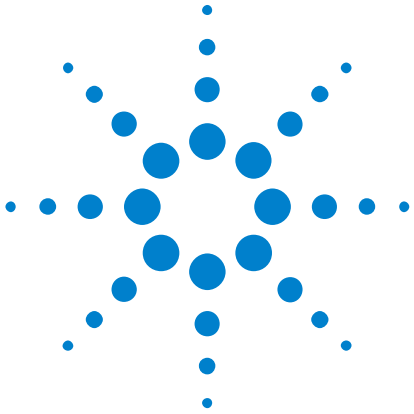


Agilent Infiniium 90000 Series Oscilloscopes

Programmer's Reference

Agilent Technologies

Notices

© Agilent Technologies, Inc. 2007-2014

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

MIPI™ is a licensed trademark of MIPI, Inc. in the U.S. and other jurisdictions.

PCI-SIG®, PCIe® and the PCI Express® are US registered trademarks and/or service marks of PCI-SIG.

Manual Part Number

Version 04.60.0013

Edition

January 23, 2014

Available in electronic format only

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

Warranty

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

Technology Licenses

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

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

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

WARNING

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

In This Book

This book is your guide to programming the Infiniium 90000 Series oscilloscopes These oscilloscopes include the 90000A Series, the 90000 X-Series, and the 90000 Q-Series.

Chapter 1, “What's New,” starting on page 29, describes programming command changes in the latest version of oscilloscope software.

Chapter 2, “Setting Up,” starting on page 45, describes the steps you must take before you can control the oscilloscope with remote programs.

The next several chapters give you an introduction to programming the oscilloscopes, along with necessary conceptual information. These chapters describe basic program communications, interface, syntax, data types, and status reporting:

- Chapter 3, “Introduction to Programming,” starting on page 53
- Chapter 4, “LAN, USB, and GPIB Interfaces,” starting on page 89
- Chapter 5, “Message Communication and System Functions,” starting on page 101
- Chapter 6, “Status Reporting,” starting on page 105
- Chapter 7, “Remote Acquisition Synchronization,” starting on page 129
- Chapter 8, “Programming Conventions,” starting on page 139

The next chapters describe the commands used to program the oscilloscopes. Each chapter describes the set of commands that belong to an individual subsystem, and explains the function of each command.

- Chapter 9, “Acquire Commands,” starting on page 149
- Chapter 10, “Bus Commands,” starting on page 177
- Chapter 11, “Calibration Commands,” starting on page 187
- Chapter 12, “Channel Commands,” starting on page 195
- Chapter 13, “Common Commands,” starting on page 251
- Chapter 14, “Digital Commands,” starting on page 275
- Chapter 15, “Disk Commands,” starting on page 281
- Chapter 16, “Display Commands,” starting on page 299
- Chapter 17, “Function Commands,” starting on page 325
- Chapter 18, “Hardcopy Commands,” starting on page 371
- Chapter 19, “Histogram Commands,” starting on page 377
- Chapter 20, “InfiniiScan (ISCan) Commands,” starting on page 389
- Chapter 21, “Limit Test Commands,” starting on page 411
- Chapter 22, “Lister Commands,” starting on page 419
- Chapter 23, “Marker Commands,” starting on page 423
- Chapter 24, “Mask Test Commands,” starting on page 435

- Chapter 25, “Measure Commands,” starting on page 479
- Chapter 26, “Pod Commands,” starting on page 741
- Chapter 27, “Root Level Commands,” starting on page 747
- Chapter 28, “Serial Bus Commands,” starting on page 783
- Chapter 29, “Self-Test Commands,” starting on page 809
- Chapter 30, “Serial Data Equalization Commands,” starting on page 813
- Chapter 31, “System Commands,” starting on page 857
- Chapter 32, “Time Base Commands,” starting on page 871
- Chapter 33, “Trigger Commands,” starting on page 883
- Chapter 34, “Waveform Commands,” starting on page 983
- Chapter 35, “Waveform Memory Commands,” starting on page 1043

Chapter 36, “Obsolete and Discontinued Commands,” starting on page 1053, describes obsolete (deprecated) commands that still work but have been replaced by newer commands, and lists discontinued commands that are no longer supported.

Chapter 37, “Error Messages,” starting on page 1059, describes error messages.

Chapter 38, “Sample Programs,” starting on page 1071, shows example programs in various languages using the VISA COM, VISA, and SICL libraries.

Finally, Chapter 39, “Reference,” starting on page 1203, contains file format descriptions.

See Also

- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Agilent IO Libraries Suite.
- For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Agilent 82350A GPIB interface).
- For information on oscilloscope front-panel operation, see the *User's Guide*.
- For detailed connectivity information, refer to the *Agilent Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to "www.agilent.com" and search for "Connectivity Guide".
- For the latest versions of this and other manuals, see: "http://www.agilent.com/find/Infiniium-manuals"

Contents

In This Book 3

1 What's New

What's New in Version 4.60 30

What's New in Version 4.50 31

What's New in Version 4.30 34

What's New in Version 4.20 35

What's New in Version 4.10 36

What's New in Version 4.00 37

What's New in Version 3.50 38

What's New in Version 3.20 40

What's New in Version 3.11 41

What's New in Version 3.10 42

2 Setting Up

Step 1. Install Agilent IO Libraries Suite software 46

Step 2. Connect and set up the oscilloscope 47

- Using the USB (Device) Interface 47
- Using the LAN Interface 47
- Using the GPIB Interface 47

Step 3. Verify the oscilloscope connection 48

3 Introduction to Programming

Communicating with the Oscilloscope 55

Instructions 56

Instruction Header 57

White Space (Separator) 58

Braces 59

Ellipsis 60

Square Brackets 61

Command and Query Sources 62
Program Data 63
Header Types 64
Simple Command Header 64
Compound Command Header 64
Combining Commands in the Same Subsystem 65
Common Command Header 65
Duplicate Mnemonics 65
Query Headers 66
Program Header Options 67
Character Program Data 68
Numeric Program Data 69
Embedded Strings 70
Program Message Terminator 71
Common Commands within a Subsystem 72
Selecting Multiple Subsystems 73
Programming Getting Started 74
Referencing the IO Library 75
Opening the Oscilloscope Connection via the IO Library 76
Initializing the Interface and the Oscilloscope 77
Autoscale 77
Setting Up the Oscilloscope 78
Example Program 79
Using the DIGITIZE Command 80
Receiving Information from the Oscilloscope 82
String Variable Example 83
Numeric Variable Example 84
Definite-Length Block Response Data 85
Multiple Queries 86
Oscilloscope Status 87

4 LAN, USB, and GPIB Interfaces

LAN Interface Connector 90
GPIB Interface Connector 91

Default Startup Conditions 92
Interface Capabilities 93
GPIB Command and Data Concepts 94
Communicating Over the GPIB Interface 95
Interface Select Code 95
Oscilloscope Address 95
Communicating Over the LAN Interface 96
Communicating via Telnet and Sockets 98
Telnet 98
Sockets 98
Bus Commands 100
Device Clear 100
Group Execute Trigger 100
Interface Clear 100

5 Message Communication and System Functions

Protocols 102
Functional Elements 102
Protocol Overview 102
Protocol Operation 103
Protocol Exceptions 103
Suffix Multiplier 103
Suffix Unit 104

6 Status Reporting

Status Reporting Data Structures 108
Status Byte Register 110
Service Request Enable Register 112
Message Event Register 113
Trigger Event Register 114
Standard Event Status Register 115
Standard Event Status Enable Register 116
Operation Status Register 117
Operation Status Enable Register 118
Mask Test Event Register 119
Mask Test Event Enable Register 120

Acquisition Done Event Register 121
Process Done Event Register 122
Trigger Armed Event Register 123
Auto Trigger Event Register 124
Error Queue 125
Output Queue 126
Message Queue 127
Clearing Registers and Queues 128

7 Remote Acquisition Synchronization

Programming Flow 130
Setting Up the Oscilloscope 131
Acquiring a Waveform 132
Retrieving Results 133
Acquisition Synchronization 134
Blocking Synchronization 134
Polling Synchronization With Timeout 134
Single Shot Device Under Test (DUT) 136
Averaging Acquisition Synchronization 138

8 Programming Conventions

Truncation Rule 140
The Command Tree 141
Command Types 141
Tree Traversal Rules 141
Tree Traversal Examples 142
Infinity Representation 144
Sequential and Overlapped Commands 145
Response Generation 146
EOI 147

9 Acquire Commands

:ACQuire:AVERage 150
:ACQuire[:AVERage]:COUNt 151
:ACQuire:BANDwidth 152

:ACQuire:COMPlete 154
:ACQuire:COMPlete:STATe 156
:ACQuire:HRESolution 157
:ACQuire:INTerpolate 158
:ACQuire:MODE 159
:ACQuire:POINts[:ANALog] 161
:ACQuire:POINts:AUTO 165
:ACQuire:POINts:DIGital? 166
:ACQuire:REDGe 167
:ACQuire:RESPonse 168
:ACQuire:SEGMented:COUNt 169
:ACQuire:SEGMented:INDex 170
:ACQuire:SEGMented:TTAGs 171
:ACQuire:SRATe[:ANALog] 172
:ACQuire:SRATe[:ANALog]:AUTO 174
:ACQuire:SRATe:DIGital 175
:ACQuire:SRATe:DIGital:AUTO 176

10 Bus Commands

:BUS:B<N>:TYPE 178
:BUS<N>:BIT<M> 179
:BUS<N>:BITS 180
:BUS<N>:CLEar 181
:BUS<N>:CLOCk 182
:BUS<N>:CLOCk:SLOPe 183
:BUS<N>:DISPlay 184
:BUS<N>:LABel 185
:BUS<N>:READout 186

11 Calibration Commands

:CALibrate:DATE? 189
:CALibrate:OUTPut 190
:CALibrate:SKEW 191
:CALibrate:STATus? 192
:CALibrate:TEMP? 193

12 Channel Commands

:CHANnel<N>:COMMonmode 197
:CHANnel<N>:DIFFerential 198
:CHANnel<N>:DIFFerential:SKEW 199
:CHANnel<N>:DISPlay 200

:CHANnel<N>:DISPlay:AUTO 201
:CHANnel<N>:DISPlay:OFFSet 202
:CHANnel<N>:DISPlay:RANGe 203
:CHANnel<N>:DISPlay:SCALe 204
:CHANnel<N>:INPut 205
:CHANnel<N>:ISIM:APPLy 206
:CHANnel<N>:ISIM:BANDwidth 207
:CHANnel<N>:ISIM:BWLimit 208
:CHANnel<N>:ISIM:CONVolve 209
:CHANnel<N>:ISIM:CORRection 210
:CHANnel<N>:ISIM:DECONvolve 212
:CHANnel<N>:ISIM:DELay 213
:CHANnel<N>:ISIM:NORMalize 215
:CHANnel<N>:ISIM:PEXTraction 216
:CHANnel<N>:ISIM:SPAN 218
:CHANnel<N>:ISIM:STATe 219
:CHANnel<N>:LABel 220
:CHANnel<N>:OFFSet 221
:CHANnel<N>:PROBe 222
:CHANnel<N>:PROBe:ACCAL 223
:CHANnel<N>:PROBe:ATTenuation 224
:CHANnel<N>:PROBe:AUTozero 225
:CHANnel<N>:PROBe:COUPling 226
:CHANnel<N>:PROBe:EADapter 227
:CHANnel<N>:PROBe:ECOupling 229
:CHANnel<N>:PROBe:EXTernal 230
:CHANnel<N>:PROBe:EXTernal:GAIN 231
:CHANnel<N>:PROBe:EXTernal:OFFSet 232
:CHANnel<N>:PROBe:EXTernal:UNITs 233
:CHANnel<N>:PROBe:GAIN 234
:CHANnel<N>:PROBe:HEAD:ADD 235
:CHANnel<N>:PROBe:HEAD:DELete ALL 236
:CHANnel<N>:PROBe:HEAD:SELect 237
:CHANnel<N>:PROBe:HEAD:VTERm 238
:CHANnel<N>:PROBe:ID? 239
:CHANnel<N>:PROBe:MODE 240
:CHANnel<N>:PROBe:PRECprobe:BANDwidth 241
:CHANnel<N>:PROBe:PRECprobe:CALibration 242
:CHANnel<N>:PROBe:PRECprobe:MODE 243
:CHANnel<N>:PROBe:PRECprobe:ZSRC 244
:CHANnel<N>:PROBe:SKEW 246
:CHANnel<N>:PROBe:STYPe 247

:CHANnel<N>:RANGe 248
:CHANnel<N>:SCALe 249
:CHANnel<N>:UNITs 250

13 Common Commands

*CLS 253
*ESE 254
*ESR? 256
*IDN? 257
*LRN? 258
*OPC 260
*OPT? 261
*PSC 264
*RCL 265
*RST 266
*SAV 267
*SRE 268
*STB? 270
*TRG 272
*TST? 273
*WAI 274

14 Digital Commands

:DIGital<N>:DISPlay 276
:DIGital<N>:LABel 277
:DIGital<N>:SIZE 278
:DIGital<N>:THReshold 279

15 Disk Commands

:DISK:CDIRectory 282
:DISK:COPY 283
:DISK:DELete 284
:DISK:DIRectory? 285
:DISK:LOAD 286
:DISK:MDIRectory 287
:DISK:PWD? 288
:DISK:SAVE:COMPosite 289
:DISK:SAVE:IMAGe 290
:DISK:SAVE:JITTer 291
:DISK:SAVE:LISTing 292
:DISK:SAVE:MEASurements 293

:DISK:SAVE:PRECprobe 294
:DISK:SAVE:SETup 295
:DISK:SAVE:WAVeform 296
:DISK:SEGMented 298

16 Display Commands

:DISPlay:CGRade 300
:DISPlay:CGRade:LEVels? 301
:DISPlay:CGRade:SCHeme 303
:DISPlay:COLumn 305
:DISPlay:CONNect 306
:DISPlay:DATA? 307
:DISPlay:GRATicule 308
:DISPlay:GRATicule:INTensity 309
:DISPlay:GRATicule:NUMBer 310
:DISPlay:GRATicule:SETGrat 311
:DISPlay:GRATicule:SIZE 312
:DISPlay:LABel 313
:DISPlay:LINE 314
:DISPlay:PERSistence 315
:DISPlay:ROW 316
:DISPlay:SCOLor 317
:DISPlay:STATus:COL 319
:DISPlay:STATus:ROW 320
:DISPlay:STRing 321
:DISPlay:TAB 322
:DISPlay:TEXT 323

17 Function Commands

:FUNCtion<F>? 328
:FUNCtion<F>:ABSolute 329
:FUNCtion<F>:ADD 330
:FUNCtion<F>:ADEMod 331
:FUNCtion<F>:AVERage 332
:FUNCtion<F>:COMMonmode 333
:FUNCtion<F>:DELay 334
:FUNCtion<F>:DIFF 335
:FUNCtion<F>:DISPlay 336
:FUNCtion<F>:DIVide 337
:FUNCtion<F>:FFT:FREQuency 338
:FUNCtion<F>:FFT:REFerence 339

:FUNCtion<F>:FFT:RESolution? 340
:FUNCtion<F>:FFT:TDELay 341
:FUNCtion<F>:FFT:WINDow 342
:FUNCtion<F>:FFTMagnitude 344
:FUNCtion<F>:FFTPhase 345
:FUNCtion<F>:GATing 346
:FUNCtion<F>:HIGHpass 347
:FUNCtion<F>:HORizontal 348
:FUNCtion<F>:HORizontal:POSition 349
:FUNCtion<F>:HORizontal:RANGe 350
:FUNCtion<F>:INTegrate 351
:FUNCtion<F>:INVert 352
:FUNCtion<F>:LOWPass 353
:FUNCtion<F>:MAGNify 354
:FUNCtion<F>:MAXimum 355
:FUNCtion<F>:MHIStogram 356
:FUNCtion<F>:MINimum 357
:FUNCtion<F>:MTRend 358
:FUNCtion<F>:MULTiply 359
:FUNCtion<F>:OFFSet 360
:FUNCtion<F>:RANGe 361
:FUNCtion<F>:SMOoth 362
:FUNCtion<F>:SQRT 363
:FUNCtion<F>:SQUare 364
:FUNCtion<F>:SUBTract 365
:FUNCtion<F>:VERSus 366
:FUNCtion<F>:VERTical 367
:FUNCtion<F>:VERTical:OFFSet 368
:FUNCtion<F>:VERTical:RANGe 369

18 Hardcopy Commands

:HARDcopy:AREA 372
:HARDcopy:DPRinter 373
:HARDcopy:FACTors 374
:HARDcopy:IMAGe 375
:HARDcopy:PRINters? 376

19 Histogram Commands

:HISTogram:AXIS 379
:HISTogram:MODE 380
:HISTogram:SCALe:SIZE 381

:HISTogram:WINDow:DEFault 382
:HISTogram:WINDow:SOURce 383
:HISTogram:WINDow:LLIMit 384
:HISTogram:WINDow:RLIMit 385
:HISTogram:WINDow:BLIMit 386
:HISTogram:WINDow:TLIMit 387

20 InfiniiScan (ISCan) Commands

:ISCan:DELay 390
:ISCan:MEASurement:FAIL 391
:ISCan:MEASurement:LLIMit 392
:ISCan:MEASurement 393
:ISCan:MEASurement:ULIMit 394
:ISCan:MODE 395
:ISCan:NONMonotonic:EDGE 396
:ISCan:NONMonotonic:HYSTeresis 397
:ISCan:NONMonotonic:SOURce 398
:ISCan:RUNT:HYSTeresis 399
:ISCan:RUNT:LLEVel 400
:ISCan:RUNT:SOURce 401
:ISCan:RUNT:ULEVel 402
:ISCan:SERial:PATTern 403
:ISCan:SERial:SOURce 404
:ISCan:ZONE:HIDE 405
:ISCan:ZONE:SOURce 406
:ISCan:ZONE<N>:MODE 407
:ISCan:ZONE<N>:PLACement 408
:ISCan:ZONE<N>:STATe 409

21 Limit Test Commands

:LTESt:FAIL 412
:LTESt:LLIMit 413
:LTESt:MEASurement 414
:LTESt:RESults? 415
:LTESt:TEST 416
:LTESt:ULIMit 417

22 Lister Commands

:LISTer:DATA 420
:LISTer:DISPlay 421

23 Marker Commands

:MARKer:CURSor? 424
:MARKer:MEASurement:MEASurement 425
:MARKer:MODE 426
:MARKer:X1Position 427
:MARKer:X2Position 428
:MARKer:X1Y1source 429
:MARKer:X2Y2source 430
:MARKer:XDELta? 431
:MARKer:Y1Position 432
:MARKer:Y2Position 433
:MARKer:YDELta? 434

24 Mask Test Commands

:MTESt:ALIGn 437
:MTESt:AlignFIT 438
:MTESt:AMASk:CREate 440
:MTESt:AMASk:SOURce 441
:MTESt:AMASk:SAVE 442
:MTESt:AMASk:UNITs 443
:MTESt:AMASk:XDELta 444
:MTESt:AMASk:YDELta 445
:MTESt:AUTO 446
:MTESt:AVERage 447
:MTESt:AVERage:COUNt 448
:MTESt:COUNt:FAILures? 449
:MTESt:COUNt:FUI? 450
:MTESt:COUNt:FWAVeforms? 451
:MTESt:COUNt:UI? 452
:MTESt:COUNt:WAVeforms? 453
:MTESt:DELete 454
:MTESt:ENABle 455
:MTESt:FOLDing 456
:MTESt:FOLDing:BITS 457
:MTESt:FOLDing:FAST 458
:MTESt:HAMPlitude 459
:MTESt:IMPedance 460
:MTESt:INVert 461
:MTESt:LAMPlitude 462
:MTESt:LOAD 463
:MTESt:NREGions? 464

:MTESt:PROBe:IMPedance? 465
:MTESt:RUMode 466
:MTESt:RUMode:SOFailure 467
:MTESt:SCALe:BIND 468
:MTESt:SCALe:X1 469
:MTESt:SCALe:XDELta 470
:MTESt:SCALe:Y1 471
:MTESt:SCALe:Y2 472
:MTESt:SOURce 473
:MTESt:STARt 474
:MTESt:STOP 475
:MTESt:STIMe 476
:MTESt:TITLe? 477
:MTESt:TRIGger:SOURce 478

