":TRIGger:CAN:PATtern:ID" on page 373
- ":TRIGger:CAN:PATtern:ID:MODE" on page 374
- ":TRIGger:CAN:SAMplepoint" on page 375
- ":TRIGger:CAN:SIGNal:BAUDrate" on page 376
- ":TRIGger:CAN:SIGNal:DEFinition" on page 581
- ":TRIGger:CAN:SOURce" on page 377
- ":TRIGger:CAN:TRIGger" on page 378
- ":TRIGger:DURation:GREaterthan" on page 381
- ":TRIGger:DURation:LESSthan" on page 382
- ":TRIGger:DURation:PATtern" on page 383
- ":TRIGger:DURation:QUALifier" on page 384
- ":TRIGger:DURation:RANGe" on page 385
- ":TRIGger[:EDGE]:COUpling" on page 391
- ":TRIGger[:EDGE]:LEVel" on page 392
- ":TRIGger[:EDGE]:REJect" on page 393
- ":TRIGger[:EDGE]:SLOPe" on page 394
- ":TRIGger[:EDGE]:SOURce" on page 395
- ":TRIGger:FLEXray:ERRor:TYPE" on page 397
- ":TRIGger:FLEXray:FRAMe:CCBase" on page 399
- ":TRIGger:FLEXray:FRAMe:CCRepetition" on page 400
- ":TRIGger:FLEXray:FRAMe:ID" on page 401

- [":TRIGger:FLEXray:FRAME:TYPE"](#) on page 402
- [":TRIGger:FLEXray:TIME:CBASe"](#) on page 403
- [":TRIGger:FLEXray:TIME:CREPetition"](#) on page 404
- [":TRIGger:FLEXray:TIME:SEGMENT"](#) on page 405
- [":TRIGger:FLEXray:TIME:SLOT"](#) on page 406
- [":TRIGger:FLEXray:TRIGger"](#) on page 407
- [":TRIGger:GLITCh:GREaterthan"](#) on page 410
- [":TRIGger:GLITCh:LESSthan"](#) on page 411
- [":TRIGger:GLITCh:LEVel"](#) on page 412
- [":TRIGger:GLITCh:POLarity"](#) on page 413
- [":TRIGger:GLITCh:QUALifier"](#) on page 414
- [":TRIGger:GLITCh:RANGe"](#) on page 415
- [":TRIGger:GLITCh:SOURce"](#) on page 416
- [":TRIGger:HFReject"](#) on page 362
- [":TRIGger:HOLDoff"](#) on page 363
- [":TRIGger:IIC:PATtern:ADDReSS"](#) on page 418
- [":TRIGger:IIC:PATtern:DATA"](#) on page 419
- [":TRIGger:IIC:PATtern:DATA2"](#) on page 420
- [":TRIGger:IIC:SOURce:CLOCK"](#) on page 421
- [":TRIGger:IIC:SOURce:DATA"](#) on page 422
- [":TRIGger:IIC:TRIGger:QUALifier"](#) on page 423
- [":TRIGger:IIC:TRIGger\[:TYPE\]"](#) on page 424
- [":TRIGger:LIN:SIGNAL:BAUDrate"](#) on page 429
- [":TRIGger:LIN:SIGNAL:DEFinition"](#) on page 582
- [":TRIGger:LIN:SOURce"](#) on page 430
- [":TRIGger:LIN:TRIGger"](#) on page 433
- [":TRIGger:MODE"](#) on page 364
- [":TRIGger:NREJect"](#) on page 365
- [":TRIGger:PATtern"](#) on page 366
- [":TRIGger:SEQuence:COUNT"](#) on page 435
- [":TRIGger:SEQuence:EDGE"](#) on page 436
- [":TRIGger:SEQuence:FIND"](#) on page 437
- [":TRIGger:SEQuence:PATtern"](#) on page 438
- [":TRIGger:SEQuence:RESet"](#) on page 439
- [":TRIGger:SEQuence:TIMer"](#) on page 440

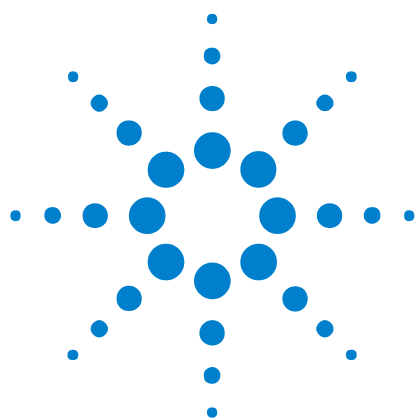
- ":TRIGger:SEQuence:TRIGger" on page 441
- ":TRIGger:SPI:CLOCK:SLOPe" on page 443
- ":TRIGger:SPI:CLOCK:TIMEout" on page 444
- ":TRIGger:SPI:FRAMing" on page 445
- ":TRIGger:SPI:PATtern:DATA" on page 446
- ":TRIGger:SPI:PATtern:WIDTh" on page 447
- ":TRIGger:SPI:SOURce:CLOCK" on page 448
- ":TRIGger:SPI:SOURce:DATA" on page 449
- ":TRIGger:SPI:SOURce:FRAMe" on page 450
- ":TRIGger:SWEep" on page 368
- ":TRIGger:THReshold" on page 583
- ":TRIGger:TV:LINE" on page 452
- ":TRIGger:TV:MODE" on page 453
- ":TRIGger:TV:POLarity" on page 454
- ":TRIGger:TV:SOURce" on page 455
- ":TRIGger:TV:STANdard" on page 456
- ":TRIGger:TV:TVMode" on page 584
- ":TRIGger:UART:BAUDrate" on page 459
- ":TRIGger:UART:BITOrder" on page 460
- ":TRIGger:UART:BURSt" on page 461
- ":TRIGger:UART:DATA" on page 462
- ":TRIGger:UART:IDLE" on page 463
- ":TRIGger:UART:PARity" on page 464
- ":TRIGger:UART:POLarity" on page 465
- ":TRIGger:UART:QUALifier" on page 466
- ":TRIGger:UART:SOURce:RX" on page 467
- ":TRIGger:UART:SOURce:TX" on page 468
- ":TRIGger:UART:TYPE" on page 469
- ":TRIGger:UART:WIDTh" on page 470
- ":TRIGger:USB:SOURce:DMINus" on page 472
- ":TRIGger:USB:SOURce:DPLus" on page 473
- ":TRIGger:USB:SPEed" on page 474
- ":TRIGger:USB:TRIGger" on page 475
- "*TST (Self Test)" on page 92
- TSTArt, ":MEASure:TSTArt" on page 569

- TSTOp, ":MEASure:TSTOp" on page 570
- TTAG, ":WAVEform:SEGmented:TTAG" on page 497
- TV, ":TRIGger:TV Commands" on page 451
- TVALue, ":MEASure:TVALue" on page 274
- TVOLt, ":MEASure:TVOLt" on page 571
- TX, ":TRIGger:UART:SOURce:TX" on page 468
- TXFRames, ":SBUS:UART:COUNt:TXFRames" on page 336
- TYPE Commands:
 - ":ACQuire:TYPE" on page 145
 - ":WAVEform:TYPE" on page 503
 - ":TRIGger:FLEXray:ERRor:TYPE" on page 397
 - ":TRIGger:FLEXray:FRAMe:TYPE" on page 402
 - ":TRIGger:IIC:TRIGger[:TYPE]" on page 424
 - ":TRIGger:UART:TYPE" on page 469

- U**
- UART Commands:
 - ":SBUS:UART:BASE" on page 332
 - ":SBUS:UART:COUNt:ERRor" on page 333
 - ":SBUS:UART:COUNt:RESet" on page 334
 - ":SBUS:UART:COUNt:RXFRames" on page 335
 - ":SBUS:UART:COUNt:TXFRames" on page 336
 - ":SBUS:UART:FRAMing" on page 337
 - ":TRIGger:UART:BAUDrate" on page 459
 - ":TRIGger:UART:BITorder" on page 460
 - ":TRIGger:UART:BURSt" on page 461
 - ":TRIGger:UART:DATA" on page 462
 - ":TRIGger:UART:IDLE" on page 463
 - ":TRIGger:UART:PARity" on page 464
 - ":TRIGger:UART:POLarity" on page 465
 - ":TRIGger:UART:QUALifier" on page 466
 - ":TRIGger:UART:SOURce:RX" on page 467
 - ":TRIGger:UART:SOURce:TX" on page 468
 - ":TRIGger:UART:TYPE" on page 469
 - ":TRIGger:UART:WIDTh" on page 470
 - UNITs Commands:
 - ":CHANnel<n>:UNITs" on page 181

- [":EXtErnal:UNITs"](#) on page 209
 - [UNSigned, ":WAVeform:UNSigned"](#) on page 504
 - [UPPer, ":MEASure:UPPer"](#) on page 573
 - [USB, ":TRIGger:USB Commands"](#) on page 471
 - [UTILization, ":SBUS:CAN:COUNt:UTILization"](#) on page 322
- V**
- [VAMplitude, ":MEASure:VAMplitude"](#) on page 276
 - [VAverage, ":MEASure:VAverage"](#) on page 277
 - [VBASe, ":MEASure:VBASe"](#) on page 278
 - [VDELta, ":MEASure:VDELta"](#) on page 574
 - [VECTors, ":DISPlay:VECTors"](#) on page 199
 - [VERNier, ":CHANnel<n>:VERNier"](#) on page 182
 - [":VIEW"](#) on page 131
 - [VMAX, ":MEASure:VMAX"](#) on page 279
 - [VMIN, ":MEASure:VMIN"](#) on page 280
 - [VPP, ":MEASure:VPP"](#) on page 281
 - [VRMS, ":MEASure:VRMS"](#) on page 282
 - [VStArt, ":MEASure:VStArt"](#) on page 575
 - [VStOp, ":MEASure:VStOp"](#) on page 576
 - [VTIME, ":MEASure:VTIME"](#) on page 283
 - [VTOp, ":MEASure:VTOp"](#) on page 284
- W**
- ["*WAI \(Wait To Continue\)"](#) on page 93
 - [WAVeform Commands:](#)
 - [":SAVE:WAVeform:FORMat"](#) on page 309
 - [":SAVE:WAVeform:LENGth"](#) on page 310
 - [":SAVE:WAVeform\[:StArt\]"](#) on page 308
 - [":WAVeform:BYTeorder"](#) on page 484
 - [":WAVeform:COUNt"](#) on page 485
 - [":WAVeform:DATA"](#) on page 486
 - [":WAVeform:FORMat"](#) on page 488
 - [":WAVeform:POINts"](#) on page 489
 - [":WAVeform:POINts:MODE"](#) on page 491
 - [":WAVeform:PREamble"](#) on page 493
 - [":WAVeform:SEGmented:COUNt"](#) on page 496
 - [":WAVeform:SEGmented:TTAG"](#) on page 497

- [":WAVeform:SOURce"](#) on page 498
 - [":WAVeform:SOURce:SUBSource"](#) on page 502
 - [":WAVeform:TYPE"](#) on page 503
 - [":WAVeform:UNSigned"](#) on page 504
 - [":WAVeform:VIEW"](#) on page 505
 - [":WAVeform:XINCrement"](#) on page 506
 - [":WAVeform:XORigin"](#) on page 507
 - [":WAVeform:XREFerence"](#) on page 508
 - [":WAVeform:YINCrement"](#) on page 509
 - [":WAVeform:YORigin"](#) on page 510
 - [":WAVeform:YREFerence"](#) on page 511
 - **WIDTh Commands:**
 - [":SBUS:SPI:WIDTh"](#) on page 331
 - [":TRIGger:SPI:PATtern:WIDTh"](#) on page 447
 - [":TRIGger:UART:WIDTh"](#) on page 470
 - **WINDow, ":FUNcTion:WINDow"** on page 221
- X**
- **X1Position, ":MARKer:X1Position"** on page 235
 - **X1Y1source, ":MARKer:X1Y1source"** on page 236
 - **X2Position, ":MARKer:X2Position"** on page 237
 - **X2Y2source, ":MARKer:X2Y2source"** on page 238
 - **XDELta, ":MARKer:XDELta"** on page 239
 - **XINCrement, ":WAVeform:XINCrement"** on page 506
 - **XMAX, ":MEASure:XMAX"** on page 285
 - **XMIN, ":MEASure:XMIN"** on page 286
 - **XORigin, ":WAVeform:XORigin"** on page 507
 - **XREFerence, ":WAVeform:XREFerence"** on page 508
- Y**
- **Y1Position, ":MARKer:Y1Position"** on page 240
 - **Y2Position, ":MARKer:Y2Position"** on page 241
 - **YDELta, ":MARKer:YDELta"** on page 242
 - **YINCrement, ":WAVeform:YINCrement"** on page 509
 - **YORigin, ":WAVeform:YORigin"** on page 510
 - **YREFerence, ":WAVeform:YREFerence"** on page 511



5 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 616).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 167)	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see page 168)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 171)	
ANALog<n>:LABel	:CHANnel<n>:LABel (see page 172)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 173)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 174)	
ANALog<n>:PMODE	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see page 179)	
:CHANnel:ACTivity (see page 544)	:ACTivity (see page 97)	
:CHANnel:LABel (see page 545)	:CHANnel<n>:LABel (see page 172) or :DIGital<n>:LABel (see page 186)	use CHANnel<n>:LABel for analog channels and use DIGital<n>:LABel for digital channels
:CHANnel:THReshold (see page 546)	:POD<n>:THReshold (see page 290) or :DIGital<n>:THReshold (see page 189)	
:CHANnel2:SKEW (see page 547)	:CHANnel<n>:PROBe:SKEW (see page 176)	



5 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:CHANnel<n>:INPut (see page 548)	:CHANnel<n>:IMPedance (see page 170)	
:CHANnel<n>:PMODE (see page 549)	none	
:DISPlay:CONNect (see page 550)	:DISPlay:VECTors (see page 199)	
:DISPlay:ORDer (see page 551)	none	
:ERASe (see page 552)	:CDISplay (see page 104)	
:EXTernal:INPut (see page 553)	:EXTernal:IMPedance (see page 203)	
:EXTernal:PMODE (see page 554)	none	
FUNcTion1, FUNcTion2	:FUNcTion Commands (see page 210)	ADD not included
:FUNcTion:VIEW (see page 555)	:FUNcTion:DISPlay (see page 213)	
:HARDcopy:DESTination (see page 556)	:HARDcopy:FILename (see page 558)	
:HARDcopy:DEVice (see page 557)	:HARDcopy:FORMat (see page 559)	PLOTter, THINkjet not supported; TIF, BMP, CSV, SEIko added
:HARDcopy:FILename (see page 558)	:RECall:FILename (see page 293) :SAVE:FILename (see page 293)	
:HARDcopy:FORMat (see page 559)	:HARDcopy:APRinter (see page 225) :SAVE:IMAGe:FORMat (see page 303) :SAVE:WAVEform:FORMat (see page 309)	
:HARDcopy:GRAYscale (see page 560)	:HARDcopy:PALETTE (see page 229)	
:HARDcopy:IGColors (see page 561)	:HARDcopy:INKSaver (see page 228)	
:HARDcopy:PDRiver (see page 562)	:HARDcopy:APRinter (see page 225)	
:MEASure:LOWer (see page 563)	:MEASure:DEFine:THResholds (see page 252)	MEASure:DEFine:THResholds can define absolute values or percentage

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:SCRatch (see page 564)	:MEASure:CLEar (see page 250)	
:MEASure:TDELta (see page 565)	:MARKer:XDELta (see page 239)	
:MEASure:THResholds (see page 566)	:MEASure:DEFine:THResholds (see page 252)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 567)	:MEASure:XMAX (see page 285)	
:MEASure:TMIN (see page 568)	:MEASure:XMIN (see page 286)	
:MEASure:TSTArt (see page 569)	:MARKer:X1Position (see page 235)	
:MEASure:TSTOp (see page 570)	:MARKer:X2Position (see page 237)	
:MEASure:TVOLt (see page 571)	:MEASure:TVALue (see page 274)	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see page 573)	:MEASure:DEFine:THResholds (see page 252)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 574)	:MARKer:YDELta (see page 242)	
:MEASure:VSTArt (see page 575)	:MARKer:Y1Position (see page 240)	
:MEASure:VSTOp (see page 576)	:MARKer:Y2Position (see page 241)	
:PRINt? (see page 577)	:DISPlay:DATA? (see page 193)	
:TIMebase:DELay (see page 579)	:TIMebase:POSition (see page 349) or :TIMebase:WINDow:POSition (see page 355)	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of delayed time base window.
:TRIGger:CAN:ACKnowledge (see page 580)	none	
:TRIGger:CAN:SIGNal:DEFinition (see page 581)	none	
:TRIGger:LIN:SIGNal:DEFinition (see page 582)	none	

5 Obsolete and Discontinued Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:TRIGger:THReshold (see page 583)	:POD<n>:THReshold (see page 290) or :DIGital<n>:THReshold (see page 189)	
:TRIGger:TV:TVMODE (see page 584)	:TRIGger:TV:MODE (see page 453)	

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 7000 Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERsistence INFinite (see page 197)	
CHANnel:MATH	:FUNCTion:OPERation (see page 215)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTectioN (see page 178)	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	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported

Discontinued Command	Current Command Equivalent	Comments
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see page 92)	
TRACE subsystem	none	All commands discontinued, feature not available
TRIGger:ADVanced subsystem		Use new GLITch, PATtern, or TV trigger modes
TRIGger:TV:FIEld	:TRIGger:TV:MODE (see page 453)	
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 7000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:ACTivity

O (see [page 616](#))

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 97](#)) 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 616](#))

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 172](#)) or :DIGital<n>:LABel command (see [page 186](#)) for the InfiniiVision 7000 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 616](#))

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 290](#)) or :DIGital<n>:THReshold command (see [page 189](#)) for the InfiniiVision 7000 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 616](#))

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 176](#)) 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 165

:CHANnel<n>:INPut

O (see [page 616](#))

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 M Ω) and FIFTy (50 Ω).

NOTE

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see [page 170](#)) 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 616](#))

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 616](#))

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 199](#)) 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 199

:DISPlay:ORDer

O (see [page 616](#))

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 187

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 6"](#) on page 692

:ERASe

O (see [page 616](#))

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 104](#)) instead.

:EXternal:INPut

O (see [page 616](#))

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 203](#)) 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 200
 - ["Introduction to :TRIGger Commands"](#) on page 358
 - [":CHANnel<n>:IMPedance"](#) on page 170

:EXternal:PMODE

O (see [page 616](#))

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:VIEW

O (see [page 616](#))

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 213](#)) 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 616](#))

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 558](#)) 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 222
 - [":HARDcopy:FORMat"](#) on page 559

:HARDcopy:DEVICE

O (see [page 616](#))

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 559](#)) 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 616](#))

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 299](#)) and :RECall:FILENAME command (see [page 293](#)) 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 222
 - "[:HARDcopy:FORMat](#)" on page 559

:HARDcopy:FORMat

O (see [page 616](#))

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 303](#)), :SAVE:WAVEform:FORMat (see [page 309](#)), and :HARDcopy:APRinter (see [page 225](#)) 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 222

:HARDcopy:GRAYscale

O (see [page 616](#))

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 229](#)) 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 222

:HARDcopy:IGColors

O (see [page 616](#))

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 228](#)) 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 222

:HARDcopy:PDRiver

O (see [page 616](#))

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 225](#)) 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 222
 - ":HARDcopy:FORMat" on page 559

:MEASure:LOWer

O (see [page 616](#))

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 252](#)) 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 248
 - "[:MEASure:THResholds](#)" on page 566
 - "[:MEASure:UPPer](#)" on page 573

:MEASure:SCRatch

O (see [page 616](#))

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 250](#)) instead.

:MEASure:TDELta

O (see [page 616](#))

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 239](#)) instead.

Return Format <value><NL>

<value> ::= time difference between start and stop markers in NR3 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 233
 - "[Introduction to :MEASure Commands](#)" on page 248
 - "[:MARKer:X1Position](#)" on page 235
 - "[:MARKer:X2Position](#)" on page 237
 - "[:MARKer:XDELta](#)" on page 239
 - "[:MEASure:TSTArt](#)" on page 569
 - "[:MEASure:TSTOp](#)" on page 570

:MEASure:THResholds

O (see [page 616](#))

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 252](#)) 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 248](#)
 - "[:MEASure:LOWer](#)" on [page 563](#)
 - "[:MEASure:UPPer](#)" on [page 573](#)

:MEASure:TMAX

O (see [page 616](#))

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 285](#)) 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 248
 - [":MEASure:TMIN"](#) on page 568
 - [":MEASure:XMAX"](#) on page 285
 - [":MEASure:XMIN"](#) on page 286

:MEASure:TMIN

O (see [page 616](#))

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 286](#)) 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 248
 - "[:MEASure:TMAX](#)" on page 567
 - "[:MEASure:XMAX](#)" on page 285
 - "[:MEASure:XMIN](#)" on page 286

:MEASure:TSTArt

O (see [page 616](#))

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 618](#)). 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 235](#)) 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 233
 - ["Introduction to :MEASure Commands"](#) on page 248
 - [":MARKer:X1Position"](#) on page 235
 - [":MARKer:X2Position"](#) on page 237
 - [":MARKer:XDELta"](#) on page 239
 - [":MEASure:TDELta"](#) on page 565
 - [":MEASure:TSTOP"](#) on page 570

:MEASure:TSTOp

O (see [page 616](#))

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 618](#)). 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 237](#)) 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 233
 - "[Introduction to :MEASure Commands](#)" on page 248
 - "[:MARKer:X1Position](#)" on page 235
 - "[:MARKer:X2Position](#)" on page 237
 - "[:MARKer:XDELta](#)" on page 239
 - "[:MEASure:TDELta](#)" on page 565
 - "[:MEASure:TSTArt](#)" on page 569

:MEASure:TVOLt

O (see [page 616](#))

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 274](#)) for the InfiniiVision 7000 Series oscilloscopes.

Return Format <value><NL>

5 Obsolete and Discontinued Commands

<value> ::= time in seconds of the specified voltage crossing
in NR3 format

:MEASure:UPPer

O (see [page 616](#))

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 252](#)) 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 248
 - "[:MEASure:LOWer](#)" on page 563
 - "[:MEASure:THResholds](#)" on page 566

:MEASure:VDELta

O (see [page 616](#))

Query Syntax :MEASure:VDELta?

The :MEASure:VDELta? query returns the voltage difference between vertical marker 1 (Y1 cursor) and vertical marker 2 (Y2 cursor). No measurement is made when the :MEASure:VDELta? query is received by the oscilloscope. The delta value that is returned is the current value. This is the same value as the front-panel cursors delta Y value.

VDELta = value at marker 2 - value at marker 1

NOTE

The :MEASure:VDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:YDELta command (see [page 242](#)) instead.

Return Format <value><NL>

<value> ::= delta V value in NR1 format

- See Also**
- "[Introduction to :MARKer Commands](#)" on page 233
 - "[Introduction to :MEASure Commands](#)" on page 248
 - "[:MARKer:Y1Position](#)" on page 240
 - "[:MARKer:Y2Position](#)" on page 241
 - "[:MARKer:YDELta](#)" on page 242
 - "[:MEASure:TDELta](#)" on page 565
 - "[:MEASure:TSTArt](#)" on page 569

:MEASure:VSTArt

O (see [page 616](#))

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 618](#)). 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 240](#)) 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 233
 - "[Introduction to :MEASure Commands](#)" on page 248
 - "[:MARKer:Y1Position](#)" on page 240
 - "[:MARKer:Y2Position](#)" on page 241
 - "[:MARKer:YDELta](#)" on page 242
 - "[:MARKer:X1Y1source](#)" on page 236
 - "[:MEASure:SOURce](#)" on page 270
 - "[:MEASure:TDELta](#)" on page 565
 - "[:MEASure:TSTArt](#)" on page 569

:MEASure:VSTOp

O (see [page 616](#))

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 618](#)). 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 241](#)) 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 233
 - "[Introduction to :MEASure Commands](#)" on page 248
 - "[:MARKer:Y1Position](#)" on page 240
 - "[:MARKer:Y2Position](#)" on page 241
 - "[:MARKer:YDELta](#)" on page 242
 - "[:MARKer:X2Y2source](#)" on page 238
 - "[:MEASure:SOURce](#)" on page 270
 - "[:MEASure:TDELta](#)" on page 565
 - "[:MEASure:TSTArt](#)" on page 569

:PRINt?

O (see [page 616](#))

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 193](#)) 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

5 Obsolete and Discontinued Commands

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 96
 - ["Introduction to :HARDcopy Commands"](#) on page 222
 - [":HARDcopy:FORMat"](#) on page 559
 - [":HARDcopy:FACTors"](#) on page 226
 - [":HARDcopy:GRAYscale"](#) on page 560
 - [":DISPlay:DATA"](#) on page 193

:TIMEbase:DElay

O (see [page 616](#))

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 352](#)).

NOTE

The :TIMEbase:DElay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMEbase:POsition command (see [page 349](#)) 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 6](#)" on [page 692](#)

:TRIGger:CAN:ACKnowledge

O (see [page 616](#))

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 7000 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 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:CAN:TRIGger](#)" on page 378

:TRIGger:CAN:SIGNal:DEFinition

O (see [page 616](#))

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 7000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is DIFF.

Query Syntax :TRIGger:CAN:SIGNal:DEFinition?

The :TRIGger:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Return Format <value><NL>
 <value> ::= DIFF

- See Also**
- "[Introduction to :TRIGger Commands](#)" on page 358
 - "[:TRIGger:MODE](#)" on page 364
 - "[:TRIGger:CAN:SIGNal:BAUDrate](#)" on page 376
 - "[:TRIGger:CAN:SOURce](#)" on page 377
 - "[:TRIGger:CAN:TRIGger](#)" on page 378

:TRIGger:LIN:SIGNal:DEFinition

O (see [page 616](#))

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 7000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is LIN.

Query Syntax :TRIGger:LIN:SIGNal:DEFinition?

The :TRIGger:LIN:SIGNal:DEFinition? query returns the current LIN signal type.

Return Format <value><NL>
<value> ::= LIN

- See Also**
- ["Introduction to :TRIGger Commands"](#) on page 358
 - [":TRIGger:MODE"](#) on page 364
 - [":TRIGger:LIN:SIGNal:BAUDrate"](#) on page 429
 - [":TRIGger:LIN:SOURce"](#) on page 430

:TRIGger:THReshold

O (see [page 616](#))

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 290](#)), :DIGital<n>:THReshold command (see [page 189](#)), or :TRIGger[:EDGE]:LEVel command (see [page 392](#)) for the InfiniiVision 7000 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 616](#))

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 456](#)).

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

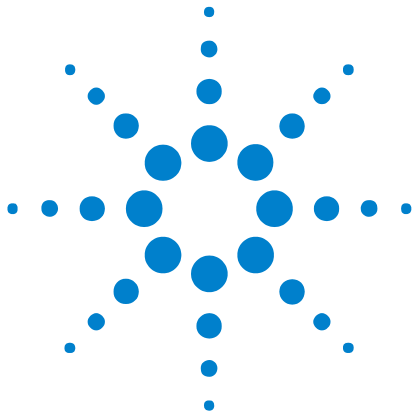
The :TRIGger:TV:TVMode command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see [page 453](#)) instead.