25 Measure Commands

:MEASure:AREA 488
:MEASure:BINTerval 489
:MEASure:BPERiod 490
:MEASure:BWIDth 491
:MEASure:CDRRATE 492
:MEASure:CGRade:CROSsing 493
:MEASure:CGRade:DCDistortion 494
:MEASure:CGRade:EHEight 495
:MEASure:CGRade:EWIDth 496
:MEASure:CGRade:EWINdow 497
:MEASure:CGRade:JITTer 498
:MEASure:CGRade:QFACtor 499
:MEASure:CLEar 500
:MEASure:CLOCk 501
:MEASure:CLOCk:METHod 502
:MEASure:CLOCk:METHod:ALIGn 504
:MEASure:CLOCk:METHod:DEEMphasis 505
:MEASure:CLOCk:METHod:EDGE 506
:MEASure:CLOCk:METHod:JTF 508
:MEASure:CLOCk:METHod:OJTF 510
:MEASure:CLOCk:VERTical 512
:MEASure:CLOCk:VERTical:OFFSet 513
:MEASure:CLOCk:VERTical:RANGe 514
:MEASure:CROSsing 515
:MEASure:CTCDutycycle 516

:MEASure:CTCJitter 518
:MEASure:CTCNwidth 520
:MEASure:CTCPwidth 522
:MEASure:DATarate 524
:MEASure:DEEMphasis 526
:MEASure:DELTatime 528
:MEASure:DELTatime:DEFine 530
:MEASure:DUTYcycle 532
:MEASure:EDGE 534
:MEASure:ETOedge 536
:MEASure:FALLtime 537
:MEASure:FFT:DFRequency 539
:MEASure:FFT:DMAGnitude 541
:MEASure:FFT:FREQuency 543
:MEASure:FFT:MAGNitude 544
:MEASure:FFT:PEAK1 545
:MEASure:FFT:PEAK2 546
:MEASure:FFT:THReshold 547
:MEASure:FREQuency 548
:MEASure:HISTogram:HITS 550
:MEASure:HISTogram:M1S 551
:MEASure:HISTogram:M2S 552
:MEASure:HISTogram:M3S 553
:MEASure:HISTogram:MAX 554
:MEASure:HISTogram:MEAN 555
:MEASure:HISTogram:MEDian 556
:MEASure:HISTogram:MIN 557
:MEASure:HISTogram:MODE 558
:MEASure:HISTogram:PEAK 559
:MEASure:HISTogram:PP 560
:MEASure:HISTogram:RESolution 561
:MEASure:HISTogram:STDDev 562
:MEASure:HOLDtime 563
:MEASure:JITTer:HISTogram 565
:MEASure:JITTer:MEASurement 566
:MEASure:JITTer:SPECtrum 567
:MEASure:JITTer:SPECtrum:HORizontal 568
:MEASure:JITTer:SPECtrum:HORizontal:POSition 569
:MEASure:JITTer:SPECtrum:HORizontal:RANGe 570
:MEASure:JITTer:SPECtrum:VERTical 571
:MEASure:JITTer:SPECtrum:VERTical:OFFSet 572
:MEASure:JITTer:SPECtrum:VERTical:RANGe 573

:MEASure:JITTer:SPECtrum:VERTical:TYPE 574
:MEASure:JITTer:SPECtrum:WINDow 575
:MEASure:JITTer:STATistics 576
:MEASure:JITTer:TRENd 577
:MEASure:JITTer:TRENd:SMOoth 578
:MEASure:JITTer:TRENd:SMOoth:POINts 579
:MEASure:JITTer:TRENd:VERTical 580
:MEASure:JITTer:TRENd:VERTical:OFFSet 581
:MEASure:JITTer:TRENd:VERTical:RANGe 582
:MEASure:NAME 583
:MEASure:NCJitter 584
:MEASure:NOISe 586
:MEASure:NOISe:ALL? 587
:MEASure:NOISe:BANDwidth 589
:MEASure:NOISe:LOCation 590
:MEASure:NOISe:METHod 591
:MEASure:NOISe:REPort 592
:MEASure:NOISe:RN 593
:MEASure:NOISe:SCOPe:RN 594
:MEASure:NOISe:STATe 595
:MEASure:NOISe:UNITs 596
:MEASure:NPERiod 597
:MEASure:NPULses 598
:MEASure:NUI 599
:MEASure:NWIDth 600
:MEASure:OVERshoot 602
:MEASure:PAMPlitude 604
:MEASure:PBASe 605
:MEASure:PERiod 606
:MEASure:PHASe 608
:MEASure:PPULses 610
:MEASure:PREShoot 611
:MEASure:PTOP 613
:MEASure:PWIDth 614
:MEASure:QUALifier<M>:CONDition 616
:MEASure:QUALifier<M>:SOURce 617
:MEASure:QUALifier<M>:STATe 618
:MEASure:RESults? 619
:MEASure:RISetime 622
:MEASure:RJDJ:ALL? 624
:MEASure:RJDJ:APLength? 626
:MEASure:RJDJ:BANDwidth 627

:MEASure:RJDJ:BER 628
:MEASure:RJDJ:CLOCk 630
:MEASure:RJDJ:EDGE 631
:MEASure:RJDJ:INTerpolate 632
:MEASure:RJDJ:METHod 633
:MEASure:RJDJ:MODe 634
:MEASure:RJDJ:PLENgth 635
:MEASure:RJDJ:REPort 636
:MEASure:RJDJ:RJ 637
:MEASure:RJDJ:SCOPe:RJ 638
:MEASure:RJDJ:SOURce 639
:MEASure:RJDJ:STATe 640
:MEASure:RJDJ:TJRJDJ? 641
:MEASure:RJDJ:UNITs 642
:MEASure:SCRatch 643
:MEASure:SENDvalid 644
:MEASure:SETuptime 645
:MEASure:SLEWrate 647
:MEASure:SOURce 648
:MEASure:STATistics 649
:MEASure:TEDGe 650
:MEASure:THResholds:ABSolute 652
:MEASure:THResholds:GENeral:ABSolute 654
:MEASure:THResholds:GENeral:HYSTeresis 656
:MEASure:THResholds:GENeral:METHod 658
:MEASure:THResholds:GENeral:PERCent 660
:MEASure:THResholds:GENeral:TOPBase:ABSolute 662
:MEASure:THResholds:GENeral:TOPBase:METHod 664
:MEASure:THResholds:HYSTeresis 666
:MEASure:THResholds:METHod 668
:MEASure:THResholds:PERCent 670
:MEASure:THResholds:RFALl:ABSolute 672
:MEASure:THResholds:RFALl:HYSTeresis 674
:MEASure:THResholds:RFALl:METHod 676
:MEASure:THResholds:RFALl:PERCent 678
:MEASure:THResholds:RFALl:TOPBase:ABSolute 680
:MEASure:THResholds:RFALl:TOPBase:METHod 682
:MEASure:THResholds:SERial:ABSolute 684
:MEASure:THResholds:SERial:HYSTeresis 686
:MEASure:THResholds:SERial:METHod 688
:MEASure:THResholds:SERial:PERCent 690
:MEASure:THResholds:SERial:TOPBase:ABSolute 692

:MEASure:THResholds:SERial:TOPBase:METHod 694
:MEASure:THResholds:TOPBase:ABSolute 696
:MEASure:THResholds:TOPBase:METHod 698
:MEASure:TIEClock2 700
:MEASure:TIEData 702
:MEASure:TIEFilter:SHAPe 704
:MEASure:TIEFilter:STARt 705
:MEASure:TIEFilter:STATe 706
:MEASure:TIEFilter:STOP 707
:MEASure:TIEFilter:TYPE 708
:MEASure:TMAX 709
:MEASure:TMIN 710
:MEASure:TVOLt 711
:MEASure:UITouijitter 713
:MEASure:UNITinterval 714
:MEASure:VAMPlitude 716
:MEASure:VAVerage 717
:MEASure:VBASe 719
:MEASure:VLOWer 720
:MEASure:VMAX 721
:MEASure:VMIDdle 723
:MEASure:VMIN 724
:MEASure:VOVershoot 726
:MEASure:VPP 727
:MEASure:VPReshoot 729
:MEASure:VRMS 730
:MEASure:VTIMe 732
:MEASure:VTOP 734
:MEASure:VUPPer 735
:MEASure:WINDow 737
:MEASurement<N>:NAME 738
:MEASurement<N>:SOURce 739

26 Pod Commands

:POD<N>:DISPlay 742
:POD<N>:PSKew 743
:POD<N>:THReshold 744

27 Root Level Commands

:ADER? 749
:AER? 750

:ASTate? 751
:ATER? 752
:AUToscale 753
:AUToscale:CHANnels 754
:AUToscale:PLACement 755
:AUToscale:VERTical 756
:BEEP 757
:BLANk 758
:CDISplay 759
:DIGitize 760
:DISable DIGital 761
:ENABle DIGital 762
:MODel? 763
:MTEE 764
:MTER? 765
:OPEE 766
:OPER? 767
:OVLRegister? 768
:PDER? 769
:PRINt 770
:RECall:SETup 771
:RSTate? 772
:RUN 773
:SERial 774
:SINGle 775
:STATus? 776
:STOP 777
:STORe:JITTer 778
:STORe:SETup 779
:STORe:WAVeform 780
:TER? 781
:VIEW 782

28 Serial Bus Commands

General :SBUS<N> Commands 784
:SBUS<N>[:DISPlay] 785
:SBUS<N>:MODE 786

:SBUS<N>:CAN Commands 787
:SBUS<N>:CAN:SAMPlepoint 788
:SBUS<N>:CAN:SIGNal:BAUDrate 789
:SBUS<N>:CAN:SIGNal:DEFinition 790

:SBUS<N>:CAN:SOURce 791

:SBUS<N>:IIC Commands 792
:SBUS<n>:IIC:ASIZe 793
:SBUS<N>:IIC:SOURce:CLOCk 794
:SBUS<N>:IIC:SOURce:DATA 795

:SBUS<N>:SPI Commands 796
:SBUS<N>:SPI:BITorder 797
:SBUS<N>:SPI:CLOCk:SLOPe 798
:SBUS<N>:SPI:CLOCk:TIMeout 799
:SBUS<N>:SPI:FRAMe:STATe 800
:SBUS<N>:SPI:SOURce:CLOCk 801
:SBUS<N>:SPI:SOURce:DATA 802
:SBUS<N>:SPI:SOURce:FRAMe 803
:SBUS<N>:SPI:SOURce:MISO 804
:SBUS<N>:SPI:SOURce:MOSI 805
:SBUS<N>:SPI:TYPE 806
:SBUS<N>:SPI:WIDTh 807

29 Self-Test Commands

:SELFtest:CANCel 810
:SELFtest:SCOPETEST 811

30 Serial Data Equalization Commands

:SPRocessing:CTLequalizer:DISPlay 815
:SPRocessing:CTLequalizer:SOURce 816
:SPRocessing:CTLequalizer:DCGain 817
:SPRocessing:CTLequalizer:NUMPoles 818
:SPRocessing:CTLequalizer:P1 819
:SPRocessing:CTLequalizer:P2 820
:SPRocessing:CTLequalizer:P3 821
:SPRocessing:CTLequalizer:RATe 822
:SPRocessing:CTLequalizer:VERTical 823
:SPRocessing:CTLequalizer:VERTical:OFFSet 824
:SPRocessing:CTLequalizer:VERTical:RANGe 825
:SPRocessing:CTLequalizer:ZERo 826
:SPRocessing:DFEQualizer:STATe 827
:SPRocessing:DFEQualizer:SOURce 828
:SPRocessing:DFEQualizer:NTAPs 829
:SPRocessing:DFEQualizer:TAP 830
:SPRocessing:DFEQualizer:TAP:WIDTh 831
:SPRocessing:DFEQualizer:TAP:DELay 832

:SPRocessing:DFEQualizer:TAP:MAX 833
:SPRocessing:DFEQualizer:TAP:MIN 834
:SPRocessing:DFEQualizer:TAP:GAIN 835
:SPRocessing:DFEQualizer:TAP:UTARget 836
:SPRocessing:DFEQualizer:TAP:LTARget 837
:SPRocessing:DFEQualizer:TAP:AUTomatic 838
:SPRocessing:FFEQualizer:DISPlay 839
:SPRocessing:FFEQualizer:SOURce 840
:SPRocessing:FFEQualizer:NPRecursor 841
:SPRocessing:FFEQualizer:NTAPs 842
:SPRocessing:FFEequalizer:RATe 843
:SPRocessing:FFEQualizer:TAP 844
:SPRocessing:FFEQualizer:TAP:PLENgth 845
:SPRocessing:FFEQualizer:TAP:WIDTh 846
:SPRocessing:FFEQualizer:TAP:DELay 847
:SPRocessing:FFEQualizer:TAP:AUTomatic 848
:SPRocessing:FFEQualizer:TAP:BANDwidth 849
:SPRocessing:FFEQualizer:TAP:BWMode 850
:SPRocessing:FFEQualizer:TAP:TDELay 851
:SPRocessing:FFEQualizer:TAP:TDMode 852
:SPRocessing:FFEQualizer:VERTical 853
:SPRocessing:FFEQualizer:VERTical:OFFSet 854
:SPRocessing:FFEQualizer:VERTical:RANGe 855

31 System Commands

:SYSTem:DATE 858
:SYSTem:DEBug 859
:SYSTem:DSP 861
:SYSTem:ERRor? 862
:SYSTem:HEADer 863
:SYSTem:LOCK 864
:SYSTem:LONGform 865
:SYSTem:PRESet 866
:SYSTem:SETup 867
:SYSTem:TIME 869

32 Time Base Commands

:TIMebase:POSition 872
:TIMebase:RANGe 873
:TIMebase:REFClock 874
:TIMebase:REFerence 875

:TIMebase:SCALe 876
:TIMebase:VIEW 877
:TIMebase:WINDow:DELay 878
:TIMebase:WINDow:POSition 879
:TIMebase:WINDow:RANGe 880
:TIMebase:WINDow:SCALe 881

33 Trigger Commands

General Trigger Commands 885
:TRIGger:AND:ENABle 886
:TRIGger:AND:SOURce 887
:TRIGger:HOLDoff 888
:TRIGger:HOLDoff:MAX 889
:TRIGger:HOLDoff:MIN 890
:TRIGger:HOLDoff:MODE 891
:TRIGger:HTHReshold 892
:TRIGger:HYSTeresis 893
:TRIGger:LEVel 894
:TRIGger:LTHReshold 895
:TRIGger:MODE 896
:TRIGger:SWEep 898

Comm Trigger Commands 899
:TRIGger:COMM:BWIDth 900
:TRIGger:COMM:ENCode 901
:TRIGger:COMM:PATTern 902
:TRIGger:COMM:POLarity 903
:TRIGger:COMM:SOURce 904

Delay Trigger Commands 905
:TRIGger:DELay:ARM:SOURce 906
:TRIGger:DELay:ARM:SLOPe 907
:TRIGger:DELay:EDELay:COUNt 908
:TRIGger:DELay:EDELay:SOURce 909
:TRIGger:DELay:EDELay:SLOPe 910
:TRIGger:DELay:MODE 911
:TRIGger:DELay:TDELay:TIME 912
:TRIGger:DELay:TRIGger:SOURce 913
:TRIGger:DELay:TRIGger:SLOPe 914

Edge Trigger Commands 915
:TRIGger:EDGE:SLOPe 916
:TRIGger:EDGE:SOURce 917

Glitch Trigger Commands 918
:TRIGger:GLITch:POLarity 919
:TRIGger:GLITch:SOURce 920
:TRIGger:GLITch:WIDTh 921
Pattern Trigger Commands 922
:TRIGger:PATTern:CONDition 923
:TRIGger:PATTern:LOGic 924
Pulse Width Trigger Commands 925
:TRIGger:PWIDth:DIRection 926
:TRIGger:PWIDth:POLarity 927
:TRIGger:PWIDth:SOURce 928
:TRIGger:PWIDth:TPOint 929
:TRIGger:PWIDth:WIDTh 930
Runt Trigger Commands 931
:TRIGger:RUNT:POLarity 932
:TRIGger:RUNT:QUALified 933
:TRIGger:RUNT:SOURce 934
:TRIGger:RUNT:TIME 935
Sequence Trigger Commands 936
:TRIGger:SEQuence:TERM1 937
:TRIGger:SEQuence:TERM2 938
:TRIGger:SEQuence:RESet:ENABle 939
:TRIGger:SEQuence:RESet:TYPE 940
:TRIGger:SEQuence:RESet:EVENt 941
:TRIGger:SEQuence:RESet:TIME 942
:TRIGger:SEQuence:WAIT:ENABle 943
:TRIGger:SEQuence:WAIT:TIME 944
Setup and Hold Trigger Commands 945
:TRIGger:SHOLd:CSOurce 946
:TRIGger:SHOLd:CSOurce:EDGE 947
:TRIGger:SHOLd:DSOurce 948
:TRIGger:SHOLd:HoldTIMe (HTIMe) 949
:TRIGger:SHOLd:MODE 950
:TRIGger:SHOLd:SetupTIMe 951
State Trigger Commands 952
:TRIGger:STATe:CLOCk 953
:TRIGger:STATe:LOGic 954
:TRIGger:STATe:LTYPe 955
:TRIGger:STATe:SLOPe 956

Timeout Trigger Commands 957
:TRIGger:TIMeout:CONDition 958
:TRIGger:TIMeout:SOURce 959
:TRIGger:TIMeout:TIME 960
Transition Trigger Commands 961
:TRIGger:TRANsition:DIRection 962
:TRIGger:TRANsition:SOURce 963
:TRIGger:TRANsition:TIME 964
:TRIGger:TRANsition:TYPE 965
TV Trigger Commands 966
:TRIGger:TV:LINE 967
:TRIGger:TV:MODE 968
:TRIGger:TV:POLarity 969
:TRIGger:TV:SOURce 970
:TRIGger:TV:STANdard 971
:TRIGger:TV:UDTV:ENUMber 972
:TRIGger:TV:UDTV:HSYNc 973
:TRIGger:TV:UDTV:HTIMe 974
:TRIGger:TV:UDTV:PGTHan 975
:TRIGger:TV:UDTV:POLarity 976
Window Trigger Commands 977
:TRIGger:WINDow:CONDition 978
:TRIGger:WINDow:SOURce 979
:TRIGger:WINDow:TIME 980
:TRIGger:WINDow:TPOint 981

34 Waveform Commands

:WAVeform:BANDpass? 986
:WAVeform:BYTeorder 987
:WAVeform:COMPlete? 988
:WAVeform:COUNt? 989
:WAVeform:COUPling? 990
:WAVeform:DATA? 991
:WAVeform:FORMat 1013
:WAVeform:POINts? 1016
:WAVeform:PREamble? 1017
:WAVeform:SEGMented:ALL 1022
:WAVeform:SEGMented:COUNt? 1023
:WAVeform:SEGMented:TTAG? 1024
:WAVeform:SEGMented:XLISt? 1025

:WAVeform:SOURce 1026
:WAVeform:STReaming 1027
:WAVeform:TYPE? 1028
:WAVeform:VIEW 1029
:WAVeform:XDISplay? 1031
:WAVeform:XINCrement? 1032
:WAVeform:XORigin? 1033
:WAVeform:XRANge? 1034
:WAVeform:XREFerence? 1035
:WAVeform:XUNits? 1036
:WAVeform:YDISplay? 1037
:WAVeform:YINCrement? 1038
:WAVeform:YORigin? 1039
:WAVeform:YRANge? 1040
:WAVeform:YREFerence? 1041
:WAVeform:YUNits? 1042

35 Waveform Memory Commands

:WMEMory<N>:CLEar 1044
:WMEMory<N>:DISPlay 1045
:WMEMory<N>:LOAD 1046
:WMEMory<N>:SAVE 1047
:WMEMory<N>:XOFFset 1048
:WMEMory<N>:XRANge 1049
:WMEMory<N>:YOFFset 1050
:WMEMory<N>:YRANge 1051

36 Obsolete and Discontinued Commands

:MEASure:CLOCk:METHod 1054
:MEASure:DDPWS 1057

37 Error Messages

Error Queue 1060
Error Numbers 1061
Command Error 1062
Execution Error 1063
Device- or Oscilloscope-Specific Error 1064
Query Error 1065
List of Error Messages 1066

38 Sample Programs

VISA COM Examples 1072
- VISA COM Example in Visual Basic 1072
- VISA COM Example in C# 1083
- VISA COM Example in Visual Basic .NET 1093
- VISA COM Example in Python 1103

VISA Examples 1111
- VISA Example in C 1111
- VISA Example in Visual Basic 1120
- VISA Example in C# 1130
- VISA Example in Visual Basic .NET 1142
- VISA Example in Python 1154

SICL Examples 1162
- SICL Example in C 1162
- SICL Example in Visual Basic 1171

SCPI.NET Examples 1181
- SCPI.NET Example in C# 1181
- SCPI.NET Example in Visual Basic .NET 1189
- SCPI.NET Example in IronPython 1196

39 Reference

HDF5 Example 1204

CSV and TSV Header Format 1205

BIN Header Format 1207
- File Header 1207
- Waveform Header 1207
- Waveform Data Header 1209
- Example Program for Reading Binary Data 1210

Index

1 What's New

What's New in Version 4.60 30
What's New in Version 4.50 31
What's New in Version 4.30 34
What's New in Version 4.20 35
What's New in Version 4.10 36
What's New in Version 4.00 37
What's New in Version 3.50 38
What's New in Version 3.20 40
What's New in Version 3.11 41
What's New in Version 3.10 42

What's New in Version 4.60

New command descriptions for Version 4.60 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:ASTate? (see page 751)	Returns the acquisition state.
:CHANnel<N>:ISIM:NORMalize (see page 215)	Lets you normalize the gain of an InfiniiSim transfer function.
:RSTate? (see page 772)	Returns the run state.
:SBUS<N>:CAN Commands (see page 787)	These commands control the CAN serial decode bus viewing, mode, and other options.