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}
```

6 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

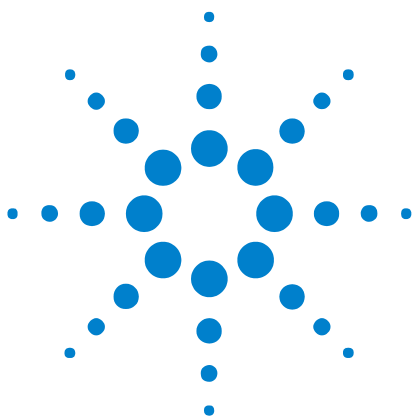
+108, Compression Error

+109, No Data For Operation

+112, Unknown File Type

+113, Directory Not Supported

6 Error Messages



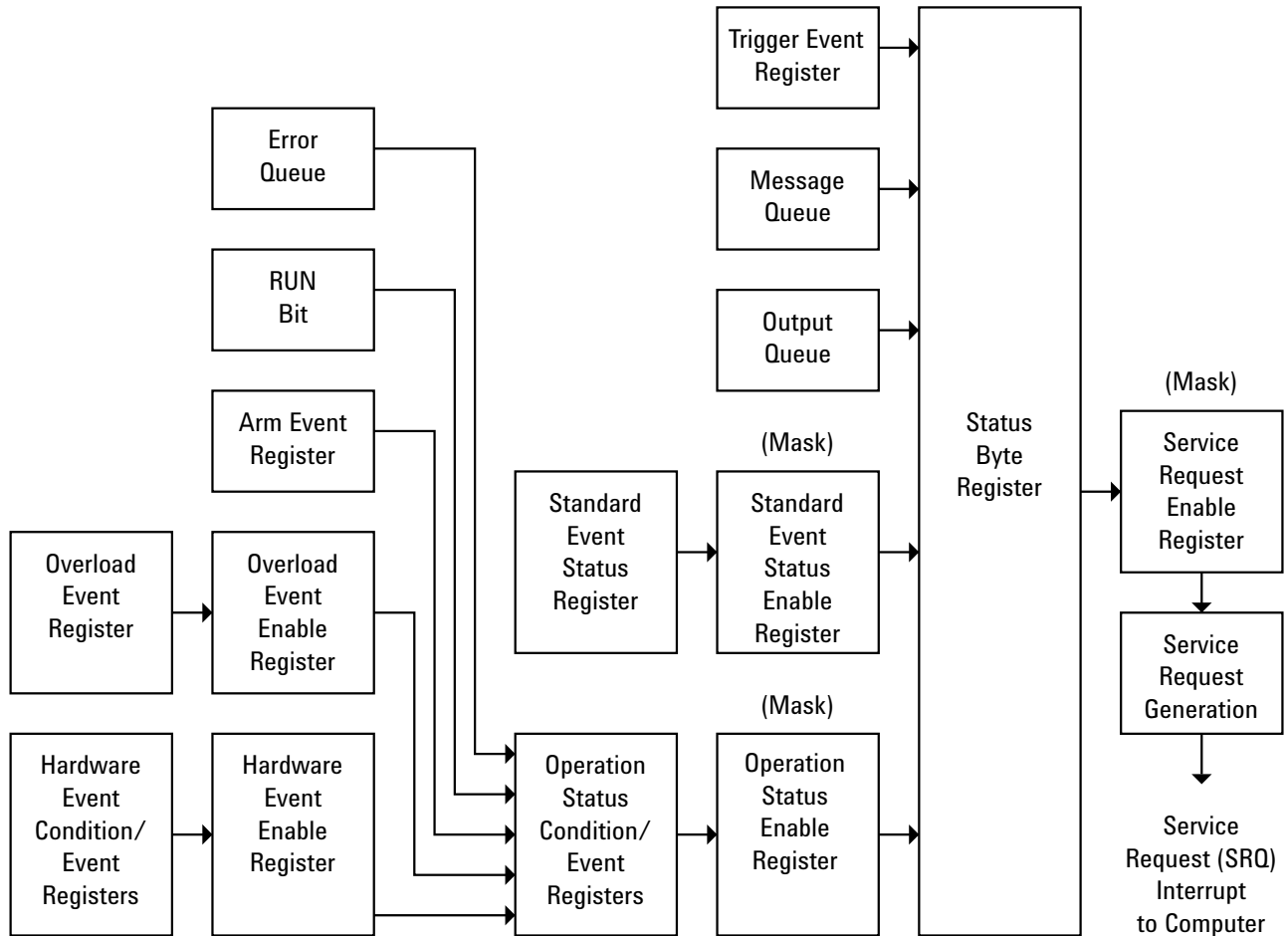
7 Status Reporting

Status Reporting Data Structures	595
Status Byte Register (STB)	598
Service Request Enable Register (SRE)	600
Trigger Event Register (TER)	601
Output Queue	602
Message Queue	603
(Standard) Event Status Register (ESR)	604
(Standard) Event Status Enable Register (ESE)	605
Error Queue	606
Operation Status Event Register (:OPERRegister[:EVENTt])	607
Operation Status Condition Register (:OPERRegister:CONDition)	608
Arm Event Register (AER)	609
Hardware Event Event Register (:HWERRegister[:EVENTt])	610
Hardware Event Condition Register (:HWERRegister:CONDition)	611
Clearing Registers and Queues	612
Status Reporting Decision Chart	613

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).

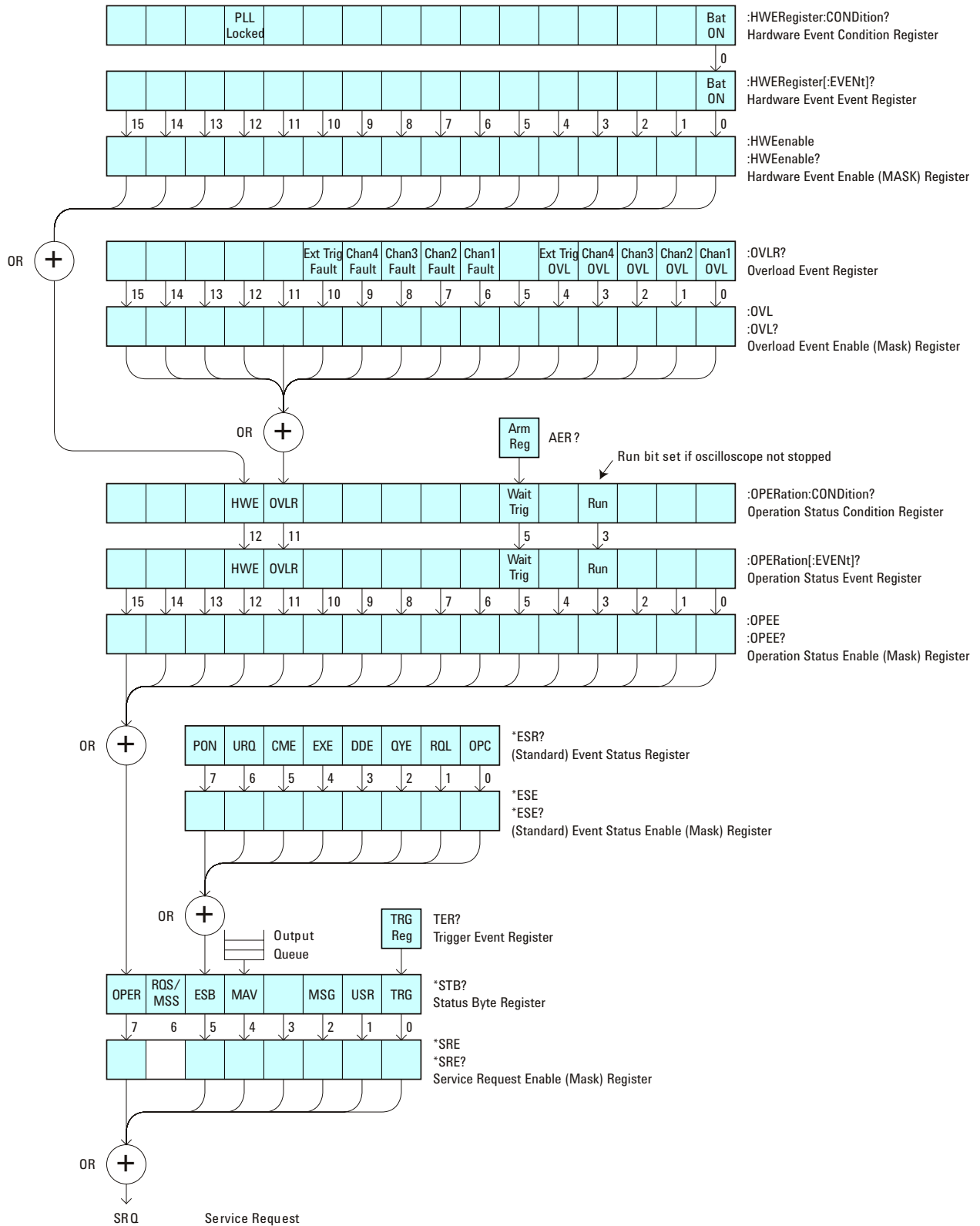
The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

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.

7 Status Reporting



The status register bits are described in more detail in the following tables:

- "Status Byte Register (STB)" on page 89
- "Standard Event Status Register (ESR)" on page 76
- "Operation Status Condition Register" on page 116
- "Operation Status Event Register" on page 118
- "Overload Event Register (OVLRL)" on page 122
- "Hardware Event Condition Register" on page 109
- "Hardware Event Event Register" on page 111

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 GPIB serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? command reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

No other bits in the Status Byte Register are cleared by either the *STB? query or the serial poll, except the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? command.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

The following example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register.

```
myScope.WriteString "*STB?"
varQueryResult = myScope.ReadNumber
MsgBox "Status Byte Register, Read: 0x" + Hex(varQueryResult)
```

The next program prints 0xD1 and clears bit 6 (RQS) and bit 4 (MAV) of the Status Byte Register. The difference in the output value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set and is cleared when the Status Byte Register is read by the serial poll command.

Example The following example uses the resource session object's ReadSTB method to read the contents of the oscilloscope's Status Byte Register.

```
varQueryResult = myScope.IO.ReadSTB
MsgBox "Status Byte Register, Serial Poll: 0x" + Hex(varQueryResult)
```

NOTE

Use Serial Polling to Read Status Byte Register. Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

Service Request Enable Register (SRE)

Setting the Service Request Enable Register bits enable corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command and the bits that are set are read with the *SRE? query.

Example The following example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
myScope.WriteString "*SRE " + CStr(CInt("&H30"))
```

This example uses the decimal parameter value of 48, the string returned by CStr(CInt("&H30")), to enable the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

Trigger Event Register (TER)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TER event register stays set until it is cleared by reading the register or using the *CLS command. If your application needs to detect multiple triggers, the TER event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation, you must clear the event register each time the trigger bit is set.

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

When using the Agilent VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The length of the oscilloscope's message queue is 1. Note that messages sent with the :SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

(Standard) Event Status Register (ESR)

The (Standard) Event Status Register (ESR) monitors the following oscilloscope status events:

- PON - Power On
- URQ - User Request
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occur, the event sets the corresponding bit in the register. If the bits are enabled in the Standard Event Status Enable Register, the bits set in this register generate a summary bit to set bit 5 (ESB) in the Status Byte Register.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all of the bits that are set at the present time.

Example The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
MsgBox "Standard Event Status Register: 0x" + Hex(varQueryResult)
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

(Standard) Event Status Enable Register (ESE)

To allow any of the (Standard) Event Status Register (ESR) bits to generate a summary bit, you must first enable that bit. Enable the bit by using the *ESE (Event Status Enable) common command to set the corresponding bit in the (Standard) Event Status Enable Register (ESE).

Set bits are read with the *ESE? query.

Example Suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the (Standard) Event Status Register are bits 2 through 5 (hexadecimal value 0x3C). Therefore, you can enable any of these bits to generate the summary bit by sending:

```
myScope.WriteString "*ESE " + CStr(CInt("&H3C"))
```

Whenever an error occurs, it sets one of these bits in the (Standard) Event Status Register. Because all the error related bits are enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), an SRQ service request interrupt is sent to the controller PC.

NOTE

Disabled (Standard) Event Status Register bits respond but do not generate a summary bit. (Standard) Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit to the Status Byte Register.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error 350, Queue overflow. Any time the queue overflows, the least recent errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the Queue overflow message).

The error queue is read with the `:SYSTEM:ERROR?` query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return "0, No error".

The error queue is cleared when:

- the instrument is powered up,
- the instrument receives the `*CLS` common command, or
- the last item is read from the error queue.

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

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 50 Ω input overload occurs.
- If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Event Register is read and cleared with the :OPERRegister[:EVENT]? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Operation Status Condition Register (:OPERRegister:CONDition)

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 50Ω 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)

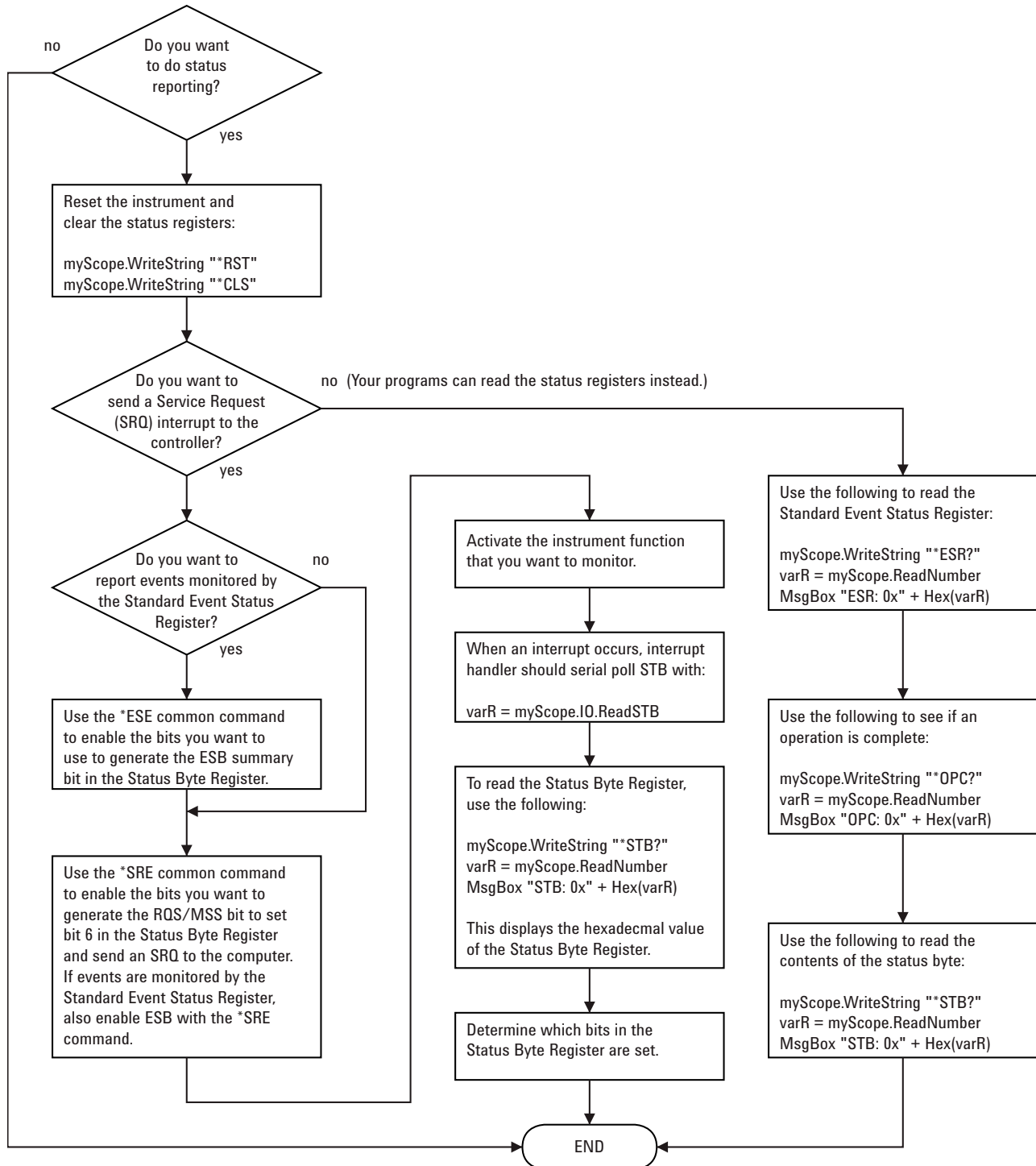
This register hosts the Bat On bit (bit 0) and the PLL LOCKED bit (bit 12).

- 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.

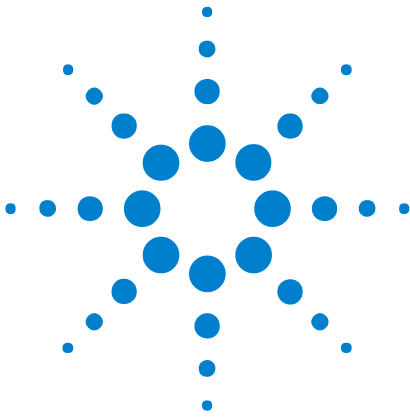
Clearing Registers and Queues

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



7 Status Reporting



8 More About Oscilloscope Commands

- Command Classifications [616](#)
- Valid Command/Query Strings [617](#)
- Query Return Values [635](#)
- All Oscilloscope Commands Are Sequential [636](#)



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 616
- ["Non-Core Commands"](#) on page 616
- ["Obsolete Commands"](#) on page 616

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:

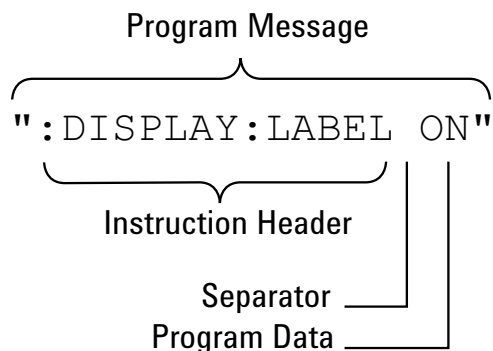
- ["Obsolete and Discontinued Commands"](#) on page 539
- As well as: ["Commands A-Z"](#) on page 513

Valid Command/Query Strings

- "Program Message Syntax" on page 617
- "Command Tree" on page 621
- "Duplicate Mnemonics" on page 633
- "Tree Traversal Rules and Multiple Commands" on page 633

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 618), 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 621 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 619
- "Compound Command Headers" on page 619
- "Common Command Headers" on page 620

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 620 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 GPIB 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 = 2800m = 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 132](#))
 - :AALias (see [page 134](#))
 - :COMPLete (see [page 135](#))
 - :COUNT (see [page 136](#))
 - :DAALias (see [page 137](#))
 - :MODE (see [page 138](#))
 - :POINTs (see [page 139](#))
 - :RSIGnal (see [page 140](#))
 - :SEGMENTed
 - :COUNT (see [page 141](#))
 - :INDEX (see [page 142](#))
 - :SRATE (see [page 144](#))
 - :TYPE (see [page 145](#))
 - :ACTivity (see [page 97](#))
 - :AER (Arm Event Register) (see [page 98](#))
 - :AUToscale (see [page 99](#))
 - :AMODE (see [page 101](#))
 - :CHANnels (see [page 102](#))
 - :BLANK (see [page 103](#))
 - :BUS<n> (see [page 147](#))
 - :BIT<m> (see [page 149](#))

- :BITS (see [page 150](#))
- :CLear (see [page 152](#))
- :DISPlay (see [page 153](#))
- :LABel (see [page 154](#))
- :MASK (see [page 155](#))
- :CALibrate (see [page 156](#))
 - :DATE (see [page 157](#))
 - :LABel (see [page 158](#))
 - :STARt (see [page 159](#))
 - :STATus (see [page 160](#))
 - :SWITch (see [page 161](#))
 - :TEMPerature (see [page 162](#))
 - :TIME (see [page 163](#))
- :CDISplay (see [page 104](#))
- :CHANnel<n> (see [page 164](#))
 - :BWLimit (see [page 167](#))
 - :COUPling (see [page 168](#))
 - :DISPlay (see [page 169](#))
 - :IMPedance (see [page 170](#))
 - :INVert (see [page 171](#))
 - :LABel (see [page 172](#))
 - :OFFSet (see [page 173](#))
 - :PROBe (see [page 174](#))
 - :ID (see [page 175](#))
 - :SKEW (see [page 176](#))
 - :STYPe (see [page 177](#))
 - :PROTection (see [page 178](#))
 - :RANGe (see [page 179](#))
 - :SCALE (see [page 180](#))
 - :UNITs (see [page 181](#))
 - :VERNier (see [page 182](#))
- :DIGital<n> (see [page 183](#))
 - :DISPlay (see [page 185](#))
 - :LABel (see [page 186](#))
 - :POSition (see [page 187](#))

- :SIZE (see [page 188](#))
- :THReshold (see [page 189](#))
- :DIGitize (see [page 105](#))
- :DISPlay (see [page 190](#))
 - :CLEAr (see [page 192](#))
 - :DATA (see [page 193](#))
 - :LABEl (see [page 195](#))
 - :LABList (see [page 196](#))
 - :PERsistence (see [page 197](#))
 - :SOURce (see [page 198](#))
 - :VECTors (see [page 199](#))
- :EXTernal (see [page 200](#))
 - :BWLimit (see [page 202](#))
 - :IMPedance (see [page 203](#))
 - :PROBe (see [page 204](#))
 - :ID (see [page 205](#))
 - :STYPe (see [page 206](#))
 - :PROTection (see [page 207](#))
 - :RANGe (see [page 208](#))
 - :UNITs (see [page 209](#))
- :FUNCTion (see [page 210](#))
 - :CENTer (see [page 212](#))
 - :DISPlay (see [page 213](#))
 - :OFFSet (see [page 214](#))
 - :OPERation (see [page 215](#))
 - :RANGe (see [page 216](#))
 - :REFerence (see [page 217](#))
 - :SCALE (see [page 218](#))
 - :SOURce (see [page 219](#))
 - :SPAN (see [page 220](#))
 - :WINDow (see [page 221](#))
- :HARDcopy (see [page 222](#))
 - :AREA (see [page 224](#))
 - :APRinter (see [page 225](#))
 - :FACTors (see [page 226](#))

- :FFEed (see [page 227](#))
- :INKSaver (see [page 228](#))
- :PALette (see [page 229](#))
- [:PRINter]
 - :LIST (see [page 230](#))
- [:START] (see [page 231](#))
- :HWEenable (Hardware Event Enable Register) (see [page 107](#))
- :HWERegister
 - :CONDition (Hardware Event Condition Register) (see [page 109](#))
 - [:EVENT] (Hardware Event Event Register) (see [page 111](#))
- :MARKer (see [page 232](#))
 - :MODE (see [page 234](#))
 - :X1Position (see [page 235](#))
 - :X1Y1source (see [page 236](#))
 - :X2Position (see [page 237](#))
 - :X2Y2source (see [page 238](#))
 - :XDELta (see [page 239](#))
 - :Y1Position (see [page 240](#))
 - :Y2Position (see [page 241](#))
 - :YDELta (see [page 242](#))
- :MEASure (see [page 243](#))
 - :CLEar (see [page 250](#))
 - :COUNter (see [page 251](#))
 - :DEFine (see [page 252](#))
 - :DELay (see [page 255](#))
 - :DUTYcycle (see [page 257](#))
 - :FALLtime (see [page 258](#))
 - :FREQuency (see [page 259](#))
 - :NWIDth (see [page 260](#))
 - :OVERshoot (see [page 261](#))
 - :PERiod (see [page 263](#))
 - :PHASe (see [page 264](#))
 - :PREShoot (see [page 265](#))
 - :PWIDth (see [page 266](#))
 - :RISetime (see [page 267](#))

- :SDEVIation (see [page 268](#))
- :SHOW (see [page 269](#))
- :SOURce (see [page 270](#))
- :TEDGe (see [page 272](#))
- :TVALue (see [page 274](#))
- :VAMPLitude (see [page 276](#))
- :VAverage (see [page 277](#))
- :VBASe (see [page 278](#))
- :VMAX (see [page 279](#))
- :VMIN (see [page 280](#))
- :VPP (see [page 281](#))
- :VRMS (see [page 282](#))
- :VTIMe (see [page 283](#))
- :VTOP (see [page 284](#))
- :XMAX (see [page 285](#))
- :XMIN (see [page 286](#))
- :MERGe (see [page 113](#))
- :OPEE (Operation Status Enable Register) (see [page 114](#))
- :OPERegister
 - :CONDition (Operation Status Condition Register) (see [page 116](#))
 - [:EVENT] (Operation Status Event Register) (see [page 118](#))
- :OVLenable (Overload Event Enable Register) (see [page 120](#))
- :OVLRegister (Overload Event Register) (see [page 122](#))
- :POD<n> (see [page 287](#))
 - :DISPlay (see [page 288](#))
 - :SIZE (see [page 289](#))
 - :THReshold (see [page 290](#))
- :RECall
 - :FILename (see [page 293](#))
 - :IMAGe (see [page 294](#))
 - [:STARt] (see [page 294](#))
 - :PWD (see [page 295](#))
 - :SETup (see [page 296](#))
 - [:STARt] (see [page 296](#))
- :RUN (see [page 125](#))

- :SAVE
 - :FILename (see [page 299](#))
 - :IMAGe (see [page 300](#))
 - [:START] (see [page 300](#))
 - :AREa (see [page 301](#))
 - :FACTors (see [page 302](#))
 - :FORMat (see [page 303](#))
 - :IGColors (see [page 304](#))
 - :PALette (see [page 305](#))
 - :PWD (see [page 306](#))
 - :SETup (see [page 307](#))
 - [:START] (see [page 307](#))
 - :WAVEform (see [page 308](#))
 - [:START] (see [page 308](#))
 - :FORMat (see [page 309](#))
 - :LENGth (see [page 310](#))
- :SBUS (see [page 311](#))
 - :BUSDoctor
 - :ADDRes (see [page 314](#))
 - :BAUDrate (see [page 315](#))
 - :CHANnel (see [page 316](#))
 - :MODE (see [page 317](#))
 - :CAN
 - :COUNT
 - :ERRor (see [page 318](#))
 - :OVERload (see [page 319](#))
 - :RESet (see [page 320](#))
 - :TOTal (see [page 321](#))
 - :UTILization (see [page 322](#))
- :DISPlay (see [page 323](#))
- :FLEXray
 - :COUNT
 - :NULL? (see [page 324](#))
 - :RESet (see [page 325](#))
 - :SYNC? (see [page 326](#))

- :TOTal? (see [page 327](#))
- :IIC
 - :WIDTh (see [page 331](#))
- :LIN
 - :PARity (see [page 329](#))
- :MODE (see [page 330](#))
- :SPI
 - :ASIZe (see [page 328](#))
- :UART
 - :BASE (see [page 332](#))
 - :COUNT
 - :ERRor (see [page 333](#))
 - :RESet (see [page 334](#))
 - :RXFRames (see [page 335](#))
 - :TXFRames (see [page 336](#))
 - :FRAMing (see [page 337](#))
- :SERial (see [page 126](#))
- :SINGle (see [page 127](#))
- :STATus (see [page 128](#))
- :STOP (see [page 129](#))
- :SYSTem (see [page 338](#))
 - :DATE (see [page 339](#))
 - :DSP (see [page 340](#))
 - :ERRor (see [page 341](#))
 - :LOCK (see [page 342](#))
 - :SETup (see [page 343](#))
 - :TIME (see [page 345](#))
- :TER (Trigger Event Register) (see [page 130](#))
- :TIMebase (see [page 346](#))
 - :MODE (see [page 348](#))
 - :POSition (see [page 349](#))
 - :RANGe (see [page 350](#))
 - :REFClock (see [page 351](#))
 - :REFerence (see [page 352](#))
 - :SCALe (see [page 353](#))

- :VERNier (see [page 354](#))
- :WINDow
 - :POSition (see [page 355](#))
 - :RANGe (see [page 356](#))
 - :SCALe (see [page 357](#))
- :TRIGger (see [page 358](#))
 - :HFReject (see [page 362](#))
 - :HOLDoff (see [page 363](#))
 - :MODE (see [page 364](#))
 - :NREJect (see [page 365](#))
 - :PATTern (see [page 366](#))
 - :SWEep (see [page 368](#))
 - :CAN (see [page 369](#))
 - :ACKnowledge (see [page 580](#))
 - :PATTern
 - :DATA (see [page 371](#))
 - :LENGth (see [page 372](#))
 - :ID (see [page 373](#))
 - :MODE (see [page 374](#))
 - :SAMPlepoint (see [page 375](#))
 - :SIGNal
 - :BAUDrate (see [page 376](#))
 - :DEFinition (see [page 581](#))
 - :SOURce (see [page 377](#))
 - :TRIGger (see [page 378](#))
 - :DURation (see [page 380](#))
 - :GREaterthan (see [page 381](#))
 - :LESSthan (see [page 382](#))
 - :PATTern (see [page 383](#))
 - :QUALifier (see [page 384](#))
 - :RANGe (see [page 385](#))
 - :EBURst (see [page 386](#))
 - :COUNT (see [page 387](#))
 - :IDLE (see [page 388](#))
 - :SLOPe (see [page 389](#))

- [:EDGE] (see [page 390](#))
 - :COUPling (see [page 391](#))
 - :LEVel (see [page 392](#))
 - :REJect (see [page 393](#))
 - :SLOPe (see [page 394](#))
 - :SOURce (see [page 395](#))
- :FLEXray (see [page 396](#))
 - :ERRor
 - :TYPE (see [page 397](#))
 - :FRAMe
 - :CCBase (see [page 399](#))
 - :CCRepetition (see [page 400](#))
 - :ID (see [page 401](#))
 - :TYPE (see [page 402](#))
 - :TIME
 - :CBASe (see [page 403](#))
 - :CREPetition (see [page 404](#))
 - :SEGMENT (see [page 405](#))
 - :SLOT (see [page 406](#))
 - :TRIGger (see [page 407](#))
- :GLITCh (see [page 408](#))
 - :GREATERthan (see [page 410](#))
 - :LESSthan (see [page 411](#))
 - :LEVel (see [page 412](#))
 - :POLarity (see [page 413](#))
 - :QUALifier (see [page 414](#))
 - :RANGe (see [page 415](#))
 - :SOURce (see [page 416](#))
- :HFReject (see [page 362](#))
- :HOLDoff (see [page 363](#))
- :IIC (see [page 417](#))
 - :PATTERn
 - :ADDRESS (see [page 418](#))
 - :DATA (see [page 419](#))
 - :DATa2 (see [page 420](#))

- :SOURce
 - :CLOCK (see [page 421](#))
 - :DATA (see [page 422](#))
- :TRIGger
 - :QUALifier (see [page 423](#))
 - [:TYPE] (see [page 424](#))
- :LIN (see [page 426](#))
 - :ID (see [page 427](#))
 - :SAMPLEpoint (see [page 428](#))
 - :SIGNal
 - :BAUDrate (see [page 429](#))
 - :DEFinition (see [page 582](#))
 - :SOURce (see [page 430](#))
 - :STANDARD (see [page 431](#))
 - :SYNCbreak (see [page 432](#))
 - :TRIGger (see [page 433](#))
- :MODE (see [page 364](#))
- :NREJect (see [page 365](#))
- :PATTERN (see [page 366](#))
- :SEQUence (see [page 434](#))
 - :COUNT (see [page 435](#))
 - :EDGE (see [page 436](#))
 - :FIND (see [page 437](#))
 - :PATTERN (see [page 438](#))
 - :RESET (see [page 439](#))
 - :TIMER (see [page 440](#))
 - :TRIGger (see [page 441](#))
- :SPI (see [page 442](#))
 - :CLOCK
 - :SLOPe (see [page 443](#))
 - :TIMEout (see [page 444](#))
 - :FRAMing (see [page 445](#))
 - :PATTERN
 - :DATA (see [page 446](#))
 - :WIDTH (see [page 447](#))

- :SOURce
 - :CLOCK (see [page 448](#))
 - :DATA (see [page 449](#))
 - :FRAME (see [page 450](#))
- :SWEep (see [page 368](#))
- :TV (see [page 451](#))
 - :LINE (see [page 452](#))
 - :MODE (see [page 453](#))
 - :POLarity (see [page 454](#))
 - :SOURce (see [page 455](#))
 - :STANdard (see [page 456](#))
 - :TVMode (see [page 584](#))
- :UART (see [page 457](#))
 - :BAUDrate (see [page 459](#))
 - :BITorder (see [page 460](#))
 - :BURSt (see [page 461](#))
 - :DATA (see [page 462](#))
 - :IDLE (see [page 463](#))
 - :PARity (see [page 464](#))
 - :QUALifier (see [page 466](#))
 - :POLarity (see [page 465](#))
 - :SOURce
 - :RX (see [page 467](#))
 - :TX (see [page 468](#))
 - :TYPE (see [page 469](#))
 - :WIDTh (see [page 470](#))
- :USB (see [page 471](#))
 - :SOURce
 - :DMINus (see [page 472](#))
 - :DPLus (see [page 473](#))
 - :SPEed (see [page 474](#))
 - :TRIGger (see [page 475](#))
- :VIEW (see [page 131](#))
- :WAVEform (see [page 476](#))
 - :BYTeorder (see [page 484](#))

- :COUNT (see [page 485](#))
- :DATA (see [page 486](#))
- :FORMat (see [page 488](#))
- :POINTs (see [page 489](#))
 - :MODE (see [page 491](#))
- :PREamble (see [page 493](#))
- :SEGmented
 - :COUNT (see [page 496](#))
 - :TTAG (see [page 497](#))
- :SOURce (see [page 498](#))
 - :SUBSource (see [page 502](#))
- :TYPE (see [page 503](#))
- :UNSigned (see [page 504](#))
- :VIEW (see [page 505](#))
- :XINcrement (see [page 506](#))
- :XORigin (see [page 507](#))
- :XREFerence (see [page 508](#))
- :YINcrement (see [page 509](#))
- :YORigin (see [page 510](#))
- :YREFerence (see [page 511](#))

Common Commands (IEEE 488.2)

- *CLS (see [page 73](#))
- *ESE (see [page 74](#))
- *ESR (see [page 76](#))
- *IDN (see [page 78](#))
- *LRN (see [page 79](#))
- *OPC (see [page 80](#))
- *OPT (see [page 81](#))
- *RCL (see [page 82](#))
- *RST (see [page 83](#))
- *SAV (see [page 86](#))
- *SRE (see [page 87](#))
- *STB (see [page 89](#))
- *TRG (see [page 91](#))
- *TST (see [page 92](#))
- *WAI (see [page 93](#))

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 621](#)). 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.