What's New in Version 4.50

New command descriptions for Version 4.50 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:ACQuire:POINts:DIGital? (see page 166)	Supports the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
:ACQuire:SRATe:DIGital (see page 175)	Supports the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
:ACQuire:SRATe:DIGital:AUTO (see page 176)	Supports the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
Bus Commands (see page 177)	The :BUS<N> commands support the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
Digital Commands (see page 275)	The :DIGital commands support the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
:DISable DIGital (see page 761)	The :DISable DIGital command supports the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
:ENABle DIGital (see page 762)	The :ENABle DIGital command supports the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels.
:FUNCtion<F>:ADEMod (see page 331)	Sets the math function to show the amplitude envelope for an amplitude modulated (AM) input signal.
:MEASurement<N>:NAME (see page 738)	Lets you give specific names to measurements displayed on the oscilloscope's screen.
:MEASurement<N>:SOURce (see page 739)	Changes the source of an existing measurement.
:MTESt:FOLDing:FAST (see page 458)	Turns the "Fast, Worst Case Only" real-time eye display option ON or OFF.
Pod Commands (see page 741)	The :POD commands support the new MSO 90000 X-Series mixed-signal oscilloscopes which provide up to 16 digital input channels split into two pods.

Changed Commands

Command	Description
:BLANk (see page 758)	The command now supports digital channels, pods, and buses.
:DIGitize (see page 760)	The command now supports digital channels and pods.

Command	Description
:DISK:SAVE:WAVeform (see page 296)	There is now the H5INt format parameter which saves waveform data as integers within the H5 file.
:MEASure:DELTatime (see page 528)	Digital channels can now be specified as sources.
:MEASure:DUTYcycle (see page 532)	Digital channels can now be specified as the source.
:MEASure:EDGE (see page 534)	Digital channels can now be specified as the source.
:MEASure:FREQuency (see page 548)	Digital channels can now be specified as the source.
:MEASure:NWIDth (see page 600)	Digital channels can now be specified as the source.
:MEASure:PERiod (see page 606)	Digital channels can now be specified as the source.
:MEASure:PWIDth (see page 614)	Digital channels can now be specified as the source.
:MEASure:SETuptime (see page 645)	Digital channels can now be specified as sources.
:MEASure:SOURce (see page 648)	Digital channels can now be specified as sources.
:MEASure:TEDGe (see page 650)	Digital channels can now be specified as the source.
:MEASure:THResholds:ABSolute (see page 652)	Digital channels can now be specified as the source.
:MEASure:THResholds:HYSTeresis (see page 666)	Digital channels can now be specified as the source.
:MEASure:THResholds:METHod (see page 668)	Digital channels can now be specified as the source.
:MEASure:THResholds:TOPBase:ABSolute (see page 696)	Digital channels can now be specified as the source.
:MEASure:THResholds:TOPBase:METHod (see page 698)	Digital channels can now be specified as the source.
:SBUS<N>:IIC:SOURce:CLOCk (see page 794)	Digital channels can now be specified as the source.
:SBUS<N>:IIC:SOURce:DATA (see page 795)	Digital channels can now be specified as the source.
:SBUS<N>:SPI:SOURce:CLOCk (see page 801)	Digital channels can now be specified as the source.

Command	Description
:SBUS<N>:SPI:SOURce:DATA (see page 802)	Digital channels can now be specified as the source.
:SBUS<N>:SPI:SOURce:FRAME (see page 803)	Digital channels can now be specified as the source.
:SBUS<N>:SPI:SOURce:MISO (see page 804)	Digital channels can now be specified as the source.
:SBUS<N>:SPI:SOURce:MOSI (see page 805)	Digital channels can now be specified as the source.
:STATus? (see page 776)	The query now supports digital channels, pods, and buses.
:TRIGger:DELay:ARM:SOURce (see page 906)	Digital channels can now be specified as the source.
:TRIGger:DELay:EDELay:SOURce (see page 909)	Digital channels can now be specified as the source.
:TRIGger:DELay:TRIGger:SOURce (see page 913)	Digital channels can now be specified as the source.
:TRIGger:EDGE:SOURce (see page 917)	Digital channels can now be specified as the source.
:TRIGger:GLITch:SOURce (see page 917)	Digital channels can now be specified as the source.
:TRIGger:PATTern:LOGic (see page 924)	Digital channels can now be specified as the source.
:TRIGger:PWIDth:SOURce (see page 928)	Digital channels can now be specified as the source.
:TRIGger:STATe:CLOCk (see page 953)	Digital channels can now be specified as the source.
:TRIGger:STATe:LOGic (see page 954)	Digital channels can now be specified as the source.
:VIEW (see page 782)	The command now supports digital channels, pods, and buses.
:WAVeform:SOURce (see page 1026)	The command now supports digital pods and buses.

What's New in Version 4.30

New command descriptions for Version 4.30 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:FUNCtion<F>:DELay (see page 334)	Defines a function waveform that is the source waveform delayed in time.
:FUNCtion<F>:GATing (see page 346)	Defines a horizontal gating function of another waveform (similar to horizontal zoom). Measurements on horizontal gating functions are essentially gated measurements.
:MEASure:CLOCk:METHod:EDGE (see page 506)	Specifies which edge(s) of the data signal are used to recover a clock.
:MEASure:RJDJ:CLOCk (see page 630)	When ON, it forces the pattern to be a clock and sets the jitter for edges not examined to zero (0).

Changed Commands

Command	Description
Function Commands (see page 325)	You can now define up to 16 functions.

What's New in Version 4.20

New command descriptions for Version 4.20 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:DISPlay:CGRade:SCHeme (see page 303)	Lets you set the color grade scheme to CLASsic or TEMP.
:FUNCtion<N>:FFT:TDELay (see page 341)	Sets the time delay for the FFT phase function.
:MEASure:CLOCk:METHod:JTF (see page 508)	Specifies the clock recovery PLL's response in terms of the Jitter Transfer Function's (JTF) 3 dB bandwidth.
:MEASure:CLOCk:METHod:OJTF (see page 510)	Specifies the clock recovery PLL's response in terms of the Observed Jitter Transfer Function's (OJTF) 3 dB bandwidth.

Changed Commands

Command	Description
:MEASure:CLOCk:METHod (see page 502)	The command options for specifying clock recovery PLL options have been moved to the new commands :MEASure:CLOCk:METHod:JTF and :MEASure:CLOCk:METHod:OJTF.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Description
:MEASure:DDPWS	:MEASure:RJDJ:ALL? (see page 624)	The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements.

What's New in Version 4.10

New command descriptions for Version 4.10 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:MEASure:NOISe:METHod (see page 591)	Lets you select the method for random noise (RN) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.
:MEASure:NOISe:REPort (see page 592)	When BOTH is selected for :MEASure:NOISe:METHod, you can select SPECtral or TAILfit to specify which method is used for the reports.
:MEASure:RJDJ:METHod (see page 633)	Lets you select the method for random jitter (RJ) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.
:MEASure:RJDJ:REPort (see page 636)	When BOTH is selected for :MEASure:RJDJ:METHod, you can select SPECtral or TAILfit to specify which method is used for the reports.
:MEASure:TIEFilter:SHAPe (see page 704)	Specifies the shape of the TIE filter edge(s).

Changed Commands

Command	Description
:MEASure:NOISe:ALL (see page 587)	New results can be returned depending on the :MEASure:NOISe:METHod and :MEASure:NOISe:REPort settings.
:MEASure:RJDJ:ALL (see page 624)	New results can be returned depending on the :MEASure:RJDJ:METHod and :MEASure:RJDJ:REPort settings.

What's New in Version 4.00

New command descriptions for Version 4.00 of the Infiniium 90000 Series oscilloscope software appear below.

New Commands

Command	Description
:ACQuire:REDGe (see page 167)	For 50 GHz and 63 GHz bandwidth models of the 90000 Q-Series oscilloscopes, this command enables or disables the RealEdge channel inputs.
:DISK:SAVE:PRECprobe (see page 294)	Saves PrecisionProbe/Cable data to a file.
:ISCan:ZONE:HIDE (see page 405)	Lets you hide or show all InfiniiScan zones on the display.

Changed Commands

Command	Description
:ACQuire:BANDwidth (see page 152)	There is now a MAX option for selecting the maximum bandwidth.
:MTESt:FOLDing:BITS (see page 457)	There is now a PATTern option for specifying bit pattern qualification for the real-time eye display.

What's New in Version 3.50

New command descriptions for Version 3.50 of the Infiniium 90000A/90000X Series oscilloscope software appear below.

New Commands

Command	Description
:CHANnel<N>:PROBe:AUTozero (see page 225)	Initiates the N2893A probe's auto degauss/ offset cal.
:CHANnel<N>:PROBe:HEAD:VTERm (see page 238)	Sets the termination voltage for the N5444A probe head.
:CHANnel<N>:PROBe:MODE (see page 240)	Sets the N2750A probe's InfiniiMode configuration.
:FUNCtion<N>:MTRend (see page 358)	New Meas Trend math function.
:FUNCtion<N>:MHISto gram (see page 356)	New Meas Histogram math function.
:LISTer Commands (see page 419)	For displaying and retrieving data from the serial decode listings.
:MEASure:HISTogram:RESolution (see page 561)	The bin width value of one bar in the histogram.
:MEASure:NOISe (see page 586)	Adds a Noise measurement to the oscilloscope display (like **Measure > Data > Noise** from the front panel) or gets the measured noise value.
:MEASure:NOISe:ALL (see page 587)	Returns the NOISe measurement results for the "zeros" or "ones" level.
:MEASure:NOISe:BANDwidth (see page 589)	Sets the type of filtering used to separate the data dependent noise from the random noise and the periodic noise.
:MEASure:NOISe:LOCation (see page 590)	Specifies the noise measurement location within the bit where 0% is the beginning of the bit, 50% is the middle of the bit, and 100% is the end of the bit.
:MEASure:NOISe:RN (see page 593)	Specifies a known amount of random noise.
:MEASure:NOISe:SCOPe:RN (see page 594)	Specifies the removal of the oscilloscope's calibrated random noise from the reported RN.
:MEASure:NOISe:STATe (see page 595)	Enables or disables the NOISe measurements.
:MEASure:NOISe:UNITs (see page 596)	Sets the unit of measure for NOISe measurements to volts or unit amplitude.
:MEASure:RJDJ:RJ (see page 637)	Specifies a known amount of random jitter.

Command	Description
:MEASure:RJDJ:SCOPe:RJ (see page 638)	Specifies the removal of the oscilloscope's calibrated random jitter from the reported RJ.
:SBUS<N> Commands (see page 783)	For setting up IIC and SPI serial decode.

Changed Commands

Command	Description
:CHANnel<N>:PROBe:HEAD:SELect (see page 237)	Now lets you select probe heads by the labels given with the :CHANnel<N>:PROBe:HEAD:ADD command.
:MEASure:HISTogram:HITS (see page 550)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:M1S (see page 551)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:M2S (see page 552)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:M3S (see page 553)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:MAX (see page 554)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:MEAN (see page 555)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:MEDian (see page 556)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:MIN (see page 557)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:MODE (see page 558)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:PEAK (see page 559)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:PP (see page 560)	Can now use this command with Meas Histogram math functions.
:MEASure:HISTogram:STDDev (see page 562)	Can now use this command with Meas Histogram math functions.
:MEASure:RJDJ:ALL (see page 624)	There are two possible additional measurement results, Scope RN(rms) and DDPWS.
:TRIGger:MODE (see page 896)	Added the SBUS1, SBUS2, SBUS3, and SBUS4 selections for triggering on serial buses.

What's New in Version 3.20

New command descriptions for Version 3.20 of the Infiniium 90000A/90000X Series oscilloscope software appear below.

New Commands

Command	Description
:MARKer:MEASurement:MEASurement (see page 425)	Specifies which measurement markers track (when the :MARKer:MODE is set to MEASurement).
:MEASure:CLOCk:METHod:ALIGn (see page 504)	Lets you specify clock edges either center aligned with data or edge aligned with data when using an explicit method of clock recovery.

What's New in Version 3.11

New command descriptions for Version 3.11 of the Infiniium 90000A/90000X Series oscilloscope software appear below.

New Commands

Command	Description
:CHANnel<N>:ISIM:PEXTraction (see page 216)	Selects a channel's InfiniiSim port extraction.
:MEASure:HISTogram:MODE (see page 558)	Returns the measurement histogram's Mode value.

Changed Commands

Command	Description
:BUS:B<N>:TYPE (see page 178)	The MPHY protocol type has been added for the MIPI M-PHY serial decode selection.
:FUNCtion<N>:FFT:WINDow (see page 342)	The HAMMing window mode is now a valid selection.
:MEASure:JITTer:SPECtrum:WINDow (see page 575)	The HAMMing window mode is now a valid selection.

What's New in Version 3.10

New command descriptions for Version 3.10 of the Infiniium 90000A/90000X Series oscilloscope software appear below.

New Commands

Command	Description
:CHANnel<N>:PROBe:ACCal (see page 223)	Sets the type of AC response probe calibration to use.
:CHANnel<N>:PROBe:PRECprobe:BANDwidth (see page 241)	Specifies how the limit of PrecisionProbe or PrecisionCable correction/boosting is determined..
:CHANnel<N>:PROBe:PRECprobe:CALibration (see page 242)	Specifies the name of the PrecisionProbe or PrecisionCable calibration to use for the specified channel and probe.
:CHANnel<N>:PROBe:PRECprobe:MODE (see page 243)	Selects between PrecisionProbe or PrecisionCable AC response probe calibration.
:CHANnel<N>:PROBe:PRECprobe:ZSRC (see page 244)	Specifies how PrecisionProbe characterizes the time domain and frequency domain response.
:DISPlay:STATus:COL (see page 319)	Used to position the real time eye and InfiniiScan Zone Trigger status labels.
:DISPlay:STATus:ROW (see page 320)	Used to position the real time eye and InfiniiScan Zone Trigger status labels.
:MEASure:DDPWS (see page 1057)	For measuring Data Dependent Pulse Width Shrinkage.
:MEASure:EDGE (see page 534)	For measuring the edge time relative to the reference location.
:MEASure:JITTer:SPECtrum:VERTical:TYPE (see page 574)	Lets you select either a LINear or a LOGarithmic vertical scale for the jitter spectrum plot.
:MEASure:RJDJ:APLength? (see page 626)	Returns the determined RjDj pattern length.
:SPRocessing:CTLequalizer:NUMPoles (see page 818)	Selects either a 2 Pole or 3 Pole Continuous Time Linear Equalization.
:SPRocessing:CTLequalizer:P3 (see page 821)	Sets the Pole 3 frequency for the Continuous Time Linear Equalization.

Changed Commands

Command	Description
:ACQuire:INTerpolate (see page 158)	The INT1, INT2, INT4, INT8, INT16 options have been added for specifying the 1, 2, 4, 8, or 16 point Sin(x)/x interpolation ratios.
:MEASure:RJDJ:BER (see page 628)	You can now set J2 and J9 jitter BER levels.
:MEASure:VRMS (see page 730)	The VOLT and DBM parameters have been added for specifying the measurement units.
:MEASure:WINDow (see page 737)	The short form of the command was changed from :MEAS:WIN to :MEAS:WIND.

2 Setting Up

Step 1. Install Agilent IO Libraries Suite software 46
Step 2. Connect and set up the oscilloscope 47
Step 3. Verify the oscilloscope connection 48

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

Note that Agilent IO Libraries Suite software comes installed on Infiniium oscilloscopes, and it is possible to control the oscilloscope from programs running on the instrument.

Step 1. Install Agilent IO Libraries Suite software

To install the Agilent IO Libraries Suite software on a controller PC:

1 Download the Agilent IO Libraries Suite software from the Agilent web site at:

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

2 Run the setup file, and follow its installation instructions.

Note that Agilent IO Libraries Suite software comes installed on Infiniium oscilloscopes.

Step 2. Connect and set up the oscilloscope

The Infiniium 90000 Series oscilloscopes have three different interfaces you can use for programming:

- USB (device port, square connector).
- LAN. To configure the LAN interface, set up the Infiniium oscilloscope on the network as you would any other computer with the Windows XP operating system.
- GPIB (on the 90000A Series and 90000 X-Series — the 90000Q-Series would require the N4865A GPIB-to-LAN adapter).

When installed, these interfaces are always active.

Using the USB (Device) Interface

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

This is a USB 2.0 high-speed port.

Using the LAN Interface

1 If the controller PC is not already connected to the local area network (LAN), do that first.

2 Contact your network administrator about adding the oscilloscope to the network.

Setting up an Infiniium oscilloscope on a network is the same as setting up any other computer with the Windows XP operating system.

3 Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the oscilloscope.

Using the GPIB Interface

1 Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the oscilloscope.

2 Configure the oscilloscope's GPIB interface:

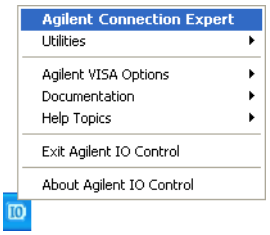
a In the Agilent Infiniium software interface, choose **Utilities > Remote Setup...**.

b In the Remote Setup dialog, use the **GPIB Address** field to set the address.

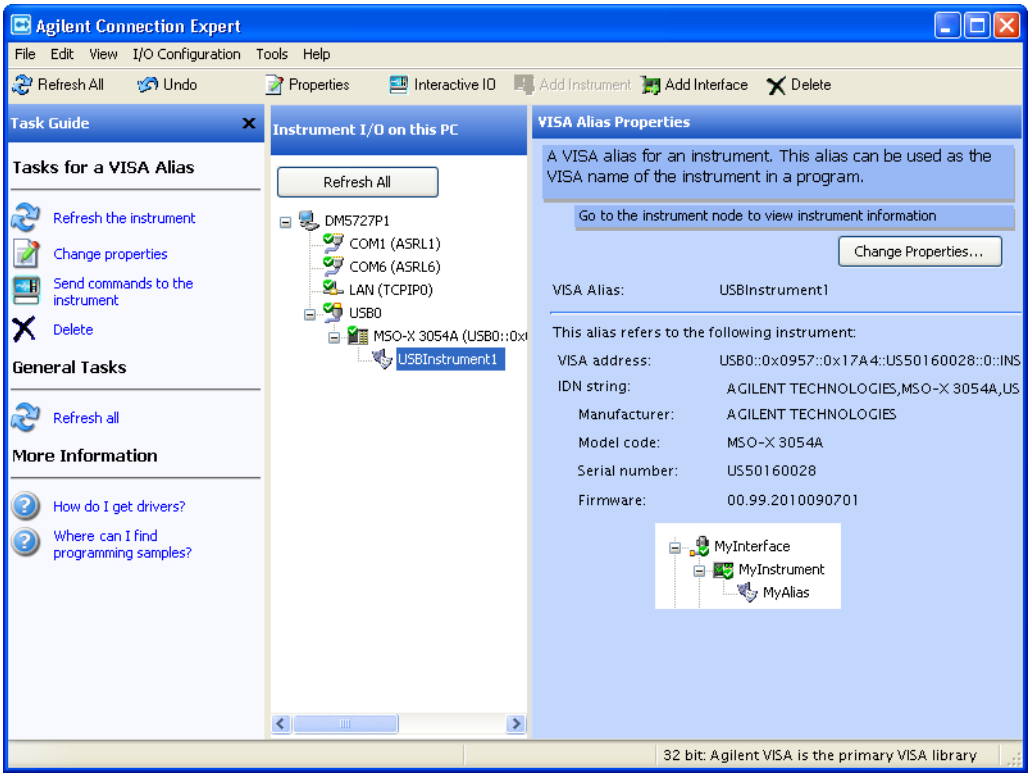
Step 3. Verify the oscilloscope connection

NOTE Make sure the Agilent Infiniium software is running on the oscilloscope. It must be running before you can make a connection.