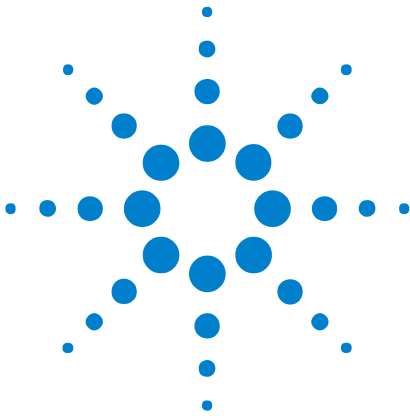
Infinity Representation The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

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.



9 Programming Examples

SICL Examples [638](#)

VISA Examples [647](#)

VISA COM Examples [692](#)

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 638

SICL Example in C

```

/*
 * 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.
*/
fprintf(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.
*/
fprintf(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.
*/
fprintf(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.
*/
fprintf(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.
*/
fprintf(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.
*/
fprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

/* TRIGGER_SOURCE - Selects the channel that actually produces the
* TV trigger. Any channel can be selected.
*/
fprintf(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.
*/
fprintf(id, ":TRIGGER:MODE EDGE\n");

/* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
* to either POSITIVE or NEGATIVE.
*/
fprintf(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.
     */
    fprintf(id, ":RUN\n");
    fprintf(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.
     */
    fprintf(id, ":BLANK CHANNEL1\n");
    fprintf(id, ":VIEW CHANNEL1\n");

    /* TIME_MODE (not executed in this example) - Set the time base
     * mode to MAIN, DELAYED, XY or ROLL.
     */
    fprintf(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.
     */
    fprintf(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.
     */
    fprintf(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. */
    iprintf(id, ":WAVEFORM:FORMAT BYTE\n");

    /* WAVE_POINTS - Sets the number of points to be transferred.
     * The number of time points available is returned by the
     * "ACQUIRE:POINTS?" query. This can be set to any binary
     * fraction of the total time points available.
     */
    iprintf(id, ":WAVEFORM:POINTS 1000\n");

    /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
     * settings returned in the form <preamble block><NL> where the
     * <preamble block> is:
     *
     * FORMAT      : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.

```

```

*   TYPE           : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
*   POINTS        : int32 - number of data points transferred.
*   COUNT         : int32 - 1 and is always 1.
*   XINCREMENT    : float64 - time difference between data points.
*   XORIGIN       : float64 - always the first data point in memory.
*   XREFERENCE    : int32 - specifies the data point associated
*                   with the x-origin.
*   YINCREMENT    : float32 - voltage difference between data points.
*   YORIGIN       : float32 - value of the voltage at center screen.
*   YREFERENCE    : int32 - data point where y-origin occurs.
*/
printf("Reading preamble\n");
ipromptf(id, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
/*
printf("Preamble FORMAT: %e\n", preamble[0]);
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");
}

```

9 Programming Examples

```
    }
    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 Examples

- ["VISA Example in C" on page 647](#)
- ["VISA Example in Visual Basic 6" on page 655](#)
- ["VISA Example in C#" on page 665](#)
- ["VISA Example in Visual Basic .NET" on page 678](#)

VISA Example in C

```

/*
 * 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 6

```

'
' Agilent VISA Example in Visual Basic 6
' -----
' 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

```

9 Programming Examples

```
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)

```

9 Programming Examples

```
' 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.
                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.

```

9 Programming Examples

```
    */
    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()
      End Try
    End Sub
  End Class
End Namespace

```

```

        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 6"](#) on page 692
- ["VISA COM Example in C#"](#) on page 701
- ["VISA COM Example in Visual Basic .NET"](#) on page 713

VISA COM Example in Visual Basic 6

```

'
' Agilent VISA COM Example in Visual Basic 6
' -----
' 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.

```

9 Programming Examples

```
' 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

```

9 Programming Examples

```
' enough time for the data acquisition process to collect data,
' and the results may not be accurate. An error value of 9.9E+37
' may be returned over the bus in this situation.
'
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

```

9 Programming Examples

```
' ":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.DoQueryIIEEEBlock(
    ":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 CHANnel");

// 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)
      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_IOException As FormattedIO488Class
Private m_strVisaAddress As String

' Constructor.
Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA COM IO object.
    OpenIo()

    ' Clear the interface.
    m_IOException.IO.Clear()
End Sub

Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    m_IOException.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
            Console.Write(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

```

9 Programming Examples

```
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, 635
+9.9E+37, measurement error, 248

Numerics

0 (zero) values in waveform data, 486
1 (one) values in waveform data, 486
10 MHz REF BNC, enabling/disabling, 351
10 MHz reference signal, 140

A

AC coupling, trigger edge, 391
AC input coupling for specified channel, 168
accumulate activity, 97
acknowledge, 580
ACQUIRE commands, 132
acquire data, 105, 145
acquire mode on autoscale, 101
acquire reset conditions, 83
acquire sample rate, 144
acquired data points, 139
acquisition anti-alias control, 134
acquisition count, 136
acquisition mode, 132, 138, 503
acquisition type, 132, 145
active edges, 97
active printer, 225
activity logic levels, 97
activity on digital channels, 97
add function, 498
ADDRESS commands, 514
address field size, IIC serial decode, 328
address, IIC trigger pattern, 418
AER (Arm Event Register), 98, 116, 118, 609
ALB waveform data format, 309
alphabetical list of commands, 513
amplitude, vertical, 276
analog channel coupling, 168
analog channel display, 169
analog channel impedance, 170
analog channel input, 548
analog channel inversion, 171
analog channel labels, 172, 196
analog channel offset, 173
analog channel range, 179
analog channel scale, 180
analog channel source for glitch, 416
analog channel units, 181
analog channels only oscilloscopes, 4
analog probe attenuation, 174

analog probe sensing, 549
analog probe skew, 176, 547
angle brackets, 64
annotate channels, 172
anti-alias control, 134
AREA commands, 514
area for hardcopy print, 224
area for saved image, 301
Arm Event Register (AER), 98, 116, 118, 609
arrange waveforms, 551
ASCII format, 488
ASCII format for data transfer, 478
ASCII string, quoted, 64
ASCII waveform data format, 309
assign channel names, 172
attenuation factor (external trigger) probe, 204
attenuation for oscilloscope probe, 174
AUT option for probe sense, 549, 554
auto trigger sweep mode, 358
automatic measurements constants, 174
automatic probe type detection, 549, 554
autoscale, 99
autoscale acquire mode, 101
autoscale channels, 102
average value measurement, 277
averaging acquisition type, 133, 478

B

bandwidth filter limits, 202
bandwidth filter limits to 20 MHz, 167
base value measurement, 278
basic instrument functions, 71
Bat On bit, 109, 111
baud rate, 376, 429, 459
BAUDrate commands, 514
begin acquisition, 105, 125, 127
binary block data, 64, 343, 486
BINary waveform data format, 309
bit order, 460
bit selection command, bus, 149
bit weights, 76
bitmap display, 193
bits in Service Request Enable Register, 88
bits in Standard Event Status Enable Register, 75
bits in Status Byte Register, 90
bits selection command, bus, 150
blank, 103
block data, 64, 79, 193, 343
BMP (bitmap) hardcopy format, 559
braces, 63
burst, minimum time before next, 388

bus bit selection command, 149
bus bits selection commands, 150
bus clear command, 152
BUS commands, 147
bus commands, 148
bus display, 153
bus label command, 154
bus mask command, 155
BUSDoctor commands, 515
button disable, 342
BWLIMIT commands, 515
byte format for data transfer, 478, 488
BYTeorder, 484

C

C#, VISA COM example, 701
C#, VISA example, 665
C, SICL library example, 638
C, VISA library example, 647
CAL PROTECT switch, 156, 161
calculating preshoot of waveform, 265
calculating the waveform overshoot, 261
calibrate, 157, 158, 161, 163
CALibrate commands, 156
calibrate date, 157
calibrate introduction, 156
calibrate label, 158
calibrate start, 159
calibrate status, 160
calibrate switch, 161
calibrate temperature, 162
calibrate time, 163
CALibrate commands, 156
calibrate date, 157
calibrate introduction, 156
calibrate label, 158
calibrate start, 159
calibrate status, 160
calibrate switch, 161
calibrate temperature, 162
calibrate time, 163
CAN, 371
CAN acknowledge, 375, 580
CAN baud rate, 376
CAN commands, 515
CAN frame counters, reset, 320
CAN id pattern, 373
CAN signal definition, 581
CAN source, 377
CAN trigger, 372, 378
CAN trigger commands, 369
CAN trigger pattern id mode, 374
CAN triggering, 358
capture data, 105
CDISplay, 104
center frequency set, 211, 212
center of screen, 511
center reference, 352
center screen, vertical value at, 214
channel, 131, 172, 544, 546
CHANnel commands, 164, 165

Index

channel coupling, 168
channel display, 169
channel input impedance, 170
channel inversion, 171
channel label, 172, 545
channel labels, 195, 196
channel numbers, 551
channel overload, 178
channel probe ID, 205
channel protection, 178
channel reset conditions, 83
channel selected to produce trigger, 416, 455
channel signal type, 177
channel skew for oscilloscope probe, 176, 547
channel status, 128, 551
channel threshold, 546
channel vernier, 182
channel, stop displaying, 103
channels to autoscale, 102
channels, how autoscale affects, 99
characters to display, 340
classes of input signals, 221
classifications, command, 616
clear, 192
clear bus command, 152
CLEAR commands, 516
clear cumulative edge variables, 544
clear display, 104
clear markers, 250, 564
clear measurement, 250, 564
clear message queue, 73
clear screen, 552
clear status, 73
clear waveform area, 190
clipped high waveform data value, 486
clipped low waveform data value, 486
clock, 421, 443, 444, 448
CLOCK commands, 516
CLS (Clear Status), 73
CME (Command Error) status bit, 75, 77
CMOS threshold voltage for digital channels, 189, 546
CMOS trigger threshold voltage, 583
code, *RST, 85
code, :ACQUIRE:COMPLETE, 135
code, :ACQUIRE:SEGMENTED, 142
code, :ACQUIRE:TYPE, 146
code, :AUTOSCALE, 100
code, :CHANNEL:LABEL, 172
code, :CHANNEL:PROBE, 174
code, :CHANNEL:RANGE, 179
code, :DIGITIZE, 105
code, :DISPLAY:DATA, 194
code, :DISPLAY:LABEL, 195
code, :DISPLAY:ORDER, 551
code, :MEASURE:PERIOD, 270
code, :MEASURE:TEDGE, 273
code, :POD:THRESHOLD, 290
code, :RUN:/STOP, 125
code, :SYSTEM:SETUP, 343
code, :TIMEBASE:DELAY, 579
code, :TIMEBASE:MODE, 348

code, :TIMEBASE:RANGE, 350
code, :TIMEBASE:REFERENCE, 352
code, :TRIGGER:MODE, 364
code, :TRIGGER:SLOPE, 394
code, :TRIGGER:SOURC, 395
code, :VIEW and :BLANK, 131
code, :WAVEFORM, 499
code, :WAVEFORM:DATA, 486
code, :WAVEFORM:POINTS, 490
code, :WAVEFORM:PREEMBLE, 494
code, :WAVEFORM:SEGMENTED, 142
code, SICL library example in C, 638
code, VISA COM library example in C#, 701
code, VISA COM library example in Visual Basic .NET, 713
code, VISA COM library example in Visual Basic 6, 692
code, VISA library example in C, 647
code, VISA library example in C#, 665
code, VISA library example in Visual Basic .NET, 678
code, VISA library example in Visual Basic 6, 655
colon, root commands prefixed by, 96
color palette for hardcopy, 229
color palette for image, 305
Comma Separated Values (CSV) hardcopy format, 559
Comma Separated Values (CSV) waveform data format, 309
command classifications, 616
command errors detected in Standard Event Status, 77
command header, 618
command headers, common, 620
command headers, compound, 619
command headers, simple, 619
command strings, valid, 617
command tree, 621
commands by subsystem, 67
commands in alphabetical order, 513
commands quick reference, 21
commands sent over interface, 71
commands, more about, 615
commands, obsolete and discontinued, 539
common (*) commands, 68, 69, 71
common command headers, 620
completion criteria for an acquisition, 135, 136
compound command headers, 619
compound header, 633
computer control examples, 637
conditions for external trigger, 200
conditions, reset, 83
configurations, oscilloscope, 79, 82, 86, 343
connect sampled data points, 550
constants for making automatic measurements, 174
constants for scaling display factors, 174
constants for setting trigger levels, 174
copy display, 124
core commands, 616
count, 435, 485

Nth edge of burst, 387
COUNT commands, 516
count values, 136
counter, 251
coupling, 391
COUPLING commands, 517
coupling for channels, 168
CSV (Comma Separated Values) hardcopy format, 559
CSV (Comma Separated Values) waveform data format, 309
cumulative edge activity, 544
current logic levels on digital channels, 97
current oscilloscope configuration, 79, 82, 86, 343
current probe, 181, 209
cursor mode, 234
cursor position, 235, 237, 239, 240, 242
cursor readout, 565, 569, 570
cursor reset conditions, 83
cursor source, 236, 238
cursor time, 565, 569, 570
cursors track measurements, 269
cursors, how autoscale affects, 99
cursors, X1, X2, Y1, Y2, 233
cycle base, FLEXray time trigger, 403
cycle count base, FLEXray frame trigger, 399
cycle count repetition, FLEXray frame trigger, 400
cycle measured, 257
cycle repetition, FLEXray time trigger, 404
cycle time, 263