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

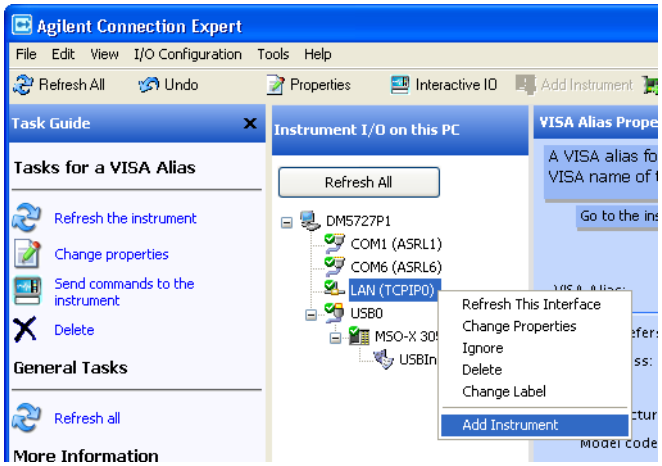


2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)

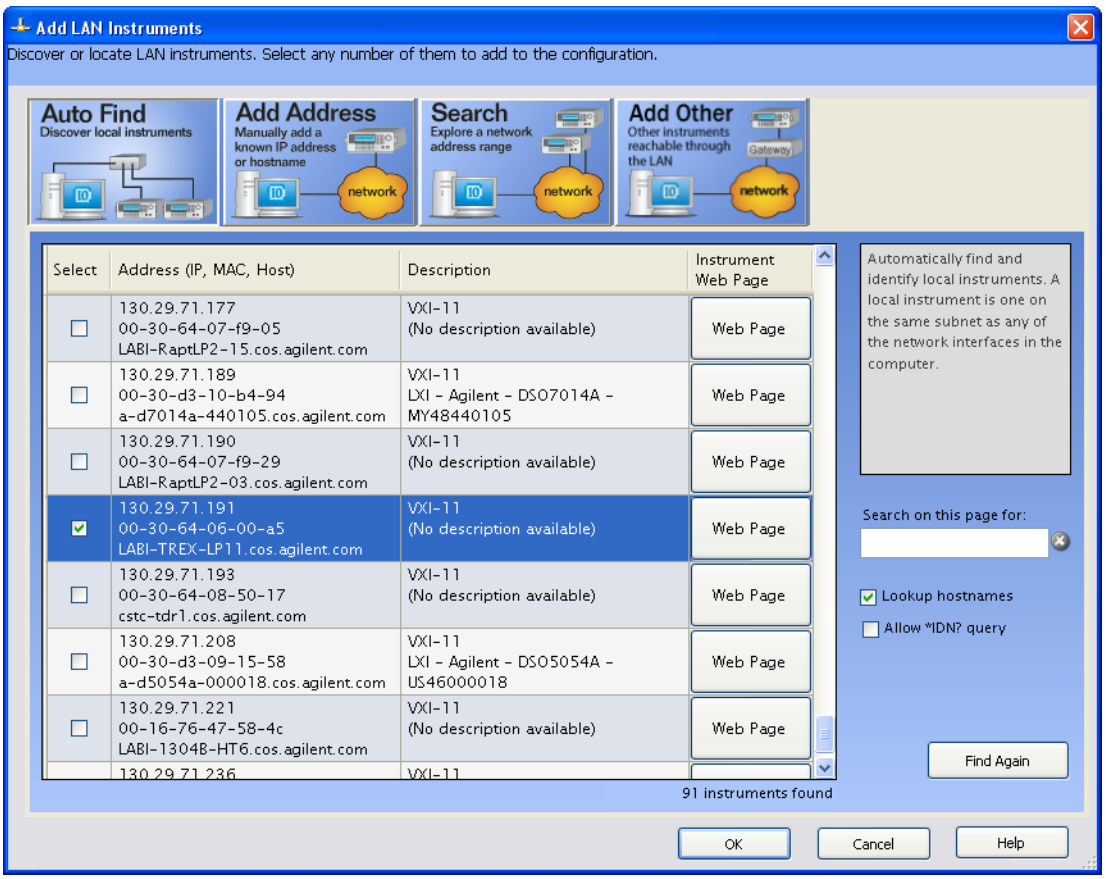


You must manually add instruments on LAN interfaces:

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



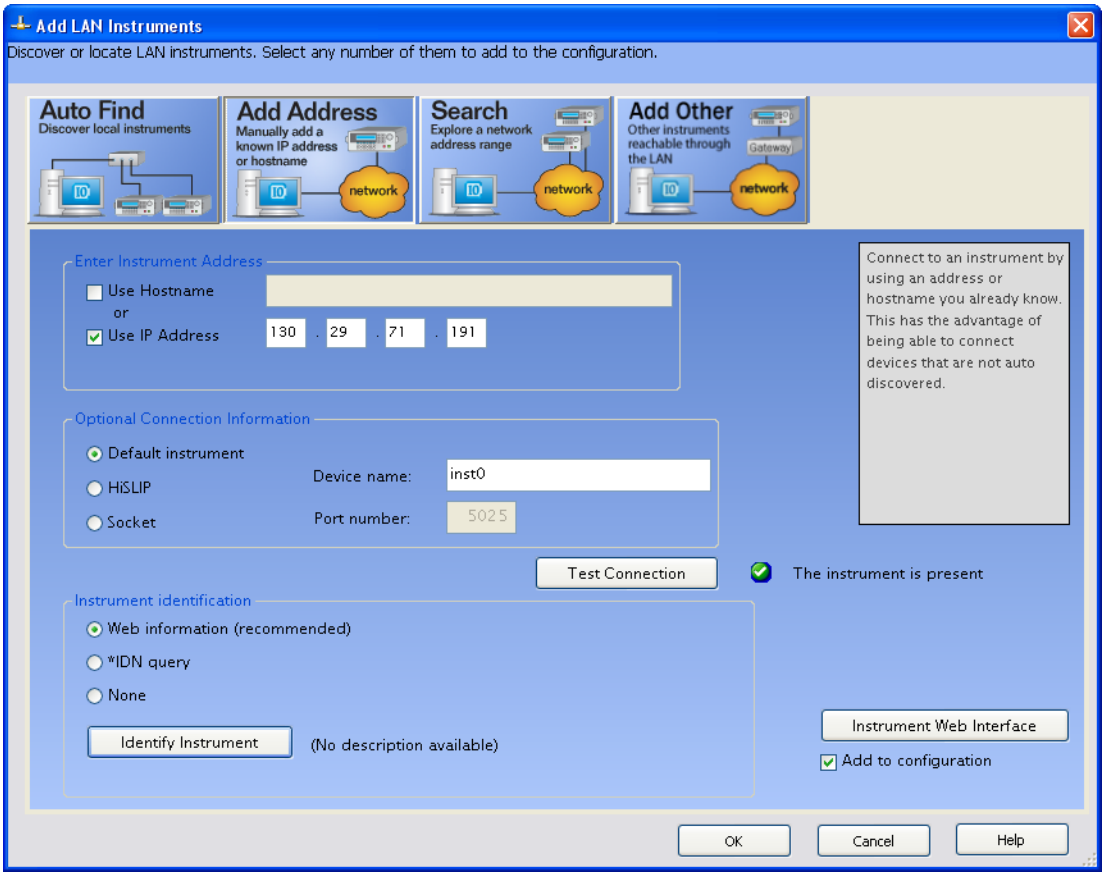
b If the oscilloscope is on the same subnet, select it, and click **OK**.



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

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

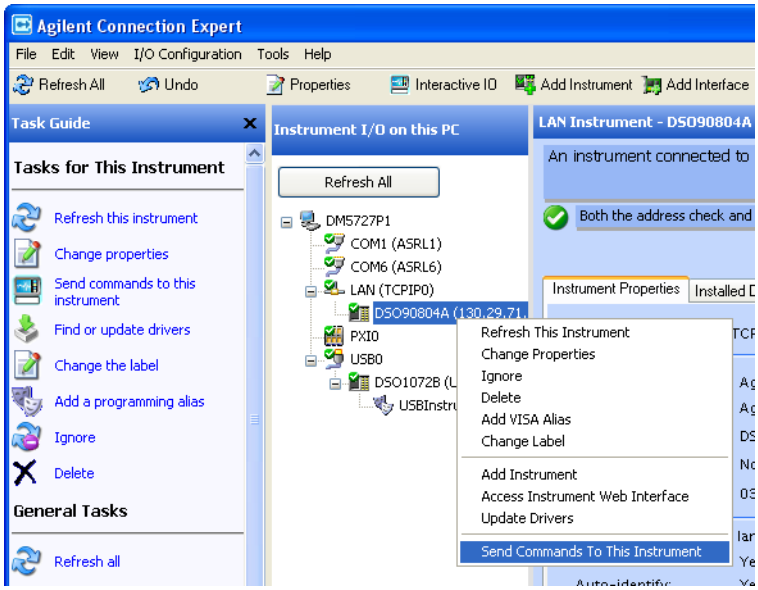
ii Click **Test Connection**.



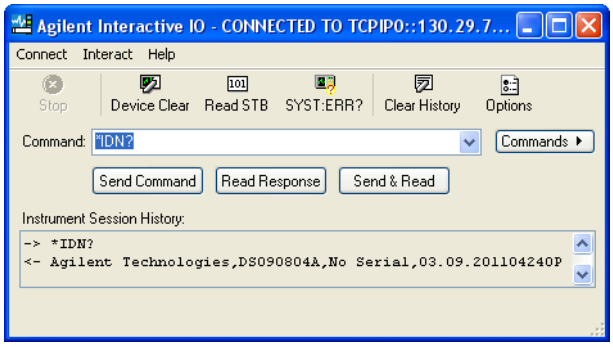
iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

3 Test some commands on the instrument:

a Right-click on the instrument and choose **Send Commands To This Instrument** from the popup menu.



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



c Choose **Connect > Exit** from the menu to exit the Agilent Interactive IO application.

4 In the Agilent Connection Expert application, choose **File > Exit** from the menu to exit the application.

3 Introduction to Programming

Communicating with the Oscilloscope 55
Instructions 56
Instruction Header 57
White Space (Separator) 58
Braces 59
Ellipsis 60
Square Brackets 61
Command and Query Sources 62
Program Data 63
Header Types 64
Query Headers 66
Program Header Options 67
Character Program Data 68
Numeric Program Data 69
Embedded Strings 70
Program Message Terminator 71
Common Commands within a Subsystem 72
Selecting Multiple Subsystems 73
Programming Getting Started 74
Referencing the IO Library 75
Opening the Oscilloscope Connection via the IO Library 76
Initializing the Interface and the Oscilloscope 77
Example Program 79
Using the DIGITIZE Command 80
Receiving Information from the Oscilloscope 82
String Variable Example 83
Numeric Variable Example 84
Definite-Length Block Response Data 85
Multiple Queries 86
Oscilloscope Status 87

This chapter introduces the basics for remote programming of an oscilloscope. The programming commands in this manual conform to the IEEE 488.2 Standard Digital Interface for Programmable Instrumentation. The programming commands provide the means of remote control.

Basic operations that you can do with a computer and an oscilloscope include:

- Set up the oscilloscope.
- Make measurements.
- Get data (waveform, measurements, and configuration) from the oscilloscope.
- Send information, such as waveforms and configurations, to the oscilloscope.

You can accomplish other tasks by combining these functions.

NOTE

Example Programs are Written in Visual Basic for Applications (VBA) and C

The programming examples for individual commands in this manual are written in Visual Basic for Applications (VBA) and C.

Communicating with the Oscilloscope

Computers communicate with the oscilloscope by sending and receiving messages over a remote interface, such as a GPIB card or a Local Area Network (LAN) card. Commands for programming normally appear as ASCII character strings embedded inside the output statements of a "host" language available on your computer. The input commands of the host language are used to read responses from the oscilloscope.

For example, the VISA COM library provides the WriteString() method for sending commands and queries. After a query is sent, the response can be read using the ReadString() method. The ReadString() method passes the value across the bus to the computer and places it in the designated variable.

The following WriteString() method sends a command that sets the channel 1 scale value to 500 mV:

```
myScope.WriteString ":CHANNEL1:SCALE 500E-3"
```

The VISA COM library setup is explained on the following pages.

NOTE

Use the Suffix Multiplier Instead

Using "mV" or "V" following the numeric voltage value in some commands will cause Error 138 - Suffix not allowed. Instead, use the convention for the suffix multiplier as described in Chapter 5, “Message Communication and System Functions,” starting on page 101.

Instructions

Instructions, both commands and queries, normally appear as strings embedded in a statement of your host language, such as Visual Basic for Applications (VBA), Visual Basic .NET, C#, C, etc.

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 with the :SYSTem:SETup command. There are only a few instructions that use block data.

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provides additional information to clarify the meaning of the instruction.

Instruction Header

The instruction header is one or more command mnemonics separated by colons (:). They represent the operation to be performed by the oscilloscope. See Chapter 8, “Programming Conventions,” starting on page 139 for more information.

Queries are formed by adding a question mark (?) to the end of the header. Many instructions can be used as either commands or queries, depending on whether or not you include the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

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. In this manual, white space is defined as one or more spaces. ASCII defines a space to be character 32 in decimal.

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.

Ellipsis

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

Square Brackets

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

Command and Query Sources

Many commands and queries require that a source be specified. Depending on the command or query and the model number of Infiniium oscilloscope being used, some of the sources are not available. The following is a list of sources:

CHANnel1	FUNCtion1	WMEMory1	
CHANnel2	FUNCtion2	WMEMory2	
CHANnel3	FUNCtion3	WMEMory3	
CHANnel4	FUNCtion4	WMEMory4	
CLOCk	MTRend	MSPectrum	HISTogram

Program Data

Program data is used to clarify the meaning of the command or query. It provides 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 and the values they accept.

When there is more than one data parameter, they are separated by commas (,). You can add spaces around the commas to improve readability.

Header Types

There are three types of headers:

- "Simple Command Header" on page 64
- "Compound Command Header" on page 64
- "Common Command Header" on page 65

See Also

- "Combining Commands in the Same Subsystem" on page 65
- "Duplicate Mnemonics" on page 65

Simple Command Header

Simple command headers contain a single mnemonic. AUTOSCALE and DIGITIZE are examples of simple command headers typically used in this oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

For example:

```
":AUTOSCALE"
```

When program data must be included with the simple command header (for example, :DIGITIZE CHAN1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

For example:

```
":DIGITIZE CHANNEL1,FUNCTION2"
```

Compound Command Header

Compound command headers are a combination of two 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"
```

Combining Commands in the Same Subsystem

To execute more than one command within the same subsystem, use a semi-colon (;) to separate the commands:

```
:<subsystem>:<command><separator><data>;<command><separator>
<data><terminator>
```

For example:

```
:CHANNEL1:INPUT DC;BWLIMIT ON
```

Common Command Header

Common command headers, such as clear status, control the IEEE 488.2 functions within the oscilloscope. The 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.

Duplicate Mnemonics

Identical function mnemonics can be used for more than one subsystem. For example, you can use the function mnemonic RANGE to change both the vertical range and horizontal range:

To set the vertical range of channel 1 to 0.4 volts full scale:

```
:CHANNEL1:RANGE .4
```

To set the horizontal time base to 1 second full scale:

```
:TIMEBASE:RANGE  1
```

In these examples, CHANNEL1 and TIMEBASE are subsystem selectors, and determine the range type being modified.

Query Headers

A command header immediately followed by a question mark (?) is a query. After receiving a query, the oscilloscope interrogates the requested subsystem and places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the bus to the designated listener (typically a computer).

For example, with VISA COM library and Visual Basic for Applications (VBA) language, the query:

```
myScope.WriteString ":TIMEBASE:RANGE?"
```

places the current time base setting in the output queue.

The computer input statement:

```
varRange = myScope.ReadNumber
```

passes the value across the bus to the computer and places it in the variablevar Range.

You can use queries to find out how the oscilloscope is currently configured and to get results of measurements made by the oscilloscope. For example, the query:

```
:MEASURE:RISETIME?
```

tells the oscilloscope to measure the rise time of your waveform and place the result in the output queue.

The output queue must be read before the next program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow it with an input statement.

With the VISA COM library and Visual Basic for Applications (VBA) language, this is usually done with a ReadString() or ReadNumber() method. These methods read the result of the query and place the result in a specified variable.

NOTE

Handle Queries Properly

If you send another command or query before reading the result of a query, the output buffer is cleared and the current response is lost. This also generates a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

Program Header Options

You can send program headers using any combination of uppercase or lowercaseASCII characters. Oscilloscope responses, however, are always returned in uppercase.

You may send program command and query headers in either long form (complete spelling), short form (abbreviated spelling), or any combination of long form and short form. For example:

":TIMebase:DELay 1E-6" is the long form.

":TIM:DEL 1E-6" is the short form.

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.

NOTE

Using Long Form or Short Form

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of computer memory needed for program storage and reduces I/O activity.

The rules for the short form syntax are described in Chapter 8, “Programming Conventions,” starting on page 139.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMEBASE:REFERENCE command can be set to left, center, or right. The character program data in this case may be LEFT, CENTER, or RIGHT. The command :TIMEBASE:REFERENCE RIGHT sets the time base reference to right.

The available mnemonics for character program data are always included with the instruction's syntax definition. You may send either the long form of commands, or the short form (if one exists). You may mix uppercase and lowercase letters freely. When receiving 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 can use exponential notation or suffix multipliers to indicate the numeric value. The following numbers are all equal:

28 = 0.28E2 = 280E-1 = 28000m = 0.028K = 28E-3K

When a syntax definition specifies that a number is an integer, it means that the number should be whole. Any fractional part is ignored and truncated. Numeric data parameters that accept fractional values are called real numbers.

All numbers are expected to be strings of ASCII characters.

- 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). The number of bytes is figured automatically when you include the entire instruction in a string.

Embedded Strings

Embedded strings contain groups of alphanumeric characters which are treated as a unit of data by the oscilloscope. An example of this is the line of text written to the advisory line of the oscilloscope with the :SYSTEM:DSP command:

```
:SYSTEM:DSP ""This is a message.""
```

You may delimit embedded strings with either single (') or double (") quotation marks. These strings are case-sensitive, and spaces are also legal characters.

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 Like EOS and EOT

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Common Commands within a Subsystem

Common commands can be received and processed by the oscilloscope whether they are sent over the bus as separate program messages or within other program messages. If you have selected a subsystem, and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
":ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the oscilloscope, the oscilloscope turns averaging on, then clears the status information without leaving the selected subsystem.

If some other type of command is received within a program message, you must re-enter the original subsystem after the command. For example, the program message

```
":ACQUIRE:AVERAGE ON;:AUTOSCALE;:ACQUIRE:AVERAGE:COUNT 1024"
```

turns averaging on, completes the autoscale operation, then sets the acquire average count. Here, :ACQUIRE must be sent again after AUTOSCALE to re-enter the ACQUIRE subsystem and set the count.

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 lets you enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

:CHANNEL1:RANGE 0.4;:TIMEBASE:RANGE 1

NOTE

You can Combine Compound and Simple Commands

Multiple program commands may be any combination of compound and simple commands.

Programming Getting Started

The remainder of this chapter explains how to set up the oscilloscope, how to retrieve setup information and measurement results, how to digitize a waveform, and how to pass data to the computer. Chapter 25, “Measure Commands,” starting on page 479 describes getting measurement data from the oscilloscope.

Referencing the IO Library

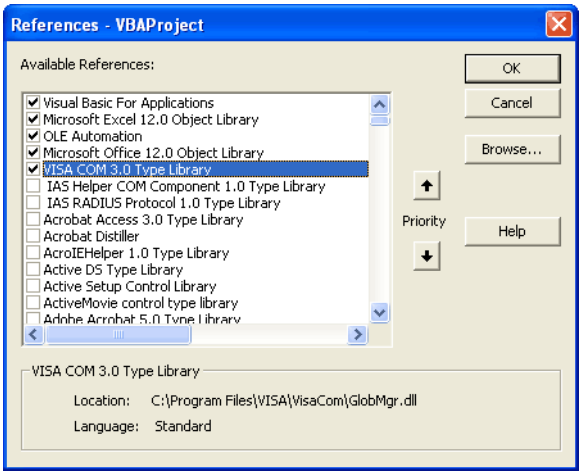
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

In C/C++, you must tell the compiler where to find the include and library files (see the Agilent IO Libraries Suite documentation for more information).

To reference the Agilent VISA COM library in Visual Basic for Applications (VBA, which comes with Microsoft Office products like Excel):

1 Choose **Tools > References...** from the main menu.

2 In the References dialog, check the "VISA COM 3.0 Type Library".



3 Click **OK**.

To reference the Agilent VISA COM library in Microsoft Visual Basic 6.0:

1 Choose **Project > References...** from the main menu.

2 In the References dialog, check the "VISA COM 3.0 Type Library".

3 Click **OK**.

Opening the Oscilloscope Connection via the IO Library

PC controllers communicate with the oscilloscope by sending and receiving messages over a remote interface. Once you have opened a connection to the oscilloscope over the remote interface, programming instructions normally appear as ASCII character strings embedded inside write statements of the programing language. Read statements are used to read query responses from the oscilloscope.

For example, when using the Agilent VISA COM library in Visual Basic (after opening the connection to the instrument using the ResourceManager object's Open method), the FormattedIO488 object's WriteString, WriteNumber, WriteList, or WriteIEEEBlock methods are used for sending commands and queries. After a query is sent, the response is read using the ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

The following Visual Basic statements open the connection and send a command that turns on the oscilloscope's label display.

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

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

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

' Send a command.
myScope.WriteString ":DISPlay:LABel ON"
```

The ":DISPLAY:LABEL ON" in the above example is called a *program message*. Program messages are explained in more detail in "Instructions" on page 56.