D

D- source, 472
D+ source, 473
data, 371, 419, 422, 446, 449, 486
data 2, 420
DATA commands, 517
data conversion, 478
data displayed, 193
data format for transfer, 478
data output order, 484
data pattern length, 372
data pattern width, 447
data point index, 508
data points, 139
data required to fill time buckets, 135
data structures, status reporting, 595
data transfer, 193
data, erasing, 104
data, saving and recalling, 190
DATE commands, 517
date, calibration, 157
date, system, 339
dB versus frequency, 211
DC coupling for edge trigger, 391
DC input coupling for specified channel, 168
dc RMS measured on waveform, 282
DDE (Device Dependent Error) status bit, 75, 77

decision chart, status reporting, [613](#)
 default conditions, [83](#)
 define channel labels, [172](#)
 define glitch trigger, [414](#)
 define logic thresholds, [546](#)
 define measurement, [253](#)
 define measurement source, [270](#)
 define trigger, [366](#), [381](#), [382](#), [383](#), [385](#), [415](#),
[436](#)
 defined as, [63](#)
 definite-length block response data, [64](#)
 DEFinition commands, [518](#)
 DELay commands, [518](#)
 delay measured to calculate phase, [264](#)
 delay measurement, [253](#)
 delay measurements, [272](#)
 delay parameters for measurement, [255](#)
 delay, how autoscale affects, [99](#)
 delayed time base, [348](#), [579](#)
 delayed time base mode, how autoscale
 affects, [99](#)
 delayed window horizontal scale, [357](#)
 delta time, [565](#)
 delta voltage measurement, [574](#)
 delta X cursor, [233](#)
 delta Y cursor, [233](#)
 DeskJet, [557](#)
 destination, [198](#)
 detecting probe types, [549](#), [554](#)
 device for hardcopy, [557](#)
 device-defined error queue clear, [73](#)
 differential signal type, [177](#), [206](#)
 differentiate math function, [137](#), [211](#), [215](#), [498](#)
 DIFFerentiate source for function, [219](#)
 digital channel commands, [183](#), [185](#), [186](#), [187](#),
[189](#)
 digital channel data, [478](#)
 digital channel labels, [196](#)
 digital channel order, [551](#)
 digital channel source for glitch trigger, [416](#)
 digital channels, [4](#)
 digital channels, activity and logic levels on, [97](#)
 digital channels, groups of, [287](#), [288](#), [290](#)
 DIGital commands, [183](#)
 digital pod, stop displaying, [103](#)
 digital reset conditions, [83](#)
 digitize channels, [105](#)
 digits, [64](#)
 disable anti-alias mode, [137](#)
 disable front panel, [342](#)
 disable function, [555](#)
 disabling calibration, [161](#)
 disabling channel display, [169](#)
 disabling status register bits, [74](#), [87](#)
 discontinued and obsolete commands, [539](#)
 display channel labels, [195](#)
 display clear, [192](#)
 DISPLAY commands, [190](#), [518](#)
 display commands introduction, [190](#)
 display connect, [550](#)
 display data, [193](#)
 display date, [339](#)

display factors scaling, [174](#)
 display for channels, [169](#)
 display frequency span, [220](#)
 display measurements, [248](#), [269](#)
 display order, [551](#)
 display persistence, [197](#)
 display reference, [349](#), [352](#)
 display reset conditions, [83](#)
 display serial number, [126](#)
 display source, [198](#)
 display vectors, [199](#)
 display wave position, [551](#)
 display, clearing, [104](#)
 display, oscilloscope, [185](#), [197](#), [198](#), [199](#), [213](#),
[288](#), [340](#)
 display, serial decode bus, [323](#)
 displaying a baseline, [368](#)
 displaying unsynchronized signal, [368](#)
 driver, printer, [562](#)
 DSO models, [4](#)
 duplicate mnemonics, [633](#)
 duration, [381](#), [382](#), [385](#)
 duration for glitch trigger, [410](#), [411](#), [415](#)
 duration pattern, [383](#)
 duration qualifier, trigger, [381](#), [382](#), [384](#)
 DURation trigger commands, [380](#)
 duration triggering, [358](#)
 duty cycle measurement, [248](#), [257](#)

E

EBURst trigger commands, [386](#)
 ECL channel threshold, [546](#)
 ECL threshold voltage for digital channels, [189](#)
 ECL trigger threshold voltage, [583](#)
 edge, [436](#)
 edge activity, [544](#)
 edge counter, [435](#)
 Nth edge of burst, [387](#)
 edge coupling, [391](#)
 edge define, [366](#), [436](#)
 edge fall time, [258](#)
 edge parameter for delay measurement, [255](#)
 edge preshoot measured, [265](#)
 edge rise time, [267](#)
 edge slope, [394](#)
 edge source, [395](#)
 EDGE trigger commands, [390](#)
 edge triggering, [358](#)
 edges (activity) on digital channels, [97](#)
 edges in measurement, [253](#)
 ellipsis, [64](#)
 enable channel labels, [195](#)
 enabling calibration, [161](#)
 enabling channel display, [169](#)
 enabling status register bits, [74](#), [87](#)
 end of string (EOS) terminator, [618](#)
 end of text (EOT) terminator, [618](#)
 end or identify (EOI), [618](#)
 enter pattern, [366](#)
 EOI (end or identify), [618](#)
 EOS (end of string) terminator, [618](#)
 EOT (end of text) terminator, [618](#)
 Epson, [557](#)
 equivalent-time acquisition mode, [133](#), [138](#)
 erase data, [104](#), [192](#)
 erase functions, [104](#)
 erase measurements, [564](#)
 erase screen, [552](#)
 ERRor commands, [519](#)
 error frame count (CAN), [318](#)
 error frame count (UART), [333](#)
 error messages, [341](#), [585](#)
 error number, [341](#)
 error queue, [341](#), [606](#)
 error, measurement, [248](#)
 ESB (Event Status Bit), [88](#), [90](#)
 ESE (Standard Event Status Enable
 Register), [74](#), [605](#)
 ESR (Standard Event Status Register), [76](#), [604](#)
 event status conditions occurred, [90](#)
 Event Status Enable Register (ESE), [74](#), [605](#)
 Event Status Register (ESR), [76](#), [130](#), [604](#)
 example code, *RST, [85](#)
 example code, :ACQUIRE:COMPLETE, [135](#)
 example code, :ACQUIRE:SEGMENTED, [142](#)
 example code, :ACQUIRE:TYPE, [146](#)
 example code, :AUTOSCALE, [100](#)
 example code, :CHANNEL:LABEL, [172](#)
 example code, :CHANNEL:PROBE, [174](#)
 example code, :CHANNEL:RANGE, [179](#)
 example code, :DIGITIZE, [105](#)
 example code, :DISPLAY:DATA, [194](#)
 example code, :DISPLAY:LABEL, [195](#)
 example code, :DISPLAY:ORDER, [551](#)
 example code, :MEASURE:PERIOD, [270](#)
 example code, :MEASURE:TEDGE, [273](#)
 example code, :POD:THRESHOLD, [290](#)
 example code, :RUN:/STOP, [125](#)
 example code, :SYSTEM:SETUP, [343](#)
 example code, :TIMEBASE:DELAY, [579](#)
 example code, :TIMEBASE:MODE, [348](#)
 example code, :TIMEBASE:RANGE, [350](#)
 example code, :TIMEBASE:REFERENCE, [352](#)
 example code, :TRIGGER:MODE, [364](#)
 example code, :TRIGGER:SLOPE, [394](#)
 example code, :TRIGGER:SOURCE, [395](#)
 example code, :VIEW and :BLANK, [131](#)
 example code, :WAVEFORM, [499](#)
 example code, :WAVEFORM:DATA, [486](#)
 example code, :WAVEFORM:POINTS, [490](#)
 example code, :WAVEFORM:PREAmble, [494](#)
 example code, :WAVEFORM:SEGMENTED, [142](#)
 example programs, [4](#), [637](#)
 EXE (Execution Error) status bit, [75](#), [77](#)
 execution error detected in Standard Event
 Status, [77](#)
 exponential notation, [63](#)
 external glitch trigger source, [416](#)
 external range, [208](#)
 external trigger, [200](#), [203](#), [204](#), [395](#), [553](#)
 EXTERNAL trigger commands, [200](#)
 external trigger input impedance, [203](#), [553](#)
 EXTERNAL trigger level, [392](#)

Index

external trigger overload, 207
external trigger probe attenuation factor, 204
external trigger probe ID, 205
external trigger probe sensing, 554
external trigger protection, 207
external trigger signal type, 206
EXternal trigger source, 395
external trigger units, 209

F

FACTors commands, 519
failure, self test, 92
fall time measurement, 248, 258
falling edge, 366, 436
Fast Fourier Transform (FFT) functions, 211, 212, 217, 219, 220, 221
FF values in waveform data, 486
FFT (Fast Fourier Transform) functions, 211, 212, 217, 219, 220, 221
FFT (Fast Fourier Transform) operation, 215, 498
FFT math function, 137
FILEname commands, 519
filename for hardcopy, 558
filename for recall, 293
filename for save, 299
filter for frequency reject, 393
filter for high frequency reject, 362
filter for noise reject, 365
filter used to limit bandwidth, 167, 202
filters to Fast Fourier Transforms, 221
find stage in sequence trigger, 437
fine horizontal adjustment (vernier), 354
fine vertical adjustment (vernier), 182
finish pending device operations, 80
first point displayed, 508
FLATop window for amplitude measurements, 221
FLEXray commands, 519
FlexRay frame counters, reset, 325
FLEXray trigger, 407
FLEXray trigger commands, 396
FlexRay triggering, 358
format, 488, 493
FORMat commands, 520
format for block data, 79
format for generic video, 452, 456
format for hardcopy, 556, 559
format for image, 303
format for waveform data, 309
formfeed for hardcopy, 222, 227
frame, 450
FRAME commands, 520
frame counters (CAN), error, 318
frame counters (CAN), overload, 319
frame counters (CAN), reset, 320
frame counters (CAN), total, 321
frame counters (FlexRay), null, 324, 326
frame counters (FlexRay), reset, 325
frame counters (FlexRay), total, 327
frame counters (UART), error, 333

frame counters (UART), reset, 334
frame counters (UART), Rx frames, 335
frame counters (UART), Tx frames, 336
frame ID, FLEXray frame trigger, 401
frame type, FLEXray frame trigger, 402
framing, 445
FRAMing commands, 520
frequency measurement, 248, 259
frequency resolution, 221
frequency span of display, 220
frequency versus dB, 211
front panel mode, 368
front panel Single key, 127
front panel Stop key, 129
front-panel lock, 342
full-scale horizontal time, 350, 356
full-scale vertical axis defined, 216
function, 131, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 555
FUNction commands, 210
function memory, 128
function turned on or off, 555
functions, 498
functions, erasing, 104

G

general trigger commands, 361
GENeric, 452, 456
generic video format, 452, 456
glitch duration, 415
glitch qualifier, 414
glitch source, 416
GLITCh trigger commands, 408
glitch trigger duration, 410
glitch trigger polarity, 413
glitch trigger source, 410
graphics, 193
graticule area for hardcopy print, 224
graticule area for saved image, 301
graticule colors, invert for hardcopy, 228, 561
graticule colors, invert for image, 304
graticule data, 193
grayscale palette for hardcopy, 229
grayscale palette for image, 305
grayscale on hardcopy, 560
greater than qualifier, 414
greater than time, 381, 385, 410, 415
GREaterthan commands, 521
groups of digital channels, 287, 288, 290, 546

H

HANNing window for frequency resolution, 221
hardcopy, 124, 222
HARDcopy commands, 222
hardcopy device, 557
hardcopy factors, 226, 302
hardcopy filename, 558
hardcopy format, 556, 559

hardcopy formfeed, 227
hardcopy grayscale, 560
hardcopy invert graticule colors, 228, 561
hardcopy palette, 229
hardcopy print, area, 224
hardcopy printer driver, 562
hardware event condition register, 109
Hardware Event Condition Register (:HWERegister:CONDition), 109
Hardware Event Condition Register (:OPERRegister:CONDition), 611
Hardware Event Enable Register (HWEEnable), 107
hardware event event register, 111
Hardware Event Event Register (:HWERegister[:EVENT]), 111, 610
header, 618
high-frequency reject filter, 362, 393
hold until operation complete, 80
holdoff time, 363
holes in waveform data, 486
horizontal adjustment, fine (vernier), 354
horizontal position, 355
horizontal scale, 353, 357
horizontal scaling, 493
horizontal time, 350, 356, 565
HWEEnable (Hardware Event Enable Register), 107
HWERegister:CONDition (Hardware Event Condition Register), 109, 611
HWERegister[:EVENT] (Hardware Event Event Register), 111, 610

I

I1080L50HZ, 452, 456
I1080L60HZ, 452, 456
ID commands, 521
id mode, 374
identification number, 78
identification of options, 81
identifier, 373
identifier, LIN, 427
idle, 388
IDLE commands, 521
idle until operation complete, 80
IDN (Identification Number), 78
IEEE 488.2 standard, 71
IGColors commands, 522
IIC address, 418
IIC clock, 421
IIC commands, 521
IIC data, 419, 422
IIC data 2, 420
IIC serial decode address field size, 328
IIC trigger commands, 417
IIC trigger qualifier, 423
IIC trigger type, 424
IIC triggering, 358
IMAGe commands, 522
image format, 303
image invert graticule colors, 304

image memory, 128, 198
 image palette, 305
 image, recall, 294
 image, save, 300
 image, save with inksaver, 304
 impedance, 170
 IMPedance commands, 522
 impedance for external trigger input, 203, 553
 infinity representation, 635
 initialize, 83
 initialize label list, 196
 initiate acquisition, 105
 inksaver, save image with, 304
 input, 203, 553
 input coupling for channels, 168
 input impedance for channels, 170, 548
 input impedance for external trigger, 203, 553
 input inversion for specified channel, 171
 insert label, 172
 installed options identified, 81
 instruction header, 618
 instrument number, 78
 instrument options identified, 81
 instrument requests service, 90
 instrument serial number, 126
 instrument settings, 222
 instrument type, 78
 integrate math function, 211, 215, 498
 INTegrate source for function, 219
 INTERN files, 198
 internal low-pass filter, 167, 202
 introduction to :ACQuire commands, 132
 introduction to :BUS commands, 148
 introduction to :CALibrate commands, 156
 introduction to :CHANnel commands, 165
 introduction to :DIGital commands, 183
 introduction to :DISPlay commands, 190
 introduction to :EXternal commands, 200
 introduction to :FUNCtion commands, 211
 introduction to :HARDcopy commands, 222
 introduction to :MARKer commands, 233
 introduction to :MEASure commands, 248
 introduction to :POD commands, 287
 introduction to :RECall commands, 292
 introduction to :SAVE commands, 298
 introduction to :SBUS commands, 312
 introduction to :SYSTem commands, 338
 introduction to :TIMebase commands, 347
 introduction to :TRIGger commands, 358
 introduction to :WAVEform commands, 478
 introduction to common (*) commands, 71
 introduction to root (:) commands, 96
 invert graticule colors for hardcopy, 228, 561
 invert graticule colors for image, 304
 inverting input for channels, 171

K

key disable, 342
 key press detected in Standard Event Status Register, 77
 knob disable, 342

known state, 83

L

label, 186, 545
 label command, bus, 154
 LABEL commands, 522
 label list, 172, 196
 labels, 172, 195, 196
 labels to store calibration information, 158
 labels, specifying, 190
 LaserJet, 557
 leakage into peak spectrum, 221
 learn string, 79, 343
 least significant byte first, 484
 left reference, 352
 legal values for channel offset, 173
 legal values for frequency span, 220
 legal values for offset, 214
 LENGth commands, 522
 length for waveform data, 310
 less than qualifier, 414
 less than time, 382, 385, 411, 415
 LESSthan commands, 522
 LEVel commands, 522
 level for trigger voltage, 392, 412
 LF coupling, 391
 license information, 81
 limits for line number, 452
 LIN acknowledge, 428
 LIN baud rate, 429
 LIN identifier, 427
 LIN serial decode bus parity bits, 329
 LIN source, 430
 LIN standard, 431
 LIN sync break, 432
 LIN trigger, 433
 LIN trigger commands, 426
 LIN trigger definition, 582
 LIN triggering, 358
 line glitch trigger source, 416
 line number for TV trigger, 452
 line terminator, 63
 LINE trigger level, 392
 LINE trigger source, 395
 list of channel labels, 196
 load utilization (CAN), 322
 local lockout, 342
 lock, 342
 lockout message, 342
 logic level activity, 544
 long form, 618
 lower threshold, 263
 lower threshold voltage for measurement, 563
 lowercase characters in commands, 617
 low-frequency reject filter, 393
 low-pass filter used to limit bandwidth, 167, 202
 LRN (Learn Device Setup), 79
 lsbfirst, 484