Initializing the Interface and the Oscilloscope

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

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

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

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

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

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

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

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

NOTE

Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in Chapter 13, "Common Commands," starting on page 251.

Refer to the Agilent IO Libraries Suite documentation for information on initializing the interface.

See Also

- "Autoscale" on page 77
- "Setting Up the Oscilloscope" on page 78

Autoscale

The AUTOSCALE feature of Agilent Technologies digitizing oscilloscopes performs a very useful function on unknown waveforms by automatically setting up the vertical channel, time base, and trigger level of the oscilloscope.

The syntax for the autoscale function is:

```
:AUTOSCALE<terminator>
```

Setting Up the Oscilloscope

A typical oscilloscope setup configures the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope.

A typical example of the commands sent to the oscilloscope are:

```
:CHANNEL1:PROBE 10; RANGE 16;OFFSET 1.00<terminator>

:SYSTEM:HEADER OFF<terminator>

:TIMEBASE:RANGE 1E-3;DELAY 100E-6<terminator>
```

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

Example Program

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

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear
myScope.IO.Timeout = 15000   ' Set interface timeout to 15 seconds.

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

' Set the time base mode to normal with the horizontal time at
' 50 ms/div with 0 s of delay referenced at the center of the
' graticule.
myScope.WriteString ":TIMebase:RANGe 5E-4"    ' Time base to 500 us/div.
myScope.WriteString ":TIMebase:DELay 0"       ' Delay to zero.
myScope.WriteString ":TIMebase:REFerence CENTer"    ' Display ref. at
                                                    ' center.

' Set the vertical range to 1.6 volts full scale with center screen
' at -0.4 volts with 10:1 probe attenuation and DC coupling.
myScope.WriteString ":CHANnel1:PROBe 1.0"     ' Probe attenuation
                                              ' to 1:1.
myScope.WriteString ":CHANnel1:RANGe 1.6"     ' Vertical range
                                              ' 1.6 V full scale.
myScope.WriteString ":CHANnel1:OFFSet -0.4"   ' Offset to -0.4.
myScope.WriteString ":CHANnel1:INPut DC"      ' Coupling to DC.

' Configure the instrument to trigger at -0.4 volts with normal
' triggering.
myScope.WriteString ":TRIGger:SWEep NORMal"    ' Normal triggering.
myScope.WriteString ":TRIGger:LEVel CHAN1,-0.4"    ' Trigger level to -0.
4.
myScope.WriteString ":TRIGger:MODE EDGE"       ' Edge triggering
myScope.WriteString ":TRIGger:EDGE:SLOPe POSitive" ' Trigger on pos. slo
pe.

' Configure the instrument for normal acquisition.
myScope.WriteString ":ACQuire:MODE RTIMe"    ' Normal acquisition.
myScope.WriteString ":SYSTem:HEADer OFF"    ' Turn system headers off.
myScope.WriteString ":DISPlay:GRATicule FRAMe"    ' Grid off.
```

Using the DIGITIZE Command

The DIGITIZE command is a macro that captures data using the acquisition (ACQUIRE) subsystem. When the digitize process is complete, the acquisition is stopped. You can measure the captured data by using the oscilloscope or by transferring the data to a computer for further analysis. The captured data consists of two parts: the preamble and the waveform data record.

After changing the oscilloscope configuration, the waveform buffers are cleared. Before doing a measurement, you should send the DIGITIZE command to ensure new data has been collected.

You can send the DIGITIZE command with no parameters for a higher throughput. Refer to the DIGITIZE command in Chapter 27, “Root Level Commands,” starting on page 747 for details.

When the DIGITIZE command is sent to an oscilloscope, the specified channel's waveform is digitized using the current ACQUIRE parameters. Before sending the :WAVEFORM:DATA? query to download waveform data to your computer, you should specify the WAVEFORM parameters.

The number of data points comprising a waveform varies according to the number requested in the ACQUIRE subsystem. The ACQUIRE subsystem determines the number of data points, type of acquisition, and number of averages used by the DIGITIZE command. This lets you specify exactly what the digitized information contains. The following program example shows a typical setup:

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:MODE RTIME"
myScope.WriteString ":ACQUIRE:COMPLETE 100"
myScope.WriteString ":WAVEFORM:SOURCE CHANNEL1"
myScope.WriteString ":WAVEFORM:FORMAT BYTE"
myScope.WriteString ":ACQUIRE:COUNT 8"
myScope.WriteString ":ACQUIRE:POINTS 500"
myScope.WriteString ":DIGITIZE CHANNEL1"
myScope.WriteString ":WAVEFORM:DATA?"
```

This setup places the oscilloscope into the real time sampling mode using eight averages. This means that when the DIGITIZE command is received, the command will execute until the waveform has been averaged at least eight times.

After receiving the :WAVEFORM:DATA? query, the oscilloscope will start downloading the waveform information.

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

The easiest method of receiving a digitized waveform depends on data structures, available formatting, and I/O capabilities. You must convert the data values to determine the voltage value of each point. These data values are passed starting with the left most point on the oscilloscope's display. For more information, refer to the chapter, "Waveform Commands."

When using GPIB, you may abort a digitize operation by sending a Device Clear over the bus.

Receiving Information from the Oscilloscope

After receiving a query (a command header followed by a question mark), the oscilloscope places the answer in its output queue. The answer remains in the output queue until it is read or until another command is issued. When read, the answer is transmitted across the interface to the computer.

The input statement for receiving a response message from an oscilloscope's output queue typically has two parameters; the device address and a format specification for handling the response message. For example, with the VISA COM library, to read the result of the query command :CHANNEL1:INPUT? you would use the ReadString() method:

```
Dim strSetting As String
myScope.WriteString ":CHANnel1:INPut?"
strSetting = myScope.ReadString
```

This would enter the current setting for the channel 1 coupling in the string variable strSetting.

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :MEASURE:RISETIME?, you must follow that query with an input statement.

NOTE

Handle Queries Properly

If you send another command or query before reading the result of a query, the output buffer will be cleared and the current response will be lost. This will also generate a query-interrupted error in the error queue. If you execute an input statement before you send a query, it will cause the computer to wait indefinitely.

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

String Variable Example

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

NOTE

Express String Variables Using Exact Syntax

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

This example shows the data being returned to a string variable:

```
Dim strRang As String
myScope.WriteString ":CHANNEL1:RANGE?"
strRang = myScope.ReadString
Debug.Print strRang
```

After running this program, the computer displays:

+8.00000E-01

Numeric Variable Example

This example shows the data being returned to a numeric variable:

```
Dim varRang As Variant
myScope.WriteString ":CHANnel1:RANGe?"
varRang = myScope.ReadNumber
Debug.Print "Channel 1 range: " + FormatNumber(varRang, 0)
```

After running this program, the computer displays:

.8

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. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 4000 bytes of data, the syntax would be:

```
#44000 <4000 bytes of data> <terminator>
```

The "4" following the pound sign represents the number of digits in the number of bytes, and "4000" represents the number of bytes to be transmitted.

Multiple Queries

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

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

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

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

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

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

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

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

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

Oscilloscope Status

Status registers track the current status of the oscilloscope. By checking the oscilloscope status, you can find out whether an operation has completed and is receiving triggers. Chapter 6, “Status Reporting,” starting on page 105 explains how to check the status of the oscilloscope.

4 LAN, USB, and GPIB Interfaces

LAN Interface Connector 90
GPIB Interface Connector 91
Default Startup Conditions 92
Interface Capabilities 93
GPIB Command and Data Concepts 94
Communicating Over the GPIB Interface 95
Communicating Over the LAN Interface 96
Communicating via Telnet and Sockets 98
Bus Commands 100

There are several types of interfaces that can be used to remotely program the Infiniium oscilloscope including Local Area Network (LAN) interface and GPIB interface. Telnet and sockets can also be used to connect to the oscilloscope.

LAN Interface Connector

The oscilloscope is equipped with a LAN interface RJ-45 connector on the rear panel. This allows direct connect to your network. However, before you can use the LAN interface to program the oscilloscope, the network properties must be configured. Unless you are a Network Administrator, you should contact your Network Administrator to add the appropriate client, protocols, and configuration information for your LAN. This information is different for every company.

GPIB Interface Connector

The 90000A Series and 90000 X-Series oscilloscopes are equipped with a GPIB interface connector on the rear panel (the 90000Q-Series requires the N4865A GPIB-to-LAN adapter). This allows direct connection to a GPIB equipped computer. You can connect an external GPIB compatible device to the oscilloscope by installing a GPIB cable between the two units. Finger tighten the captive screws on both ends of the GPIB cable to avoid accidentally disconnecting the cable during operation.

A maximum of fifteen GPIB compatible instruments (including a computer) can be interconnected in a system by stacking connectors. This allows the oscilloscopes to be connected in virtually any configuration, as long as there is a path from the computer to every device operating on the bus.

CAUTION Avoid stacking more than three or four cables on any one connector. Multiple connectors produce leverage that can damage a connector mounting.

Default Startup Conditions

The following default conditions are established during power-up:

- The Request Service (RQS) bit in the status byte register is set to zero.
- All of the event registers are cleared.
- The Standard Event Status Enable Register is set to 0xFF hex.
- Service Request Enable Register is set to 0x80 hex.
- The Operation Status Enable Register is set to 0xFFFF hex.
- The Overload Event Enable Register is set to 0xFF hex.
- The Mask Test Event Enable Register is set to 0xFF hex.

You can change the default conditions using the *PSC command with a parameter of 1 (one). When set to 1, the Standard Event Status Enable Register is set 0x00 hex and the Service Request Enable Register is set to 0x00 hex. This prevents the Power On (PON) event from setting the SRQ interrupt when the oscilloscope is ready to receive commands.

Interface Capabilities

The interface capabilities of this oscilloscope, as defined by IEEE 488.1 and IEEE 488.2, are listed in Table 1.

Table 1 Interface Capabilities

Code	Interface Function	Capability
SH1	Source Handshake	Full Capability
AH1	Acceptor Handshake	Full Capability
T5	Talker	Basic Talker/Serial Poll/Talk Only Mode/ Unaddress if Listen Address (MLA)
L4	Listener	Basic Listener/ Unaddresses if Talk Address (MTA)
SR1	Service Request	Full Capability
RL1	Remote Local	Complete Capability
PP0	Parallel Poll	No Capability
DC1	Device Clear	Full Capability
DT1	Device Trigger	Full Capability
C0	Computer	No Capability
E2	Driver Electronics	Tri State (1 MB/SEC MAX)

GPIB Command and Data Concepts

The GPIB interface has two modes of operation: command mode and data mode. The interface is in the command mode when the Attention (ATN) control line is true. The command mode is used to send talk and listen addresses and various interface commands such as group execute trigger (GET).

The interface is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. The device-dependent messages include all of the oscilloscope-specific commands, queries, and responses found in this manual, including oscilloscope status information.

Communicating Over the GPIB Interface

Device addresses are sent by the computer in the command mode to specify who talks and who listens. Because GPIB can address multiple devices through the same interface card, the device address passed with the program message must include the correct interface select code and the correct oscilloscope address.

Device Address = (Interface Select Code * 100) + Oscilloscope Address

See Also

- "Interface Select Code" on page 95
- "Oscilloscope Address" on page 95

Interface Select Code

Each interface card has a unique interface select code. This code is used by the computer to direct commands and communications to the proper interface. The default is typically "7" for the GPIB interface cards.

Oscilloscope Address

Each oscilloscope on the GPIB must have a unique oscilloscope address between decimal 0 and 30. This oscilloscope address is used by the computer to direct commands and communications to the proper oscilloscope on an interface. The default is typically "7" for this oscilloscope. You can change the oscilloscope address in the Utilities, Remote Interface dialog box.

NOTE

Do Not Use Address 21 for an Oscilloscope Address

Address 21 is usually reserved for the Computer interface Talk/Listen address, and should not be used as an oscilloscope address.

Communicating Over the LAN Interface

The device address used to send commands and receive data using the LAN interface is located in the Remote Setup dialog box (accessed by choosing **Utilities > Remote Setup...** from the Infiniium oscilloscope application's main menu) as shown below.

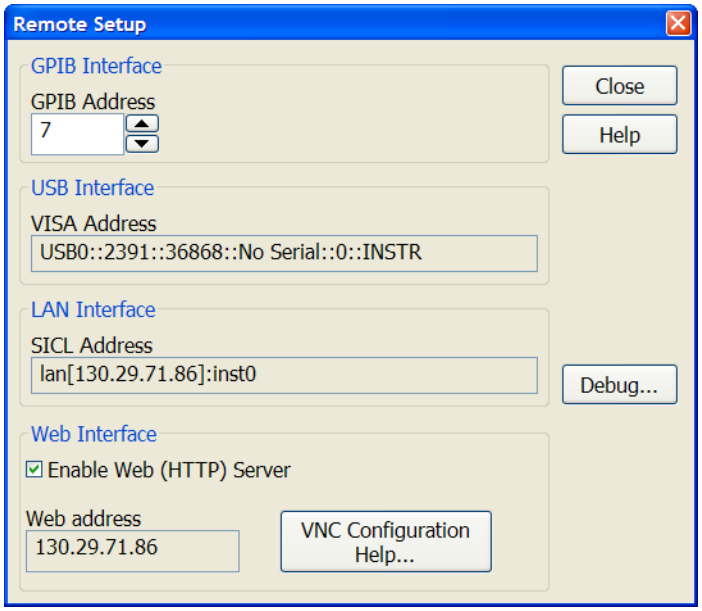


Figure 1 Remote Setup Dialog Box

The following C example program shows how to communicate with the oscilloscope using the LAN interface and the Agilent Standard Instrument Control Library (SICL).

```
#include <sicl.h>

#define BUFFER_SIZE 1024

main()
{
INST Bus;
int reason;
unsigned long actualcnt;
char buffer[ BUFFER_SIZE ];

   /* Open the LAN interface */
   Bus = iopen( "lan[130.29.71.143]:hpib7,7" );
   if( Bus != 0 ) {
      /* Bus timeout set to 20 seconds */
      itimeout( Bus, 20000 );
```

```
        /* Clear the interface */
        iclear( Bus );
        /* Query and print the oscilloscope's Id */
        iwrite( Bus, "*IDN?", 5, 1, &actualcnt );
        iread( Bus, buffer, BUFFER_SIZE, &reason, &actualcnt );
        buffer[ actualcnt - 1 ] = 0;

        printf( "%s\\n", buffer );
        iclose( Bus );
    }
}
```

Communicating via Telnet and Sockets

- "Telnet" on page 98
- "Sockets" on page 98

Telnet

To open a connection to the oscilloscope via a telnet connection, use the following syntax in a command prompt:

```
telnet Oscilloscope_IP_Address 5024
```

5024 is the port number and the name of the oscilloscope can be used in place of the IP address if desired.

After typing the above command line, press enter and a SCPI command line interface will open. You can then use this as you typically would use a command line.

Sockets

Sockets can be used to connect to your oscilloscope on either a Windows or Unix machine.

The sockets are located on port 5025 on your oscilloscope. Between ports 5024 and 5025, only six socket ports can be opened simultaneously. It is, therefore, important that you use a proper close routine to close the connection to the oscilloscope. If you forget this, the connection will remain open and you may end up exceeding the limit of six socket ports.

Some basic commands used in communicating to your oscilloscope include:

- The receive command is: recv
- The send command is: send

Below is a programming example (for a Windows-based machine) for opening and closing a connection to your oscilloscope via sockets.

```
#include <winsock2.h>

void main ()
{
   WSADATA wsaData;
   SOCKET mysocket = NULL;
   char* ipAddress = "130.29.70.70";
   const int ipPort = 5025;

   //Initialize Winsock
   int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
   if(iResult != NO_ERROR)
   {
      printf("Error at WSAStartup()\\n");
```

```
      return NULL;
   }

   //Create the socket
   mySocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCOP);
   if(mySocket == INVALID_SOCKET)
   {
      printf("Error at socket(): %ld\\n", WSAGetLastError());
      WSACleanup();
      return NULL;
   }

   sockaddr_in clientService;
   clientService.sin_family = AF_INET;
   clientService.sin.addr.s_addr = inet_addr(ipAddress);
   clientService.sin_port = htons(ipPort);

   if(connect(mySocket, (SOCKADDR*) &clientService, sizeof(clientService
)))
   {
      printf("Failed to connect.\\n");
      WSACleanup();
      return NULL;
   }

   //Do some work here

   //Close socket when finished
   closesocket(mySocket);
}
```

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions that are taken when these commands are received by the oscilloscope.

Device Clear

The device clear (DCL) and selected device clear (SDC) commands clear the input buffer and output queue, reset the parser, and clear any pending commands. If either of these commands is sent during a digitize operation, the digitize operation is aborted.

Group Execute Trigger

The group execute trigger (GET) command arms the trigger. This is the same action produced by sending the RUN command.

Interface Clear

The interface clear (IFC) command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system computer.

5 Message Communication and System Functions

Protocols 102

This chapter describes the operation of oscilloscopes that operate in compliance with the IEEE 488.2 (syntax) standard. It is intended to give you enough basic information about the IEEE 488.2 standard to successfully program the oscilloscope. You can find additional detailed information about the IEEE 488.2 standard in ANSI/IEEE Std 488.2-1987, "*IEEE Standard Codes, Formats, Protocols, and Common Commands*."

This oscilloscope series is designed to be compatible with other Agilent Technologies IEEE 488.2 compatible instruments. Oscilloscopes that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (GPIB bus standard); however, IEEE 488.1 compatible oscilloscopes may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the oscilloscope and the computer will communicate. It also defines some common capabilities that are found in all IEEE 488.2 oscilloscopes. This chapter also contains some information about the message communication and system functions not specifically defined by IEEE 488.2.

Protocols

The message exchange protocols of IEEE 488.2 define the overall scheme used by the computer and the oscilloscope to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

- "Functional Elements" on page 102
- "Protocol Overview" on page 102
- "Protocol Operation" on page 103
- "Protocol Exceptions" on page 103
- "Suffix Multiplier" on page 103
- "Suffix Unit" on page 104

Functional Elements

Before proceeding with the description of the protocol, you should understand a few system components, as described here.

Input Buffer The input buffer of the oscilloscope is the memory area where commands and queries are stored prior to being parsed and executed. It allows a computer to send a string of commands, which could take some time to execute, to the oscilloscope, then proceed to talk to another oscilloscope while the first oscilloscope is parsing and executing commands.

Output Queue The output queue of the oscilloscope is the memory area where all output data or response messages are stored until read by the computer.

Parser The oscilloscope's parser is the component that interprets the commands sent to the oscilloscope and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and execution of commands begins when either the oscilloscope recognizes a program message terminator, or the input buffer becomes full. If you want to send a long sequence of commands to be executed, then talk to another oscilloscope while they are executing, you should send all of the commands before sending the program message terminator.

Protocol Overview

The oscilloscope and computer communicate using program messages and response messages. These messages serve as the containers into which sets of program commands or oscilloscope responses are placed.

A program message is sent by the computer to the oscilloscope, and a response message is sent from the oscilloscope to the computer in response to a query message. A query message is defined as being a program message that contains one or more queries. The oscilloscope will

only talk when it has received a valid query message, and therefore has something to say. The computer should only attempt to read a response after sending a complete query message, but before sending another program message.

NOTE

Remember this Rule of Oscilloscope Communication

The basic rule to remember is that the oscilloscope will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When you turn the oscilloscope on, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

The oscilloscope and the computer communicate by exchanging complete program messages and response messages. This means that the computer should always terminate a program message before attempting to read a response. The oscilloscope will terminate response messages except during a hard copy output.

After you send a query message, the next message should be the response message. The computer should always read the complete response message associated with a query message before sending another program message to the same oscilloscope.

The oscilloscope allows the computer to send multiple queries in one query message. This is called sending a "compound query". Multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner.

Suffix Multiplier

The suffix multipliers that the oscilloscope will accept are shown in Table 2.

Table 2 <suffix mult>

Value	Mnemonic	Value	Mnemonic
1E18	EX	1E-3	M
1E15	PE	1E-6	U
1E12	T	1E-9	N
1E9	G	1E-12	P
1E6	MA	1E-15	F
1E3	K	1E-18	A

Suffix Unit

The suffix units that the oscilloscope will accept are shown in Table 3.

Table 3 <suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second

6 Status Reporting

Status Reporting Data Structures 108
Status Byte Register 110
Service Request Enable Register 112
Message Event Register 113
Trigger Event Register 114
Standard Event Status Register 115
Standard Event Status Enable Register 116
Operation Status Register 117
Operation Status Enable Register 118
Mask Test Event Register 119
Mask Test Event Enable Register 120
Acquisition Done Event Register 121
Process Done Event Register 122
Trigger Armed Event Register 123
Auto Trigger Event Register 124
Error Queue 1060
Output Queue 126
Message Queue 127
Clearing Registers and Queues 128

An overview of the oscilloscope's status reporting structure is shown in Figure 2. The status reporting structure shows you how to monitor specific events in the oscilloscope. Monitoring these events lets you determine 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 oscilloscope.
- To generate a service request (SRQ) interrupt to an external computer, enable at least one bit in the Status Byte 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. IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting. There are also oscilloscope-defined structures and bits.

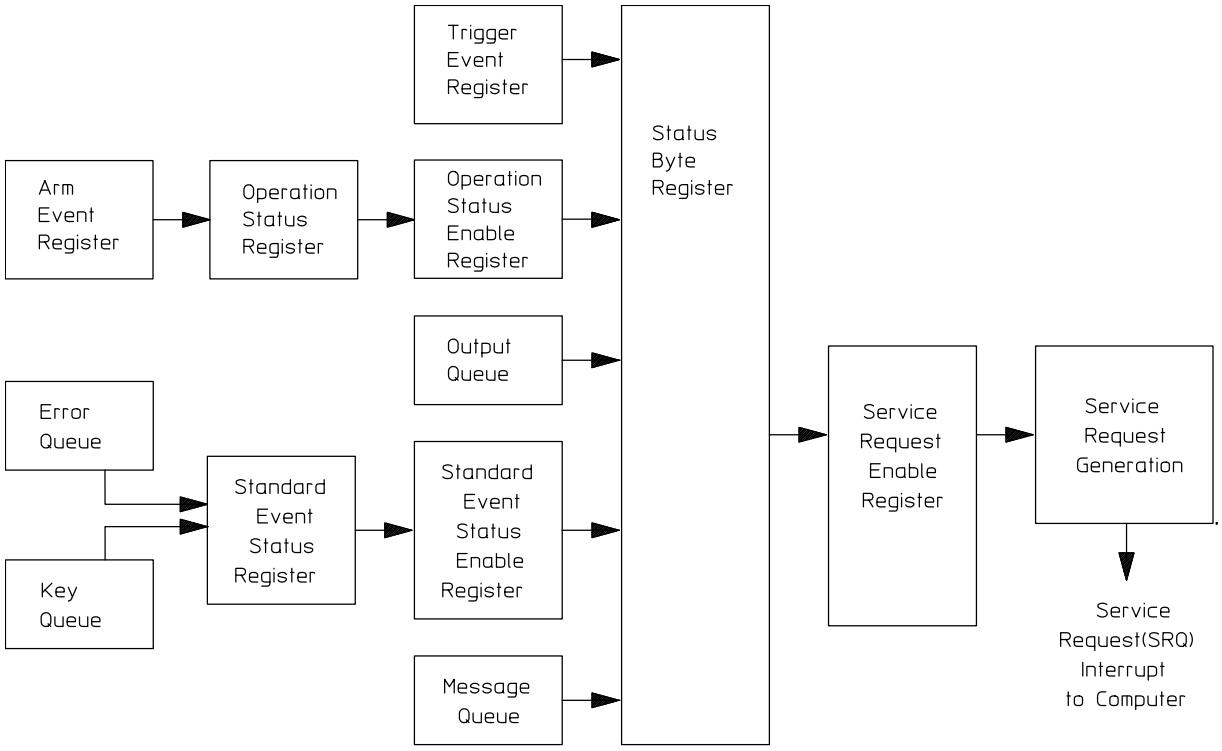


Figure 2 Status Reporting Overview Block Diagram

The status reporting structure consists of the registers shown here.

Table 4 lists the bit definitions for each bit in the status reporting data structure.

Table 4 Status Reporting Bit Definition

Bit	Description	Definition
PON	Power On	Indicates power is turned on.
URQ	User Request	Not Used. Permanently set to zero.
CME	Command Error	Indicates if the parser detected an error.
EXE	Execution Error	Indicates if a parameter was out of range or was inconsistent with the current settings.

Table 4 Status Reporting Bit Definition (continued)

Bit	Description	Definition
DDE	Device Dependent Error	Indicates if the device was unable to complete an operation for device-dependent reasons.
QYE	Query Error	Indicates if the protocol for queries has been violated.
RQL	Request Control	Indicates if the device is requesting control.
OPC	Operation Complete	Indicates if the device has completed all pending operations.
OPER	Operation Status Register	Indicates if any of the enabled conditions in the Operation Status Register have occurred.
RQS	Request Service	Indicates that the device is requesting service.
MSS	Master Summary Status	Indicates if a device has a reason for requesting service.
ESB	Event Status Bit	Indicates if any of the enabled conditions in the Standard Event Status Register have occurred.
MAV	Message Available	Indicates if there is a response in the output queue.
MSG	Message	Indicates if an advisory has been displayed.
USR	User Event Register	Indicates if any of the enabled conditions have occurred in the User Event Register.
TRG	Trigger	Indicates if a trigger has been received.
WAIT TRIG	Wait for Trigger	Indicates the oscilloscope is armed and ready for trigger.

Status Reporting Data Structures

The different status reporting data structures, descriptions, and interactions are shown in Figure 3. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, you must enable the corresponding bits. 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 the computer, you must enable at least one bit in the Status Byte Register. 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.

For more information about common commands, see the "Common Commands" chapter.

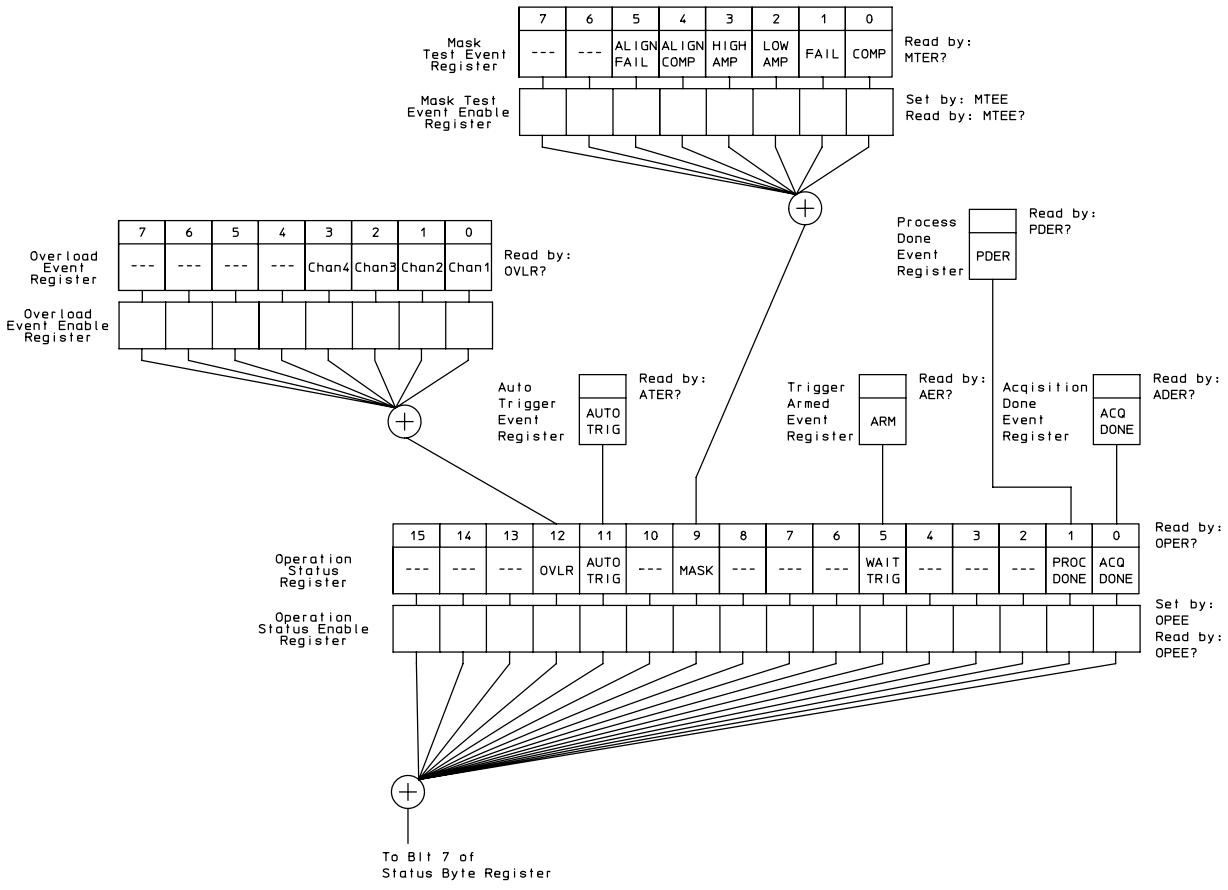


Figure 3 Status Reporting Data Structures

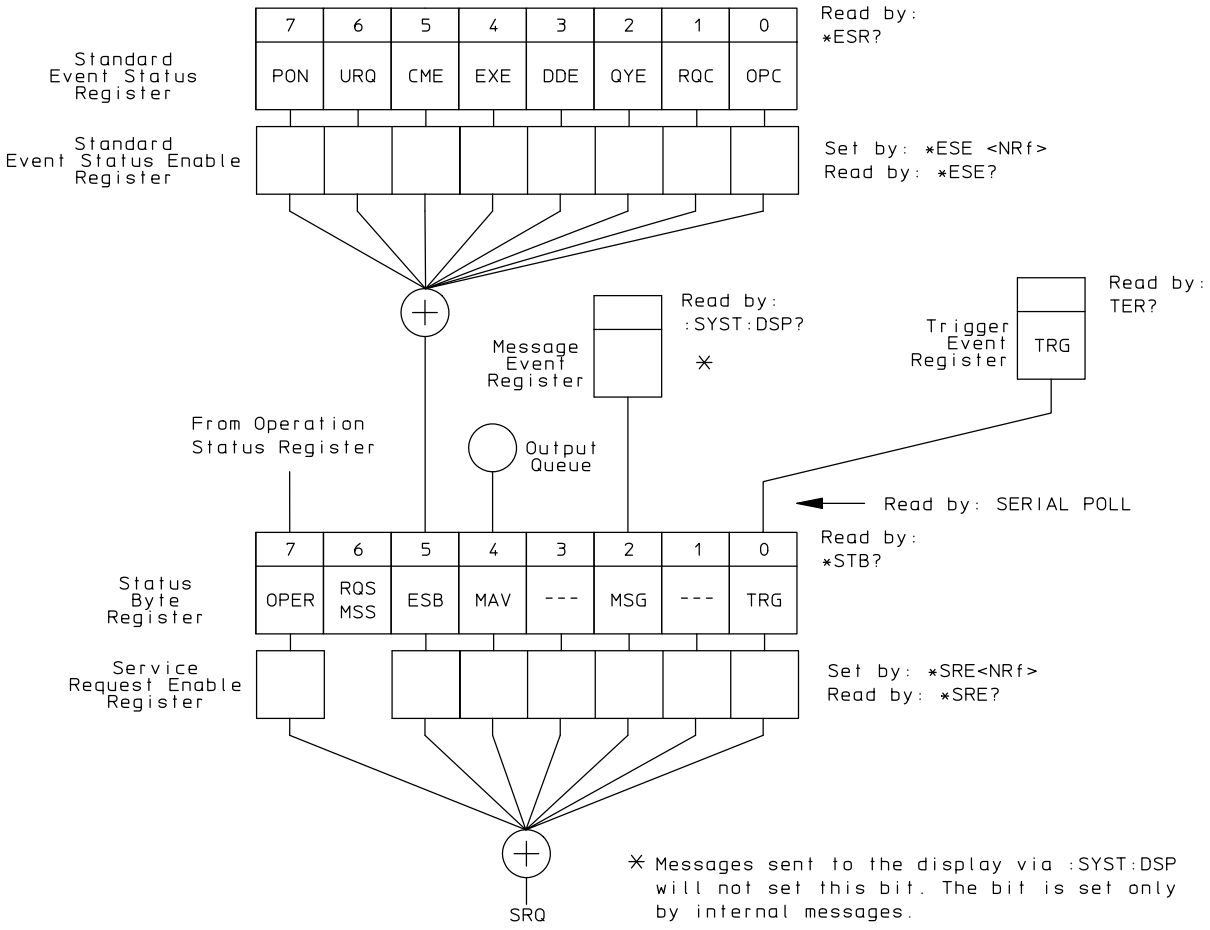


Figure 4 Status Reporting Data Structures (Continued)

Status Byte Register

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.

You can read the Status Byte Register using either the *STB? common command query 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? query reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any effect 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.

The only other bit in the Status Byte Register affected by the *STB? query is 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? query.

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

Example 1 This example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register when none of the register's summary bits are enabled to generate an SRQ interrupt.

```
Dim varStbValue As Variant
myScope.WriteString ":SYSTEM:HEADER OFF;*STB?"   'Turn headers off
varStbValue = myScope.ReadNumber
Debug.Print "Status Byte Register, Read: 0x" + Hex(varStbValue)
```

The next program prints "0x84" and clears bit 6 (RQS) of the Status Byte Register. The difference in the decimal 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 2 The following example uses the resource session object's ReadSTB method to read the contents of the oscilloscope's Status Byte Register.

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

NOTE

Use Serial Polling to Read the 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

Setting the Service Request Enable Register bits enables 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. Bit 6 always returns 0. Refer to the Status Reporting Data Structures shown in Figure 3.

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).

Message Event Register

This register sets the MSG bit in the status byte register when an internally generated message is written to the advisory line on the oscilloscope. The message is read using the :SYSTEM:DSP? query. Note that messages written to the advisory line on the oscilloscope using the :SYSTEM:DSP command does not set the MSG status bit.

Trigger Event Register

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

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

If you are using the Service Request to interrupt a computer operation when the trigger bit is set, you must clear the event register after each time it is set.

Standard Event Status Register

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

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

When one of these events occurs, the corresponding bit is set in the register. If the corresponding bit is also enabled in the Standard Event Status Enable Register, a summary bit (ESB) in the Status Byte Register is set.

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 bits 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 ":SYSTEM:HEADER OFF"    'Turn headers off
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
Debug.print "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

For any of the Standard Event Status Register bits to generate a summary bit, you must first enable the bit. Use the *ESE (Event Status Enable) common command to set the corresponding bit in the Standard Event Status Enable Register. 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), a service request interrupt (SRQ) 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 in the Status Byte Register.

Operation Status Register

This register hosts the following bits:

- ACQ DONE bit 0
- PROC DONE bit 1
- WAIT TRIG bit 5
- MASK bit 9
- AUTO TRIG bit 11
- OVLR bit 12

The ACQ DONE done bit is set by the Acquisition Done Event Register.

The PROC DONE bit is set by the Process Done Event Register and indicates that all functions and all math processes are done.

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates the trigger is armed.

The MASK bit is set whenever at least one of the Mask Test Event Register bits is enabled.

The AUTO TRIG bit is set by the Auto Trigger Event Register.

The OVLR bit is set whenever at least one of the Overload Event Register bits is enabled.

If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Register is read and cleared with the OPER? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Operation Status Enable Register

For any of the Operation Status Register bits to generate a summary bit, you must first enable the bit. Use the OPEE (Operation Event Status Enable) command to set the corresponding bit in the Operation Status Enable Register. Set bits are read with the OPEE? query.

Example Suppose your application requires an interrupt whenever any event occurs in the mask test register. The error status bit in the Operation Status Register is bit 9. Therefore, you can enable this bit to generate the summary bit by sending:

```
myScope.WriteString ":OPEE " + CStr(CInt("&H200"))
```

Whenever an error occurs, the oscilloscope sets this bit in the Mask Test Event Register. Because this bit is enabled, a summary bit is generated to set bit 9 (OPER) in the Operation Status Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

NOTE

Disabled Operation Status Register Bits Respond, but Do Not Generate a Summary Bit

Operation 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 in the Status Byte Register.

Mask Test Event Register

This register hosts the following bits:

- Mask Test Complete bit (bit 0)
- Mask Test Fail bit (bit 1)
- Mask Low Amplitude bit (bit 2)
- Mask High Amplitude bit (bit 3)
- Mask Align Complete bit (bit 4)
- Mask Align Fail bit (bit 5)

The Mask Test Complete bit is set whenever the mask test is complete.

The Mask Test Fail bit is set whenever the mask test failed.

The Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.

The Mask High Amplitude bit is set whenever the signal is above the mask amplitude.

The Mask Align Complete bit is set whenever the mask align is complete.

The Mask Align Fail bit is set whenever the mask align failed.

If any of these bits are set, the MASK bit (bit 9) of the Operation Status Register is set. The Mask Test Event Register is read and cleared with the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

Mask Test Event Enable Register

For any of the Mask Test Event Register bits to generate a summary bit, you must first enable the bit. Use the MTEE (Mask Test Event Enable) command to set the corresponding bit in the Mask Test Event Enable Register. Set bits are read with the MTEE? query.

Example Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
myScope.WriteString ":MTEE " + CStr(CInt("&H2"))
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

NOTE

Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit

Mask Test Event 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 in the Operation Status Register.

Acquisition Done Event Register

The Acquisition Done Event Register (ACQ DONE) sets bit 0 (ACQ DONE bit) in the Operation Status Register when the oscilloscope acquisition is completed.

The ACQ DONE event register stays set until it is cleared by reading the register by a ADER? query. If your application needs to detect multiple acquisitions, the ACQ DONE event register must be cleared after each acquisition.

Process Done Event Register

The Process Done Event Register (PDER) sets bit 1 (PROC DONE) of the Operation Status Register when all functions and all math operations are completed. The PDER bit stays set until cleared by a PDER? query.

Trigger Armed Event Register

The Trigger Armed Event Register (TDER) sets bit 5 (WAIT TRIG) in the Operation Status Register when the oscilloscope becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

Auto Trigger Event Register

The Auto Trigger Event Register (AUTO TRIG) sets bit 11 (AUTO TRIG) in the Operation Status Register when an auto trigger event occurs. The AUTO TRIG register stays set until it is cleared by reading the register with the ATER? query. If the application needs to detect multiple auto trigger events, the AUT TRIG register must be cleared after each one.

Error Queue

As errors are detected, they are placed in an error queue. This queue is a first-in, first-out queue. If the error queue overflows, the last error in the queue is replaced with error -350, "Queue overflow." Any time the queue overflows, the oldest 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 any of these events occur:

- When the oscilloscope is powered up.
- When the oscilloscope receives the *CLS common command.
- When the last item is read from the error queue.

For more information on reading the error queue, refer to the :SYSTEM:ERROR? query in the System Commands chapter. For a complete list of error messages, refer to the chapter, "Error Messages."

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain oscilloscope 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 queue is read with the :SYSTEM:DSP? query. Note that messages sent with the :SYSTEM:DSP command do not set the MSG status bit in the Status Byte Register.

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately following a program message terminator, the output queue is also cleared.