M

magnitude of occurrence, 274
 main sweep range, 355
 main time base mode, 348
 making measurements, 248
 MAN option for probe sense, 549, 554
 manual cursor mode, 234
 MARKer commands, 232
 marker mode, 240
 marker position, 241
 marker readout, 569, 570
 marker set for voltage measurement, 575, 576
 marker sets start time, 566
 marker time, 565
 markers for delta voltage measurement, 574
 markers track measurements, 269
 markers, command overview, 233
 markers, mode, 234
 markers, time at start, 570
 markers, time at stop, 569
 markers, X delta, 239
 markers, X1 position, 235
 markers, X1Y1 source, 236
 markers, X2 position, 237
 markers, X2Y2 source, 238
 markers, Y delta, 242
 markers, Y1 position, 240
 markers, Y2 position, 241
 mask, 74, 87, 366, 383
 mask command, bus, 155
 master summary status bit, 90
 math function, stop displaying, 103
 math operations, 211
 MAV (Message Available), 73, 88, 90
 maximum duration, 381, 382, 411
 maximum position, 349
 maximum range for delayed window, 356
 maximum scale for delayed window, 357
 maximum vertical value measurement, 279
 maximum vertical value, time of, 285, 567
 MEASure commands, 243
 measure overshoot, 261
 measure period, 263
 measure phase between channels, 264
 measure preshoot, 265
 measure start voltage, 575
 measure stop voltage, 576
 measure value at a specified time, 283
 measure value at top of waveform, 284
 measurement error, 248
 measurement setup, 248, 270
 measurement source, 270
 measurements, average value, 277
 measurements, base value, 278
 measurements, clear, 250, 564
 measurements, command overview, 248
 measurements, counter, 251
 measurements, dc RMS, 282
 measurements, definition setup, 253
 measurements, delay, 255
 measurements, duty cycle, 257

Index

measurements, fall time, [258](#)
measurements, frequency, [259](#)
measurements, how autoscale affects, [99](#)
measurements, lower threshold level, [563](#)
measurements, maximum vertical value, [279](#)
measurements, maximum vertical value, time of, [285, 567](#)
measurements, minimum vertical value, [280](#)
measurements, minimum vertical value, time of, [286, 568](#)
measurements, overshoot, [261](#)
measurements, period, [263](#)
measurements, phase, [264](#)
measurements, preshoot, [265](#)
measurements, pulse width, negative, [260](#)
measurements, pulse width, positive, [266](#)
measurements, resetting, [104](#)
measurements, rise time, [267](#)
measurements, show, [269](#)
measurements, source channel, [270](#)
measurements, standard deviation, [268](#)
measurements, start marker time, [569](#)
measurements, stop marker time, [570](#)
measurements, thresholds, [566](#)
measurements, time between start and stop markers, [565](#)
measurements, time between trigger and edge, [272](#)
measurements, time between trigger and vertical value, [274](#)
measurements, time between trigger and voltage level, [571](#)
measurements, upper threshold value, [573](#)
measurements, vertical amplitude, [276](#)
measurements, vertical peak-to-peak, [281](#)
measurements, voltage difference, [574](#)
memory setup, [86, 343](#)
merge, [113](#)
message available bit, [90](#)
message available bit clear, [73](#)
message displayed, [90](#)
message error, [585](#)
message queue, [603](#)
messages ready, [90](#)
midpoint of thresholds, [263](#)
minimum duration, [381, 382, 385, 410](#)
minimum vertical value measurement, [280](#)
minimum vertical value, time of, [286, 568](#)
mixed-signal oscilloscopes, [4](#)
mnemonics, duplicate, [633](#)
mode, [138, 145, 234, 348, 453](#)
MODE commands, [524](#)
mode, serial decode, [330](#)
model number, [78](#)
models, oscilloscope, [3](#)
modes for triggering, [364](#)
monochrome palette for image, [305](#)
most significant byte first, [484](#)
move, [211](#)
move cursors, [569, 570](#)
msbfirst, [484](#)
MSG (Message), [88, 90](#)

MSSO models, [4](#)
MSS (Master Summary Status), [90](#)
multiple commands, [633](#)
multiply math function, [211, 215, 498](#)

N

name channels, [172](#)
name list, [196](#)
negative glitch trigger polarity, [413](#)
negative pulse width, [260](#)
negative slope, [394, 443](#)
 Nth edge in burst, [389](#)
negative TV trigger polarity, [454](#)
new line (NL) terminator, [63, 618](#)
NL (new line) terminator, [63, 618](#)
noise reject filter, [365](#)
non-core commands, [616](#)
non-interlaced GENeric mode, [456](#)
non-volatile memory, label list, [154, 186, 196](#)
normal acquisition type, [132, 478](#)
normal trigger sweep mode, [358](#)
notices, [2](#)
NR1 number format, [63](#)
NR3 number format, [63](#)
Nth edge in a burst idle, [388](#)
Nth edge in burst slope, [389](#)
Nth edge of burst counter, [387](#)
NTSC, [452, 456](#)
null frame count (FlexRay), [324](#)
NULL string, [340](#)
number format, [63](#)
number of points, [139, 489, 491](#)
number of time buckets, [489, 491](#)
nwidth, [260](#)

O

obsolete and discontinued commands, [539](#)
obsolete commands, [616](#)
occurrence reported by magnitude, [571](#)
offset, [211](#)
OFFSet commands, [525](#)
offset value for channel voltage, [173](#)
offset value for selected function, [214](#)
one values in waveform data, [486](#)
OPC (Operation Complete) command, [80](#)
OPC (Operation Complete) status bit, [75, 77](#)
OPEE (Operation Status Enable Register), [114](#)
operating configuration, [79, 343](#)
operating state, [86](#)
operation complete, [80](#)
operation status condition register, [116](#)
Operation Status Condition Register
(:OPERRegister:CONDition), [116, 608](#)
operation status conditions occurred, [90](#)
Operation Status Enable Register (OPEE), [114](#)
operation status event register, [118](#)
Operation Status Event Register
(:OPERRegister[:EVENT]), [118, 607](#)
operation, math, [211](#)

operations for function, [215](#)
OPERRegister:CONDition (Operation Status
Condition Register), [116, 608](#)
OPERRegister[:EVENT] (Operation Status Event
Register), [118, 607](#)
OPT (Option Identification), [81](#)
optional syntax terms, [63](#)
options, [81](#)
order of digital channels on display, [551](#)
order of output, [484](#)
oscilloscope external trigger, [200](#)
oscilloscope models, [3](#)
oscilloscope rate, [144](#)
output messages ready, [90](#)
output queue, [80, 602](#)
output queue clear, [73](#)
output sequence, [484](#)
overlapped commands, [636](#)
overload, [178, 207](#)
Overload Event Enable Register (OVL), [120](#)
Overload Event Register (OVLr), [122](#)
overload frame count (CAN), [319](#)
overload protection, [120, 122](#)
overshoot of waveform, [261](#)
overvoltage, [178, 207](#)
OVL (Overload Event Enable Register), [120](#)
OVLr (Overload Event Register), [122](#)
OVLr bit, [111, 116, 118](#)

P

P1080L24HZ, [452, 456](#)
P1080L25HZ, [452, 456](#)
P480L60HZ, [452, 456](#)
P720L60HZ, [452, 456](#)
PAL, [452, 456](#)
PALette commands, [525](#)
palette for hardcopy, [229](#)
palette for image, [305](#)
PAL-M, [452, 456](#)
parameters for delay measurement, [255](#)
parametric measurements, [248](#)
parity, [464](#)
parity bits, LIN serial decode bus, [329](#)
PARity commands, [525](#)
parser, [96, 633](#)
pass, self test, [92](#)
path information, recall, [295](#)
path information, save, [306](#)
pattern, [366, 371, 373, 383, 418, 419, 420, 438, 446](#)
pattern and edge, [366](#)
PATtern commands, [525](#)
pattern duration, [381, 382, 410, 411](#)
pattern length, [372](#)
pattern trigger, [366](#)
pattern triggering, [358](#)
pattern width, [447](#)
peak detect, [145](#)
peak detect acquisition type, [133, 478](#)
peaks, [211](#)
peak-to-peak vertical value measurement, [281](#)

pending operations, 80
 percent of waveform overshoot, 261
 percent thresholds, 253
 period measured to calculate phase, 264
 period measurement, 248, 263
 persistence, waveform, 190, 197
 phase measured between channels, 264
 phase measurements, 272
 pixel memory, 198
 pixel memory, saving display to, 113
 PLL Locked bit, 109, 116
 pod, 287, 288, 289, 290, 498, 546
 POD commands, 287
 pod, stop displaying, 103
 points, 139, 489, 491
 POINTs commands, 526
 points in waveform data, 478
 polarity, 454, 465
 POLarity commands, 526
 polarity for glitch trigger, 413
 PON (Power On) status bit, 75, 77
 position, 187, 237, 349, 355
 POSition commands, 526
 position cursors, 569, 570
 position in delayed view, 355
 position waveforms, 551
 positive glitch trigger polarity, 413
 positive pulse width, 266
 positive slope, 394, 443
 Nth edge in burst, 389
 positive TV trigger polarity, 454
 positive width, 266
 preamble data, 478, 493
 predefined logic threshold, 546
 predefined threshold voltages, 583
 preset conditions, 83
 preshoot measured on waveform, 265
 previously stored configuration, 82
 print command, 124
 print job, start, 231
 print query, 577
 printer, 557
 printer driver for hardcopy, 562
 printer hardcopy format, 559
 printer, active, 225
 printing, 222
 printing in grayscale, 560
 probe, 392
 probe attenuation affects channel voltage
 range, 179
 probe attenuation factor (external trigger), 204
 probe attenuation factor for selected
 channel, 174
 PROBe commands, 526
 probe ID, 175, 205
 probe sense for oscilloscope, 549, 554
 probe skew value, 176, 547
 program data, 618
 program data syntax rules, 620
 program message, 71
 program message syntax, 617
 program message terminator, 618

programming examples, 4, 637
 protecting against calibration, 161
 protection, 120, 122, 178, 207
 PROTection commands, 526
 pulse width, 260, 266
 pulse width duration trigger, 410, 411, 415
 pulse width measurement, 248
 pulse width trigger, 365
 pulse width trigger level, 412
 pulse width triggering, 358
 PWD commands, 527
 pwidth, 266

Q

qualifier, 415
 QUALifier commands, 527
 qualifier, trigger duration, 381, 382, 384
 query error detected in Standard Event
 Status, 77
 query return values, 635
 query setup, 222, 233, 248, 343
 query subsystem, 148, 183, 211
 querying setup, 165
 querying the subsystem, 358
 queues, clearing, 612
 quick reference, commands, 21
 quoted ASCII string, 64
 QYE (Query Error) status bit, 75, 77

R

range, 211, 356
 RANGE commands, 527
 range for channels, 179
 range for duration trigger, 385
 range for external trigger, 208
 range for full-scale vertical axis, 216
 range for glitch trigger, 415
 range for time base, 350
 range of offset values, 173
 range of reference level values, 217
 range qualifier, 414
 ranges, value, 64
 rate, 144
 RCL (Recall), 82
 read configuration, 79
 read trace memory, 193
 readout, 565
 real-time acquisition mode, 133, 138
 recall, 82, 292, 343
 RECall commands, 292
 recall filename, 293
 recall image, 294
 recall setup, 296
 recalling and saving data, 190
 RECTangular window for transient signals, 221
 reference, 211, 352
 reference clock, 351
 REference commands, 527
 reference for time base, 579

reference level, Fast Fourier Transform (FFT)
 function, 217
 reference signal (10 MHz), 140
 reference signal mode, 351
 registers, 76, 82, 86, 98, 107, 109, 111, 114,
 116, 118, 120, 122
 registers, clearing, 612
 reject filter, 393
 reject high frequency, 362
 reject noise, 365
 remote control examples, 637
 remove cursor information, 234
 remove labels, 195
 remove message from display, 340
 reorder channels, 99
 repetitive acquisitions, 125
 report errors, 341
 report transition, 272, 274
 reporting status, 593
 reporting the setup, 358
 request service, 90
 Request-for-OPC flag clear, 73
 reset, 83, 439
 RESet commands, 527
 reset conditions, 83
 reset measurements, 104, 192
 resolution of printed copy, 560
 restore configurations, 79, 82, 86, 343
 restore labels, 195
 restore setup, 82
 return values, query, 635
 returning acquisition type, 145
 returning number of data points, 139
 right reference, 352
 rise time measurement, 248
 rise time of positive edge, 267
 rising edge, 366, 436
 RMS value measurement, 282
 roll time base mode, 348
 root (:.) commands, 94, 96
 root level commands, 68
 RQL (Request Control) status bit, 75, 77
 RQS (Request Service), 90
 RST (Reset), 83
 rules, tree traversal, 633
 rules, truncation, 618
 run, 91, 125
 Run bit, 116, 118
 running configuration, 86, 343
 Rx frame count (UART), 335
 Rx source, 467

S

sample rate, 144
 sampled data, 550
 sampled data points, 486
 SAMPlEpoint commands, 528
 SAV (Save), 86
 save, 86, 298
 SAVE commands, 297
 save filename, 299

Index

save image, [300](#)
save image with inksaver, [304](#)
save setup, [307](#)
SAVE TO INTERN, [113](#)
save waveform data, [308](#)
save waveforms to pixel memory, [113](#)
saved image, area, [301](#)
saving and recalling data, [190](#)
SBUS commands, [311](#)
scale, [218, 353, 357](#)
SCALE commands, [529](#)
scale factors output on hardcopy, [226, 302](#)
scale for channels, [180](#)
scale units for channels, [181](#)
scale units for external trigger, [209](#)
scaling display factors, [174](#)
scratch measurements, [564](#)
screen area for hardcopy print, [224](#)
screen area for saved image, [301](#)
screen data, [193](#)
SECAM, [452, 456](#)
seconds per division, [353](#)
segment, FLEXray time trigger, [405](#)
SEGmented commands, [529](#)
segments, count of waveform, [496](#)
segments, setting number of memory, [141](#)
segments, setting the index, [142](#)
segments, time tag, [497](#)
select measurement channel, [270](#)
self-test, [92](#)
sensing a channel probe, [549](#)
sensing a external trigger probe, [554](#)
sensitivity of oscilloscope input, [174](#)
sequence, [437, 438, 439](#)
sequence trigger, [441](#)
SEQUence trigger commands, [434](#)
sequence triggering, [358](#)
sequencer edge counter, [435](#)
sequencer timer, [440](#)
sequential commands, [636](#)
serial clock, [421, 448](#)
serial data, [422, 449](#)
serial decode bus, [312](#)
serial decode bus display, [323](#)
serial decode mode, [330](#)
serial frame, [450](#)
serial number, [126](#)
service request, [90](#)
Service Request Enable Register (SRE), [88, 600](#)
set, [83](#)
set center frequency, [212](#)
set conditions, [99](#)
set cursors, [569, 570](#)
set date, [339](#)
set delay, [99](#)
set thresholds, [99](#)
set time, [345](#)
set time/div, [99](#)
setting digital display, [185](#)
setting digital label, [154, 186](#)
setting digital position, [187](#)
setting digital threshold, [189](#)

setting display, [213](#)
setting external trigger level, [200](#)
setting impedance for channels, [170](#)
setting inversion for channels, [171](#)
setting pod display, [288](#)
setting pod size, [289](#)
setting pod threshold, [290](#)
settings, [82, 86](#)
settings, instrument, [222](#)
setup, [133, 148, 165, 183, 190, 211, 222, 343](#)
SETup commands, [529](#)
setup configuration, [82, 86, 343](#)
setup defaults, [83](#)
setup memory, [82](#)
setup reported, [358](#)
setup, recall, [296](#)
setup, save, [307](#)
short form, [3, 618](#)
show channel labels, [195](#)
show measurements, [248, 269](#)
SICL example in C, [638](#)
SICL examples, [638](#)
SIGNal commands, [529](#)
signal type, [177, 206](#)
simple command headers, [619](#)
single acquisition, [127](#)
single-ended signal type, [177, 206](#)
size, [188, 289](#)
SIZE commands, [530](#)
skew, [176, 547](#)
slope, [394, 443](#)
 Nth edge in burst, [389](#)
slope (direction) of waveform, [571](#)
SLOPe commands, [530](#)
slope not valid in TV trigger mode, [394](#)
slope of edge, [436](#)
slope parameter for delay measurement, [255](#)
software version, [78](#)
source, [198, 211, 270, 377, 430, 498](#)
SOURCE commands, [530](#)
source for function, [219](#)
source for trigger, [395](#)
source for TV trigger, [455](#)
span, [211](#)
span of frequency on display, [220](#)
specify measurement, [270](#)
SPI, [443, 444, 446](#)
SPI commands, [530](#)
SPI decode word width, [331](#)
SPI trigger, [445, 447](#)
SPI trigger clock, [448](#)
SPI trigger commands, [442](#)
SPI trigger data, [449](#)
SPI trigger frame, [450](#)
SPI triggering, [358](#)
SRE (Service Request Enable Register), [88, 600](#)
SRQ (Service Request interrupt), [107, 114](#)
STANdard commands, [531](#)
standard deviation measured on waveform, [268](#)
Standard Event Status Enable Register (ESE), [74, 605](#)