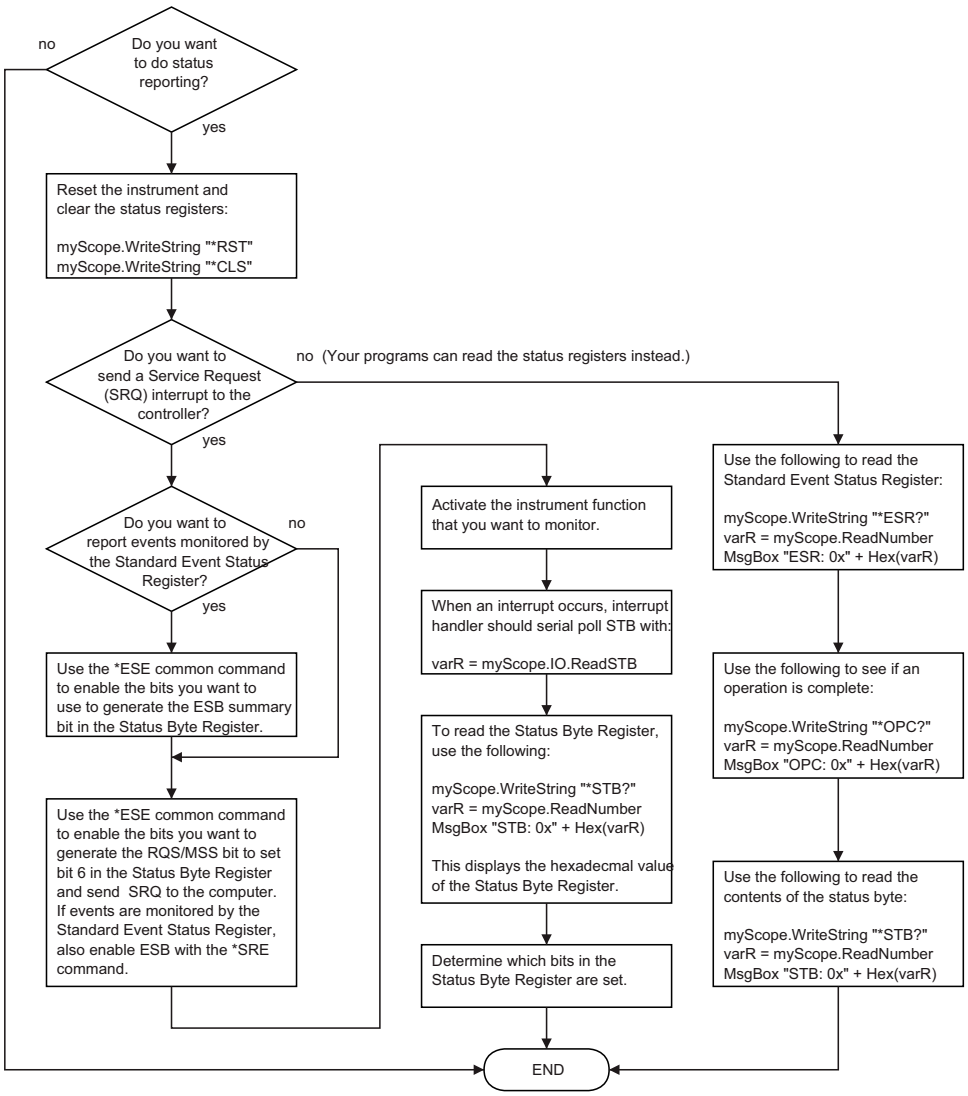


Figure 5 Status Reporting Decision Chart

7 Remote Acquisition Synchronization

Programming Flow 130
Setting Up the Oscilloscope 131
Acquiring a Waveform 132
Retrieving Results 133
Acquisition Synchronization 134
Single Shot Device Under Test (DUT) 136
Averaging Acquisition Synchronization 138

When remotely controlling an oscilloscope with SCPI commands, it is often necessary to know when the oscilloscope has finished the previous operation and is ready for the next SCPI command. The most common example is when an acquisition is started using the :DIG, :RUN, or :SINGLE commands. Before a measurement result can be queried, the acquisition must complete. Too often, fixed delays are used to accomplish this wait, but fixed delays often use excessive time or the time may not be long enough. A better solution is to use synchronous commands and status to know when the oscilloscope is ready for the next request.

Programming Flow

Most remote programming follows these three general steps:

1. Setup the oscilloscope and device under test
2. Acquire a waveform
3. Retrieve results

Setting Up the Oscilloscope

Before making changes to the oscilloscope setup, it is best to make sure it is stopped using the :STOP command followed by the *OPC? command.

NOTE It is not necessary to use the *OPC? command, hard coded waits, or status checking when setting up the oscilloscope.

After the oscilloscope is configured, it is ready for an acquisition.

Acquiring a Waveform

When acquiring a waveform, there are two possible methods used to wait for the acquisition to complete. These methods are blocking and polling. The table below details when each method should be chosen and why.

	Blocking Wait	**Polling Wait**
Use When	You know the oscilloscope will trigger based on the oscilloscope setup and device under test.	You know the oscilloscope may or may not trigger based on the oscilloscope setup and device under test.
Advantages	• No need for polling • Fast method	• Remote interface will not timeout • No need for device clear if no trigger
Disadvantages	• Remote interface may timeout • Device clear only way to get control of oscilloscope if there is no trigger	• Slower method • Required polling loop • Required known maximum wait time

Retrieving Results

Once the acquisition is complete, it is safe to retrieve measurements and statistics.

Acquisition Synchronization

- "Blocking Synchronization" on page 134
- "Polling Synchronization With Timeout" on page 134

Blocking Synchronization

Use the :DIGitize command to start the acquisition. This blocks subsequent queries until the acquisition and processing is complete.

Example

```
// Setup
:TRIGGER:MODE EDGE
:TIMEBASE:SCALE 5e-9

//Acquire
:DIG

//Get results
:MEASURE:RISETIME?
```

Polling Synchronization With Timeout

This example requires a timeout value so the operation can abort if an acquisition does not occur within the timeout period.

Example

```
TIMEOUT = 1000ms
currentTime = 0ms

// Setup
:STOP; *OPC?   // if not stopped
:ADER?         // clear ADER event

// Acquire
:SINGLE

while(currentTime <= TIMEOUT)
{
   if (:ADER? == 1)
   {
      break;
   }
   else
   {
      // Use small wait to prevent excessive
      // queries to the oscilloscope
      wait (100ms)
      currentTime += 100ms
    }
}

//Get results
if (currentTime < TIMEOUT)
{
```

```
    :MEASURE:RISETIME?
}
```

Single Shot Device Under Test (DUT)

The examples in the previous section (Acquisition Synchronization) assumed the DUT is continually running and, therefore, the oscilloscope will have more than one opportunity to trigger. With a single shot DUT, there is only one opportunity for the oscilloscope to trigger so it is necessary for the oscilloscope to be armed and ready before the DUT is enabled.

NOTE The blocking :DIGitize command cannot be used for a single shot DUT because once the :DIGitize command is issued, the oscilloscope is blocked from any further commands until the acquisition is complete.

This example is the same as the previous example with the addition of checking for the armed event status.

Example

```
TIMEOUT = 1000ms
currentTime = 0ms

// Setup
:STOP; *OPC?   // if not stopped
:ADER?         // clear ADER event

// Acquire
:SINGLE

while(AER? == 0)
{
   wait(100ms)
}

//oscilloscope is armed and ready, enable DUT here

while(currentTime <= TIMEOUT)
{
   if (:ADER? == 1)
   {
      break;
   }
   else
   {
      // Use small wait to prevent excessive
      // queries to the oscilloscope
      wait (100ms)
      currentTime += 100ms
    }
}

//Get results
if (currentTime < TIMEOUT)
{
```

```
    :MEASURE:RISETIME?
}
```

Averaging Acquisition Synchronization

When averaging, it is necessary to know when the average count has been reached. Since an ADER/PDER event occurs for every acquisition in the average count, these commands cannot be used. The :SINGle command does not average.

If it is known that a trigger will occur, a :DIG will acquire the complete number of averages, but if the number of averages is large, it may cause a timeout on the connection.

The example below acquires the desired number of averages and then stops running.

Example

```
AVERAGE_COUNT = 256

:STOP;*OPC?
:TER?
:ACQ:AVERage:COUNt AVERAGE_COUNT
:ACQ:AVERage ON
:RUN

//Assume the oscilloscope will trigger, if not put a check here

while (:WAV:COUNT? < AVERAGE_COUNT)
{
   wait(100ms)
}

:STOP;*OPC?

// Get results
```

8 Programming Conventions

Truncation Rule 140
The Command Tree 141
Infinity Representation 144
Sequential and Overlapped Commands 145
Response Generation 146
EOI 147

This chapter describes conventions used to program the Infiniium-Series Oscilloscopes, and conventions used throughout this manual. A description of the command tree and command tree traversal is also included.

Truncation Rule

The truncation rule is used to produce the short form (abbreviated spelling) for the mnemonics used in the programming headers and parameter arguments.

NOTE

Command Truncation Rule

The mnemonic is the first four characters of the keyword, unless the fourth character is a vowel. Then the mnemonic is the first three characters of the keyword. If the length of the keyword is four characters or less, this rule does not apply, and the short form is the same as the long form.

This document's command descriptions shows how the truncation rule is applied to commands.

Table 5 Mnemonic Truncation

Long Form	Short Form	How the Rule is Applied
RANGe	RANG	Short form is the first four characters of the keyword.
PATTern	PATT	Short form is the first four characters of the keyword.
DISK	DISK	Short form is the same as the long form.
DELay	DEL	Fourth character is a vowel; short form is the first three characters.

The Command Tree

The command tree in this document's table of contents shows all of the commands in the Infiniium-Series Oscilloscopes and the relationship of the commands to each other. The IEEE 488.2 common commands are not 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 oscilloscope, the parser is set to the "root" of the command tree.

- "Command Types" on page 141
- "Tree Traversal Rules" on page 141
- "Tree Traversal Examples" on page 142

Command Types

The commands in this oscilloscope can be viewed as three types: common commands, root level commands, and subsystem commands.

- Common commands are commands defined by IEEE 488.2 and control some functions that are common to all IEEE 488.2 instruments. These commands are independent of the tree and do not affect the position of the parser within the tree. *RST is an example of a common command.
- Root level commands control many of the basic functions of the oscilloscope. These commands reside at the root of the command tree. They can always be parsed if they occur at the beginning of a program message or are preceded by a colon. Unlike common commands, root level commands place the parser back at the root of the command tree. AUTOSCALE is an example of a root level command.
- Subsystem commands are grouped together under a common node of the command tree, such as the TIMEBASE commands. You may select only one subsystem at a given time. When you turn on the oscilloscope initially, the command parser is set to the root of the command tree and no subsystem is selected.

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree would be :TIMEBASE:RANGE. This is referred to as a compound header. A compound header is a header made up of two or more mnemonics separated by colons. The compound header contains no spaces. The following rules apply to traversing the tree.

NOTE

Tree Traversal Rules

A leading colon or a program message terminator (<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 places the oscilloscope in that subsystem until a leading colon or a program message terminator is found.

In the command tree, use the last mnemonic in the compound header as a reference point (for example, RANGE). Then find the last colon above that mnemonic (TIMEBASE:). That is the point where the parser resides. You can send any command below this point within the current program message without sending the mnemonics which appear above them (for example, REFERENCE).

Tree Traversal Examples

The WriteString() methods in the following examples are written using Visual Basic for Application (VBA) with the VISA COM library.

Example 1 Consider the following command:

```
myScope.WriteString ":CHANNEL1:RANGE 0.5;OFFSET 0"
```

The colon between CHANNEL1 and RANGE is necessary because :CHANNEL1:RANGE is a compound command. The semicolon between the RANGE command and the OFFSET command is required to separate the two commands or operations. The OFFSET command does not need :CHANNEL1 preceding it because the :CHANNEL1:RANGE command sets the parser to the CHANNEL1 node in the tree.

Example 2 Consider the following commands:

```
myScope.WriteString ":TIMEBASE:REFERENCE CENTER;POSITION 0.00001"
```

or

```
myScope.WriteString ":TIMEBASE:REFERENCE CENTER"
myScope.WriteString ":TIMEBASE:POSITION 0.00001"
```

In the first line of example 2, the "subsystem selector" is implied for the POSITION command in the compound command.

A second way to send these commands is shown in the second part of the example. Because the program message terminator places the parser back at the root of the command tree, you must reselect TIMEBASE to re-enter the TIMEBASE node before sending the POSITION command.

Example 3 Consider the following command:

```
myScope.WriteString ":TIMEBASE:REFERENCE CENTER;:CHANNEL1:OFFSET 0"
```

In this example, the leading colon before CHANNEL1 tells the parser to go back to the root of the command tree. The parser can then recognize the :CHANNEL1:OFFSET command and enter the correct node.

Infinity Representation

The representation for infinity for this oscilloscope is 9.99999E+37. This is also the value returned when a measurement cannot be made.

Sequential and Overlapped Commands

IEEE 488.2 makes a 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.

Response Generation

As defined by IEEE 488.2, query responses may be buffered for these reasons:

- When the query is parsed by the oscilloscope.
- When the computer addresses the oscilloscope to talk so that it may read the response.

This oscilloscope buffers responses to a query when the query is parsed.

EOI

The EOI bus control line follows the IEEE 488.2 standard without exception.

9 Acquire Commands

:ACQuire:AVERage 150
:ACQuire[:AVERage]:COUNt 151
:ACQuire:BANDwidth 152
:ACQuire:COMPlete 154
:ACQuire:COMPlete:STATe 156
:ACQuire:HRESolution 157
:ACQuire:INTerpolate 158
:ACQuire:MODE 159
:ACQuire:POINts[:ANALog] 161
:ACQuire:POINts:AUTO 165
:ACQuire:POINts:DIGital? 166
:ACQuire:REDGe 167
:ACQuire:RESPonse 168
:ACQuire:SEGMented:COUNt 169
:ACQuire:SEGMented:INDex 170
:ACQuire:SEGMented:TTAGs 171
:ACQuire:SRATe[:ANALog] 172
:ACQuire:SRATe[:ANALog]:AUTO 174
:ACQuire:SRATe:DIGital 175
:ACQuire:SRATe:DIGital:AUTO 176

The ACQuire subsystem commands set up conditions for executing a :DIGitize root level command to acquire waveform data. The commands in this subsystem select the type of data, the number of averages, and the number of data points.

:ACQuire:AVERage

Command `:ACQuire:AVERage {{ON|1} | {OFF|0}}`

The :ACQuire:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :ACQuire:AVERage:COUNt command described next.

Averaging is not available in PDETect, PDETect, SEGMented, SEGPdetect, or SEGHres modes.

The :MTESt:AVERage command performs the same function as this command.

Example This example turns averaging on.

```
myScope.WriteString ":ACQuire:AVERage ON"
```

Query `:ACQuire:AVERage?`

The :ACQuire:AVERage? query returns the current setting for averaging.

Returned Format `[:ACQuire:AVERave] {1|0}<NL>`

Example This example places the current settings for averaging into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":ACQuire:AVERage?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":ACQuire:MODE" on page 159

:ACQuire[:AVERage]:COUNt

Command `:ACQuire[:AVERage]:COUNt <count_value>`

The :ACQuire[:AVERage]:COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :ACQuire[:AVERage]:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :MTESt:AVERage:COUNt command performs the same function as this command.

<count_value> An integer, 2 to 65,534, specifying the number of data values to be averaged.

Example This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :ACQuire:COMPlete command.

```
myScope.WriteString ":ACQuire:COUNt 16"
```

Query `:ACQuire[:AVERage]:COUNt?`

The :ACQuire[:AVERage]:COUNt? query returns the currently selected count value.

Returned Format `[:ACQuire[:AVERage]:COUNt] <value><NL>`

<value> An integer, 2 to 65,534, specifying the number of data values to be averaged.

Example This example checks the currently selected count value and places that value in the string variable, strResult. The program then prints the contents of the variable to the computer's screen.

```
Dim strResult As String
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:AVERage:COUNt?"
strResult = myScope.ReadString
Debug.Print strResult
```

:ACQuire:BANDwidth

(Enhanced bandwidth or noise reduction option only)

Command

NOTE This command is only available with Enhanced Bandwidth or Noise Reduction options.

```
:ACQuire:BANDwidth {AUTO | MAX | <bandwidth>}
```

The :ACQuire:BANDwidth command changes the bandwidth frequency control for the acquisition system.

- AUTO – The bandwidth is automatically selected based on the sample rate setting in order to make a good a trade-off between bandwidth, noise, and aliasing.
- MAX – Sets the oscilloscope to the hardware bandwidth limit and disables the bandwidth filter.
- <bandwidth> – a real number representing the bandwidth of the bandwidth filter whose range of values depends on the model number of your oscilloscope.

Model	Bandwidth Filter Values
DSOX/DSAX96204Q	The maximum bandwidth down to 1 GHz in 1 GHz increments.
DSOX/DSAX95004Q	
DSOX/DSAX93304Q	
DSOX/DSAX92504Q	
DSOX/DSAX92004Q	
DSOX/DSAX93204A	The maximum bandwidth down to 1 GHz in 1 GHz increments.
DSOX/DSAX92804A	
DSOX/DSAX92504A	
DSOX/DSAX92004A	
DSOX/DSAX91604A	
DSO/DSA91304A	13E09, 12E09, 10E09, 8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09
DSO/DSA91204A	12E09, 10E09, 8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09
DSO/DSA90804A	8E09, 6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09
DSO/DSA90604A	6E09, 4E09, 3E09, 2.5E09, 2E09, 1E09
DSO/DSA90404A	4E09, 3E09, 2.5E09, 2E09, 1E09
DSO/DSA90254A	2.5E09, 2E09, 1E09

Query `:ACQuire:BANDwidth?`

The :ACQuire:BANDwidth? query returns the bandwidth setting of the bandwidth control.

Returned Format `[:ACQuire:BANDwidth] <bandwidth><NL>`

:ACQuire:COMPlete

Command `:ACQuire:COMPlete <percent>`

The :ACQuire:COMPlete command specifies how many of the data point storage bins (time buckets) in the waveform record must contain a waveform sample before a measurement will be made. For example, if the command :ACQuire:COMPlete 60 has been sent, 60% of the storage bins in the waveform record must contain a waveform data sample before a measurement is made.

- If :ACQuire:AVERage is set to OFF, the oscilloscope only needs one value per time bucket for that time bucket to be considered full.
- If :ACQuire:AVERage is set to ON, each time bucket must have *n* hits for it to be considered full, where *n* is the value set by :ACQuire:AVERage:COUNt.

Due to the nature of real time acquisition, 100% of the waveform record bins are filled after each trigger event, and all of the previous data in the record is replaced by new data when :ACQuire:AVERage is off. Hence, the complete mode really has no effect, and the behavior of the oscilloscope is the same as when the completion criteria is set to 100% (this is the same as in PDETect mode). When :ACQuire:AVERage is on, all of the previous data in the record is replaced by new data.

The range of the :ACQuire:COMPlete command is 0 to 100 and indicates the percentage of time buckets that must be full before the acquisition is considered complete. If the complete value is set to 100%, all time buckets must contain data for the acquisition to be considered complete. If the complete value is set to 0, then one acquisition cycle will take place. Completion is set by default setup or *RST to 90%. Autoscale changes it to 100%.

<percent> An integer, 0 to 100, representing the percentage of storage bins (time buckets) that must be full before an acquisition is considered complete.

Example This example sets the completion criteria for the next acquisition to 90%.

```
myScope.WriteString ":ACQUIRE:COMPLETE 90"
```

Query `:ACQuire:COMPlete?`

The :ACQuire:COMPlete? query returns the completion criteria.

Returned Format `[:ACQuire:COMPlete] <percent><NL>`

<percent> An integer, 0 to 100, representing the percentage of time buckets that must be full before an acquisition is considered complete.

Example This example reads the completion criteria and places the result in the variable, varPercent. Then, it prints the content of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:COMPLETE?"
varPercent = myScope.ReadNumber
Debug.Print FormatNumber(varPercent, 0)
```

:ACQuire:COMPlete:STATe

Command `:ACQuire:COMPlete:STATe {{ON | 1} | {OFF | 0}}`

The :ACQuire:COMPlete:STATe command specifies the state of the :ACQuire:COMPlete mode. This mode is used to make a tradeoff between how often equivalent time waveforms are measured, and how much new data is included in the waveform record when a measurement is made. This command has no effect when the oscilloscope is in real time mode because the entire record is filled on every trigger. However, in equivalent time mode, as few as 0 new data points will be placed in the waveform record as the result of any given trigger event. You set the acquire mode of the oscilloscope by using the :ACQuire:MODE command.

NOTE The :ACQuire:COMPlete:STATe command is used only when the oscilloscope is operating in equivalent time mode and a digitize operation is not being performed. The :DIGitize command temporarily overrides the setting of this mode and forces it to ON.

ON Turns the COMPlete mode on. Then you can specify the completion percent.

OFF When off, the oscilloscope makes measurements on waveforms after each acquisition cycle, regardless of how complete they are. The waveform record is not cleared after each measurement. Instead, previous data points will be replaced by new samples as they are acquired.

Query `:ACQuire:COMPlete:STATe?`

The :ACQuire:COMPlete:STATe? query returns the state of the :ACQuire:COMPlete:STATe mode.

:ACQuire:HRESolution

Command `:ACQuire:HRESolution {AUTO | BITS9 | BITS10 | BITS11 | BITS12}`

When :ACQuire:MODE is set to HRESolution or SEGHres, the :ACQuire:HRESolution command sets the desired minimum bit resolution.

- AUTO — the number of bits of vertical resolution is determined by the sampling rate, which can be controlled manually by the :ACQuire:SRATe command or automatically when adjusting :TIMebase:SCALe (or :TIMebase:RANGe).
- BITS9, BITS10, BITS11, BITS12 — selects the desired minimum number of bits of vertical resolution (which can affect the sampling rate).

Example This example sets the bit resolution setting to a minimum of 11 bits.