Standard Event Status Register (ESR), [76, 604](#)
standard for video, [456](#)
standard, LIN, [431](#)
start acquisition, [91, 105, 125, 127](#)
start and stop edges, [253](#)
STARt commands, [531](#)
start cursor, [569](#)
start measurement, [248](#)
start print job, [231](#)
start time, [415, 569](#)
start time marker, [566](#)
state memory, [86](#)
state of instrument, [79, 343](#)
status, [89, 128, 130](#)
Status Byte Register (STB), [87, 89, 90, 598](#)
STATus commands, [531](#)
status data structure clear, [73](#)
status reporting, [593](#)
STB (Status Byte Register), [87, 89, 90, 598](#)
step size for frequency span, [220](#)
stop, [105, 129](#)
stop acquisition, [129](#)
stop cursor, [570](#)
stop displaying channel, [103](#)
stop displaying math function, [103](#)
stop displaying pod, [103](#)
stop time, [415, 570](#)
storage, [86](#)
store instrument setup, [79, 86](#)
store setup, [86](#)
store waveforms to pixel memory, [113](#)
storing calibration information, [158](#)
string, quoted ASCII, [64](#)
subsource, waveform source, [502](#)
subsystem commands, [68, 633](#)
subtract math function, [211, 215, 498](#)
sweep mode, trigger, [358, 368](#)
sweep speed set to fast to measure fall time, [258](#)
sweep speed set to fast to measure rise time, [267](#)
switch disable, [342](#)
switch, calibration protect, [161](#)
sync break, LIN, [432](#)
sync frame count (FlexRay), [326](#)
syntax elements, [63](#)
syntax rules, program data, [620](#)
syntax, optional terms, [63](#)
syntax, program message, [617](#)
SYSTem commands, [338](#)
system commands, [339, 340, 341, 342, 343, 345](#)
system commands introduction, [338](#)

T

tdelta, [565](#)
tedge, [272](#)
telnet ports 5024 and 5025, [486](#)
temporary message, [340](#)
TER (Trigger Event Register), [130, 601](#)
test, self, [92](#)

- text, writing to display, 340
- threshold, 189, 290, 546, 583
- THReshold commands, 531
- threshold voltage (lower) for
 - measurement, 563
- threshold voltage (upper) for
 - measurement, 573
- thresholds, 253, 566
- thresholds used to measure period, 263
- thresholds, how autoscale affects, 99
- TIFF image format, 303
- time base, 348, 349, 350, 352, 353, 579
- time base commands introduction, 347
- time base reset conditions, 83
- time base window, 355, 356, 357
- time between points, 565
- time buckets, 135, 136
- TIME commands, 532
- time delay, 579
- time delta, 565
- time difference between data points, 506
- time duration, 381, 382, 385, 415
- time holdoff for trigger, 363
- time interval, 272, 274, 565
- time interval between trigger and
 - occurrence, 571
- time marker sets start time, 566
- time per division, 350
- time record, 221
- time slot, FLEXray time trigger, 406
- time specified, 283
- time, calibration, 163
- time, start marker, 569
- time, stop marker, 570
- time, system, 345
- time/div, how autoscale affects, 99
- time-at-max measurement, 567
- time-at-min measurement, 568
- TIMEbase commands, 346
- timebase vernier, 354
- time-ordered label list, 196
- timeout, 444
- timer, 440
- timing measurement, 248
- title channels, 172
- top of waveform value measured, 284
- TOTAL commands, 532
- total frame count (CAN), 321
- total frame count (FlexRay), 327
- trace memories, how autoscale affects, 99
- trace memory, 128, 131
- trace memory data, 193
- track measurements, 269
- trademarks, 2
- transfer instrument state, 79, 343
- transmit, 193
- tree traversal rules, 633
- tree, command, 621
- TRG (Trigger), 88, 90, 91
- trigger (external) input impedance, 203, 553
- trigger armed event register, 116, 118
- trigger burst, UART, 461
- TRIGger CAN commands, 369
- trigger channel source, 416, 455
- TRIGger commands, 358, 532
- TRIGger commands, general, 361
- trigger data, UART, 462
- trigger duration, 381, 382
- TRIGger DURation commands, 380
- TRIGger EBUrst commands, 386
- trigger edge, 436
- TRIGger EDGE commands, 390
- trigger edge coupling, 391
- trigger edge slope, 394
- trigger event bit, 130
- Trigger Event Register (TER), 601
- TRIGger FLEXray commands, 396
- TRIGger GLITCh commands, 408
- trigger holdoff, 363
- trigger idle, UART, 463
- TRIGger IIC commands, 417
- trigger level constants, 174
- trigger level voltage, 392
- TRIGger LIN commands, 426
- trigger occurred, 90
- trigger pattern, 366, 383
- trigger qualifier, 384
- trigger qualifier, UART, 466
- trigger reset conditions, 83
- TRIGger SEQUENCE commands, 434
- trigger SPI clock slope, 443
- TRIGger SPI commands, 442
- trigger status bit, 130
- trigger sweep mode, 358
- TRIGger TV commands, 451
- trigger type, UART, 469
- TRIGger UART commands, 457
- TRIGger USB commands, 471
- trigger, CAN, 378
- trigger, CAN acknowledge, 580
- trigger, CAN pattern data, 371
- trigger, CAN pattern data length, 372
- trigger, CAN pattern ID, 373
- trigger, CAN pattern ID mode, 374
- trigger, CAN sample point, 375
- trigger, CAN signal baudrate, 376
- trigger, CAN signal definition, 581
- trigger, CAN source, 377
- trigger, duration greater than, 381
- trigger, duration less than, 382
- trigger, duration pattern, 383
- trigger, duration qualifier, 384
- trigger, duration range, 385
- trigger, edge coupling, 391
- trigger, edge level, 392
- trigger, edge reject, 393
- trigger, edge slope, 394
- trigger, edge source, 395
- trigger, FLEXray, 407
- trigger, FLEXray error, 397
- trigger, glitch greater than, 410
- trigger, glitch less than, 411
- trigger, glitch level, 412
- trigger, glitch polarity, 413
- trigger, glitch qualifier, 414
- trigger, glitch range, 415
- trigger, glitch source, 416
- trigger, high frequency reject filter, 362
- trigger, holdoff, 363
- trigger, IIC clock source, 421
- trigger, IIC data source, 422
- trigger, IIC pattern address, 418
- trigger, IIC pattern data, 419
- trigger, IIC pattern data 2, 420
- trigger, IIC qualifier, 423
- trigger, IIC signal baudrate, 429
- trigger, IIC type, 424
- trigger, LIN, 433
- trigger, LIN sample point, 428
- trigger, LIN signal definition, 582
- trigger, LIN source, 430
- trigger, mode, 364
- trigger, noise reject filter, 365
- trigger, Nth edge in burst slope, 389
- trigger, Nth edge of burst count, 387
- trigger, pattern, 366
- trigger, sequence, 441
- trigger, sequence count, 435
- trigger, sequence edge, 436
- trigger, sequence find, 437
- trigger, sequence pattern, 438
- trigger, sequence reset, 439
- trigger, sequence timer, 440
- trigger, SPI clock slope, 443
- trigger, SPI clock source, 448
- trigger, SPI clock timeout, 444
- trigger, SPI data source, 449
- trigger, SPI frame source, 450
- trigger, SPI framing, 445
- trigger, SPI pattern data, 446
- trigger, SPI pattern width, 447
- trigger, sweep, 368
- trigger, threshold, 583
- trigger, TV line, 452
- trigger, TV mode, 453, 584
- trigger, TV polarity, 454
- trigger, TV source, 455
- trigger, TV standard, 456
- trigger, UART baudrate, 459
- trigger, UART bit order, 460
- trigger, UART parity, 464
- trigger, UART polarity, 465
- trigger, UART Rx source, 467
- trigger, UART Tx source, 468
- trigger, UART width, 470
- trigger, USB, 475
- trigger, USB D- source, 472
- trigger, USB D+ source, 473
- trigger, USB speed, 474
- truncation rules, 618
- TST (Self Test), 92
- tstart, 569
- tstop, 570
- TTL threshold voltage for digital channels, 189, 546
- TTL trigger threshold voltage, 583

Index

turn function on or off, 555
turn off channel, 103
turn off channel labels, 195
turn off cursors, 99
turn off delayed time base mode, 99
turn off digital pod, 103
turn off math function, 103
turn off measurements, 99
turn off trace memories, 99
turn on channel labels, 195
turn on channel number display, 551
turn on channels, 99
turning channel display on and off, 169
turning off/on function calculation, 213
turning vectors on or off, 550
TV mode, 453, 584
TV trigger commands, 451
TV trigger line number setting, 452
TV trigger mode, 455
TV trigger polarity, 454
TV trigger standard setting, 456
TV triggering, 358
tvmode, 584
Tx data, UART, 502
Tx frame count (UART), 336
Tx source, 468
type, 145, 503
TYPE commands, 536

U

UART baud rate, 459
UART bit order, 460
UART commands, 536
UART frame counters, reset, 334
UART parity, 464
UART polarity, 465
UART Rx source, 467
UART trigger burst, 461
UART trigger commands, 457
UART trigger data, 462
UART trigger idle, 463
UART trigger qualifier, 466
UART trigger type, 469
UART Tx data, 502
UART Tx source, 468
UART width, 470
UNITs commands, 536
units per division, 180, 181, 209, 353
units per division (vertical) for function, 180, 218
unsigned mode, 504
upper threshold, 263
upper threshold voltage for measurement, 573
uppercase characters in commands, 617
URQ (User Request) status bit, 75, 77
USB source, 472, 473
USB speed, 474
USB trigger, 475
USB trigger commands, 471
USB triggering, 358
user defined channel labels, 172

user defined threshold, 546
user event conditions occurred, 90
user-defined threshold voltage for digital channels, 189
user-defined trigger threshold, 583
USR (User Event bit), 88, 90
utilization, CAN bus, 322

V

valid command strings, 617
valid pattern time, 381, 382
value, 274
value measured at base of waveform, 278
value measured at specified time, 283
value measured at top of waveform, 284
value ranges, 64
values required to fill time buckets, 136
vectors, 199
vectors turned on or off, 550
vectors, turning on or off, 190
vernier, channel, 182
vernier, horizontal, 354
vertical adjustment, fine (vernier), 182
vertical amplitude measurement, 276
vertical axis defined by RANGE, 216
vertical axis range for channels, 179
vertical offset for channels, 173
vertical peak-to-peak measured on waveform, 281
vertical scale, 180, 218
vertical scaling, 493
vertical threshold, 546
vertical value at center screen, 214
vertical value maximum measured on waveform, 279
vertical value measurements to calculate overshoot, 261
vertical value minimum measured on waveform, 280
video line to trigger on, 452
video standard selection, 456
view, 131, 211, 505, 551
view turns function on or off, 555
VISA COM example in C#, 701
VISA COM example in Visual Basic .NET, 713
VISA COM example in Visual Basic 6, 692
VISA example in C, 647
VISA example in C#, 665
VISA example in Visual Basic .NET, 678
VISA example in Visual Basic 6, 655
VISA examples, 647, 692
Visual Basic .NET, VISA COM example, 713
Visual Basic .NET, VISA example, 678
Visual Basic 6, VISA COM example, 692
Visual Basic 6, VISA example, 655
voltage crossing reported or not found, 571
voltage difference between data points, 509
voltage difference measured, 574
voltage level for active trigger, 392
voltage marker used to measure waveform, 575, 576

voltage offset value for channels, 173
voltage probe, 181, 209
voltage ranges for channels, 179
voltage ranges for external trigger, 208
voltage threshold, 253

W

WAI (Wait To Continue), 93
wait, 93
wait for operation complete, 80
Wait Trig bit, 116, 118
waveform base value measured, 278
WAVEform commands, 476, 537
waveform data, 478
waveform data format, 309
waveform data length, 310
waveform data, save, 308
waveform introduction, 478
waveform maximum vertical value measured, 279
waveform minimum vertical value measured, 280
waveform must cross voltage level to be an occurrence, 571
waveform peak-to-peak vertical value measured, 281
waveform period, 263
waveform persistence, 190
waveform RMS value measured, 282
waveform source channels, 498
waveform source subsample, 502
waveform standard deviation value measured, 268
waveform vertical amplitude, 276
waveform voltage measured at marker, 575, 576
waveform, byte order, 484
waveform, count, 485
waveform, data, 486
waveform, format, 488
waveform, points, 489, 491
waveform, preamble, 493
waveform, source, 498
waveform, type, 503
waveform, unsigned, 504
waveform, view, 505
waveform, X increment, 506
waveform, X origin, 507
waveform, X reference, 508
waveform, Y increment, 509
waveform, Y origin, 510
waveform, Y reference, 511
what's new, 17
width, 415, 470
WIDTH commands, 538
window, 355, 356, 357
window time, 350
window time base mode, 348
windows, 221
windows as filters to Fast Fourier Transforms, 221

windows for Fast Fourier Transform
 functions, 221
word format, 488
word format for data transfer, 478
word width, SPI decode, 331
write text to display, 340
write trace memory, 193

X

X axis markers, 233
X delta, 239
X1 and X2 cursor value difference, 239
X1 cursor, 233, 235, 236
X2 cursor, 233, 237, 238
X-axis functions, 347
X-increment, 506
X-of-max measurement, 285
X-of-min measurement, 286
X-origin, 507
X-reference, 508
X-Y mode, 347, 348

Y

Y axis markers, 233
Y1 and Y2 cursor value difference, 242
Y1 cursor, 233, 236, 240, 242
Y2 cursor, 233, 238, 241, 242
Y-axis value, 510
Y-increment, 509
Y-origin, 510, 511
Y-reference, 511

Z

zero values in waveform data, 486