```
myScope.WriteString ":ACQuire:HRESolution BITS11"
```

Query `:ACQuire:HRESolution?`

The :ACQuire:HRESolution? query returns the bit resolution setting.

Returned Format `[:ACQuire:HRESolution] {AUTO | BITS9 | BITS10 | BITS11 | BITS12}<NL>`

Example This example places the current bit resolution setting in the string variable, strBitRes, then prints the contents of the variable to the computer's screen.

```
Dim strBitRes As String   ' Dimension variable.
myScope.WriteString ":ACQuire:HRESolution?"
strBitRes = myScope.ReadString
Debug.Print strBitRes
```

See Also

- ":ACQuire:MODE" on page 159
- ":ACQuire:SRATe[:ANALog]" on page 172
- ":TIMebase:SCALe" on page 876
- ":TIMebase:RANGe" on page 873

:ACQuire:INTerpolate

Command `:ACQuire:INTerpolate {{ON | 1} | {OFF | 0} | INT1 | INT2 | INT4 | INT8 | INT16}`

The :ACQuire:INTerpolate command turns the sin(x)/x interpolation filter on or off when the oscilloscope is in one of the real time sampling modes. You can also specify the 1, 2, 4, 8, or 16 point Sin(x)/x interpolation ratios using INT1, INT2, INT4, INT8, or INT16. When ON, the number of interpolation points is automatically determined.

Query `:ACQuire:INTerpolate?`

The :ACQuire:INTerpolate? query returns the current state of the sin(x)/x interpolation filter control.

Returned Format `[:ACQuire:INTerpolate] {1 | 0 | INT1 | INT2 | INT4 | INT8 | INT16}<NL>`

:ACQuire:MODE

Command `:ACQuire:MODE {ETIMe | RTIMe | PDETect | HRESolution | SEGMented | SEGPdetect | SEGHres}`

The :ACQuire:MODE command sets the sampling/acquisition mode of the oscilloscope.

ETIMe In Equivalent Time mode, available on 90000A Series oscilloscopes, the data record is acquired over multiple trigger events.

RTIMe In Real Time Normal mode, the complete data record is acquired on a single trigger event.

PDETect In Real Time Peak Detect mode, the oscilloscope acquires all of the waveform data points during one trigger event. The data is acquired at the fastest sample rate of the oscilloscope regardless of the horizontal scale setting. The sampling rate control then shows the storage rate into the channel memory rather than the sampling rate. The storage rate determines the number of data points per data region. From each data region, four sample points are chosen to be displayed for each time column. The four sample points chosen from each data region are:

- the minimum voltage value sample
- the maximum voltage value sample
- a randomly selected sample
- an equally spaced sample

The number of samples per data region is calculated using the equation:

$$\text{Number of Samples} = \frac{\text{Sampling Rate}}{\text{Storage Rate}}$$

The remainder of the samples are not used for display purposes.

HRESolution In Real Time High Resolution mode, the oscilloscope acquires all the waveform data points during one trigger event and averages them thus reducing noise and improving voltage resolution. The data is acquired at the fastest sample rate of the oscilloscope regardless of the horizontal scale setting. The sampling rate control then shows the storage rate into the channel memory rather than the sampling rate. The number of samples that are averaged together per data region is calculated using the equation

$$\text{Number of Samples} = \frac{\text{Sampling Rate}}{\text{Storage Rate}}$$

This number determines how many samples are averaged together to form the 16-bit samples that are stored into the channel memories.

To set the desired bits of vertical resolution, see ":ACQuire:HRESolution" on page 157.

SEGMented In this sampling mode you can view waveform events that are separated by long periods of time without capturing waveform events that are not of interest to you.

SEGPdetect Enables Peak Detect Segmented mode.

SEGHres Enables High Resolution Segmented mode.

To set the desired bits of vertical resolution, see ":ACQuire:HRESolution" on page 157.

Example This example sets the acquisition mode to Real Time Normal.

```
myScope.WriteString ":ACQuire:MODE RTIMe"
```

Query `:ACQuire:MODE?`

The :ACQuire:MODE? query returns the current acquisition sampling mode.

Returned Format `[:ACQuire:MODE] {RTIMe | PDETect | HRESolution | SEGMented}<NL>`

Example This example places the current acquisition mode in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":ACQuire:MODE?"
strMode = myScope.ReadString
Debug.Print strMode
```

:ACQuire:POINts[:ANALog]

(Memory depth)

Command `:ACQuire:POINts[:ANALog] {AUTO | <points_value>}`

The :ACQuire:POINts[:ANALog] command sets the requested analog memory depth for an acquisition. Before you download data from the oscilloscope to your computer, always query the points value with the :WAVeform:POINts? query or :WAVeform:PREamble? query to determine the actual number of acquired points.

You can set the points value to AUTO, which allows the oscilloscope to select the optimum memory depth and display update rate.

<points_value> An integer representing the memory depth.

The range of points available for a channel depends on the oscilloscope settings of sampling mode, sampling rate, and trigger sweep. The following tables show the range of memory values for the different memory options.

Table 6 1G Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution (full sample rate) Modes	16 to 1,049,600,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
High Resolution Mode at < full sample rate	16 to 524,800,000	16 to 262,400,000
Peak Detect Mode at < full sample rate	16 to 262,400,000	16 to 131,200,000
Equivalent Time Mode	16 to 262,144	

Table 7 500M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 512,500,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	

Table 7 500M Memory Option Installed (continued)

Sampling mode and sample rate	Trigger Sweep	
	Single	**Auto or Triggered**
High Resolution Mode at < full sample rate	16 to 512,500,000	16 to 256,250,000
Peak Detect Mode at < full sample rate	16 to 256,250,000	16 to 128,125,000
Equivalent Time Mode	16 to 262,144	

Table 8 200M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	**Auto or Triggered**
Normal, Peak Detect (full sample rate), and High Resolution Modes	16 to 205,000,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
Peak Detect Mode at < full sample rate	16 to 205,000,000	16 to 102,500,000
Equivalent Time Mode	16 to 262,144	

Table 9 100M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	**Auto or Triggered**
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 102,500,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
Peak Detect Mode at < full sample rate	16 to 102,500,000	
Equivalent Time Mode	16 to 262,144	

Table 10 50M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 51,250,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
Peak Detect Mode at < full sample rate	16 to 51,250,000	
Equivalent Time Mode	16 to 262,144	

Table 11 20M Memory Option Installed

Sampling mode and sample rate	Trigger Sweep	
	Single	Auto or Triggered
Normal, Peak Detect (full sample rate), and High Resolution modes	16 to 20,500,000	
Normal and High Resolution modes with averaging	16 to 2,050,000	
Peak Detect Mode at < full sample rate	16 to 20,500,000	
Equivalent Time Mode	16 to 262,144	

Interaction between :ACQuire:SRATe[:ANALog] and :ACQuire:POINts[:ANALog]

If you assign a sample rate value with :ACQuire:SRATe[:ANALog] or a points value using :ACQuire:POINts[:ANALog] the following interactions will occur. "Manual" means you are setting a non-AUTO value for SRATe or POINts.

SRATe	POINts	Result
AUTO	Manual	POINts value takes precedence (sample rate is limited)
Manual	AUTO	SRATe value takes precedence (memory depth is limited)
Manual	Manual	SRATe value takes precedence (memory depth is limited)

Example

This example sets the memory depth to 500 points.

```
myScope.WriteString ":ACQuire:POINts:ANALog 500"
```

Query `:ACQuire:POINts[:ANALog]?`

The :ACQuire:POINts[:ANALog]? query returns the value of the analog memory depth control.

Returned Format `[:ACQuire:POINts:ANALog] <points_value><NL>`

Example This example checks the current setting for memory depth and places the result in the variable, varLength. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:POINts:ANALog?"
varLength = myScope.ReadNumber
Debug.Print FormatNumber(varLength, 0)
```

See Also

- ":WAVeform:DATA?" on page 991

:ACQuire:POINts:AUTO

Command `:ACQuire:POINts:AUTO {{ON | 1} |{OFF | 0}}`

The :ACQuire:POINts:AUTO command enables (automatic) or disables (manual) the automatic memory depth selection control. When enabled, the oscilloscope chooses a memory depth that optimizes the amount of waveform data and the display update rate. When disabled, you can select the amount of memory using the :ACQuire:POINts command.

Example This example sets the automatic memory depth control to off.

```
myScope.WriteString ":ACQUIRE:POINTS:AUTO OFF"
```

Query `:ACQuire:POINts:AUTO?`

The :ACQuire:POINts:AUTO? query returns the automatic memory depth control state.

Returned Format `[:ACQuire:POINts:AUTO] {1 | 0}<NL>`

Example This example checks the current setting for automatic memory depth control and places the result in the variable, varState. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:POINTS:AUTO?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

See Also :WAVeform:DATA?

:ACQuire:POINts:DIGital?

Query `:ACQuire:POINts:DIGital?`

The :ACQuire:POINts:DIGital query returns the current memory depth for the digital channels (MSO models only).

:ACQuire:REDGe

(RealEdge Channel Inputs)

Command

NOTE This command is available for the 50 GHz and 63 GHz bandwidth models of the 90000 Q-Series oscilloscopes family.

```
:ACQuire:REDGe {{ON | 1} | {OFF | 0}}
```

The :ACQuire:REDGe command enables or disables the RealEdge channel inputs.

When RealEdge channel inputs are enabled, the :CHANnel1 and :CHANnel3 commands/queries work for the oscilloscope's 1R and 3R channel inputs, and the :CHANnel2 and :CHANnel4 commands give "hardware missing" or "undefined header" messages.

Also when RealEdge channel inputs are enabled:

- The sampling rate is fixed at 160 GSa/s.
- Real Time Normal is the only acquisition mode available.

Example This example enables the RealEdge channel inputs.

```
myScope.WriteString ":ACQuire:REDGe ON"
```

Query

```
:ACQuire:REDGe?
```

The :ACQuire:REDGe? query returns the current setting for RealEdge channel inputs.

Returned Format

```
[:ACQuire:REDGe] {1 | 0}<NL>
```

Example This example places the current setting for RealEdge channel inputs in the string variable, strRealEdge, then prints the contents of the variable to the computer's screen.

```
Dim strRealEdge As String   ' Dimension variable.
myScope.WriteString ":ACQuire:REDGe?"
strSample = myScope.ReadString
Debug.Print strRealEdge
```

:ACQuire:RESPonse

Command `:ACQuire:RESPonse {FLATmag | GAUSsianmag}`

The Flat Magnitude filter is the default one and is the filter typically used on Infiniium oscilloscopes. The Gaussian Magnitude filter eliminates all ringing (preshoot or overshoot) caused by the oscilloscope's response. Therefore, any ringing you see in the displayed signal is actually in your signal and is not caused by the oscilloscope. The main drawback to using the Gaussian Magnitude Filter is the decrease in bandwidth. Please consult the Flat Magnitude / Magnitude Magnitude Filters topic in the help system for specific information regarding the decrease in bandwidth.

Example This example turns on the Gaussian Magnitude filter.

```
myScope.WriteString ":ACQUIRE:RESPonse GAUSsianmag"
```

Query `:ACQuire:RESPonse?`

The :ACQuire:RESPonse? query returns the current filter being used.

Returned Format `[:ACQuire:POINts:AUTO] {FLATmag | NGAUSsianmag}<NL>`

Example This example checks the current filter setting and places the result in the variable, state. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:RESPonse?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

:ACQuire:SEGMented:COUNt

Command `:ACQuire:SEGMented:COUNt <#segments>`

The :ACQuire:SEGMented:COUNt command sets the number of segments to acquire in the segmented memory mode.

<#segments> An integer representing the number of segments to acquire.

Example This example sets the segmented memory count control to 1000.

```
myScope.WriteString ":ACQUIRE:SEGMented:COUNt 1000"
```

Query `:ACQuire:SEGMented:COUNt?`

The :ACQuire:SEGMented:COUNT? query returns the number of segments control value.

Returned Format `[:ACQuire:SEGMented:COUNt] <#segments><NL>`

Example This example checks the current setting for segmented memory count control and places the result in the variable, varSegments. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:SEGMents:COUNt?"
varSegments = myScope.ReadNumber
Debug.Print FormatNumber(varSegments, 0)
```

:ACQuire:SEGMented:INDex

Command `:ACQuire:SEGMented:INDex <index#>`

The :ACQuire:SEGMented:INDex command sets the index number for the segment that you want to display on screen in the segmented memory mode. If an index value larger than the total number of acquired segments is sent, an error occurs indicating that the data is out of range and the segment index is set to the maximum segment number.

<index#> An integer representing the index number of the segment that you want to display.

Example This example sets the segmented memory index number control to 1000.

```
myScope.WriteString ":ACQUIRE:SEGMented:INDex 1000"
```

Query `:ACQuire:SEGMented:INDex?`

The :ACQuire:SEGMented:INDex? query returns the segmented memory index number control value.

Returned Format `[:ACQuire:SEGMented:INDex] <index#><NL>`

Example This example checks the current setting for segmented memory index number control and places the result in the variable, varIndex. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:SEGMents:INDex?"
varIndex = myScope.ReadNumber
Debug.Print FormatNumber(varIndex, 0)
```

:ACQuire:SEGMented:TTAGs

Command `:ACQuire:SEGMented:TTAGs {{ON | 1} | {OFF | 0}}`

The :ACQuire:SEGMented:TTAGs command turns the time tags feature on or off for the segmented memory sampling mode.

Example This example turns the time tags on for segmented memory.

```
myScope.WriteString ":ACQUIRE:SEGMented:TTAGs ON"
```

Query `:ACQuire:SEGMented:TTAGs?`

The :ACQuire:SEGMented:TTAGs? query returns the segmented memory time tags control value.

Returned Format `[:ACQuire:SEGMented:TTAGs] {1 | 0}<NL>`

Example This example checks the current setting for segmented memory time tags control and places the result in the variable, varTimeTags. Then the program prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":ACQUIRE:SEGMents:TTAGs?"
varTimeTags = myScope.ReadNumber
Debug.Print FormatNumber(varTimeTags, 0)
```

:ACQuire:SRATe[:ANALog]

(Analog Sample Rate)

Command `:ACQuire:SRATe[:ANALog] {AUTO | MAX | <rate>}`

The :ACQuire:SRATe[:ANALog] command sets the analog acquisition sampling rate.

AUTO The AUTO rate allows the oscilloscope to select a sample rate that best accommodates the selected memory depth and horizontal scale.

MAX The MAX rate enables the oscilloscope to select maximum available sample rate.

<rate> A real number representing the sample rate. You can send any value, but the value is rounded to the next fastest sample rate. For a list of available sample rate values see "SRATe Sample Rate Tables" on page 173.

Interaction between :ACQuire:SRATe[:ANALog] and :ACQuire:POINts[:ANALog] If you assign a sample rate value with :ACQuire:SRATe[:ANALog] or a points value using :ACQuire:POINts[:ANALog] the following interactions will occur. "Manual" means you are setting a non-AUTO value for SRATe or POINts.

SRATe	POINts	Result
AUTO	Manual	POINts value takes precedence (sample rate is limited)
Manual	AUTO	SRATe value takes precedence (memory depth is limited)
Manual	Manual	SRATe value takes precedence (memory depth is limited)

Example This example sets the sample rate to 250 MSa/s.

```
myScope.WriteString ":ACQuire:SRATe:ANALog 250E+6"
```

Query `:ACQuire:SRATe[:ANALog]?`

The :ACQuire:SRATe[:ANALog]? query returns the current analog acquisition sample rate.

Returned Format `[:ACQuire:SRATe:ANALog] {<rate>}<NL>`

Example This example places the current sample rate in the string variable, strSample, then prints the contents of the variable to the computer's screen.

```
Dim strSample As String   ' Dimension variable.
myScope.WriteString ":ACQuire:SRATe:ANALog?"
```

```
strSample = myScope.ReadString
Debug.Print strSample
```

SRATe Sample Rate Tables

The following tables show the range of point values for the different oscilloscope modes and model numbers.

Table 12 91304A, 91204A, 90804A Models Sample Rate Values (in Sa/s)Normal Sampling Mode

10	20	25	40	50	100	200	250	400	500	1k	2k	2.5k	4k	5k
10k	20k	25k	40k	50k	100k	200k	250k	400k	500k	1M	2M	2.5M	4M	5M
10M	20M	25M	40M	50M	100 M	125 M	200 M	250 M	400 M	500 M	1G	1.25 G	2G	2.50 G
4G	5G	10G	20G	40G										

Table 13 90604A, 90404A, 90254A Models Sample Rate Values (in Sa/s)Normal Sampling Mode

10	20	25	40	50	100	200	250	400	500	1k	2k	2.5k	4k	5k
10k	20k	25k	40k	50k	100k	200k	250k	400k	500k	1M	2M	2.5M	4M	5M
10M	20M	25M	40M	50M	100 M	125 M	200 M	250 M	400 M	500 M	1G	1.25 G	2G	2.5G
4G	5G	10G	20G											

Table 14 Sample Rate Values (in Sa/s)Peak Detect Sampling

Table 15 Sample Rate Values (in Sa/s)High Resolution Sampling

:ACQuire:SRATe[:ANALog]:AUTO

Command `:ACQuire:SRATe[:ANALog]:AUTO {{ON | 1} | {OFF | 0}}`

The :ACQuire:SRATe[:ANALog]:AUTO command enables (ON) or disables (OFF) the automatic analog sampling rate selection control. On the oscilloscope front-panel interface, ON is equivalent to Automatic and OFF is equivalent to Manual.

Example This example changes the sampling rate to manual.

```
myScope.WriteString ":ACQuire:SRATe:ANALog:AUTO OFF"
```

Query `:ACQuire:SRATe[:ANALog]:AUTO?`

The :ACQuire:SRATe[:ANALog]:AUTO? query returns the current acquisition sample rate.

Returned Format `[:ACQuire:SRATe:ANALog:AUTO] {1 | 0}<NL>`

Example This example places the current analog sample rate in the variable, varSample, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SRATe:ANALog:AUTO?"
varSample = myScope.ReadNumber
Debug.Print FormatNumber(varSample, 0)
```

:ACQuire:SRATe:DIGital

(Digital Channels Sample RATe)

Command `:ACQuire:SRATe:DIGital {AUTO | MAX | <rate>}`

The :ACQuire:SRATe:DIGital command sets the digital acquisition sampling rate.

AUTO The AUTO rate allows the oscilloscope to select a sample rate that best accommodates the selected memory depth and horizontal scale.

MAX The MAX rate enables the oscilloscope to select maximum available sample rate.

<rate> A real number representing the digital sample rate. You can send any value, but the value is rounded to the next fastest sample rate.

Interaction between :ACQuire:SRATe:DIGital and :ACQuire:POINts:DIGital If you assign a sample rate value with :ACQuire:SRATe:DIGital, the digital memory depth is automatically adjusted and can be seen by using the query :ACQuire:POINts:DIGital

Query `:ACQuire:SRATe:DIGital?`

The :ACQuire:SRATe:DIGital? query returns the current digital acquisition sample rate.

Returned Format `[:ACQuire:SRATe:DIGital] {<rate>}<NL>`

Example This example places the current digital channel sample rate in the string variable, strSample, then prints the contents of the variable to the computer's screen.

```
Dim strSample As String   ' Dimension variable.
myScope.WriteString ":ACQuire:SRATe:DIGital?"
strSample = myScope.ReadString
Debug.Print strSample
```

:ACQuire:SRATe:DIGital:AUTO

Command `:ACQuire:SRATe:DIGital:AUTO {{ON | 1} | {OFF | 0}}`

The :ACQuire:SRATe:DIGital:AUTO command enables (ON) or disables (OFF) the automatic digital channel sampling rate selection control.

Example This example changes the digital channel sampling rate to manual.

```
myScope.WriteString ":ACQuire:SRATe:DIGital:AUTO OFF"
```

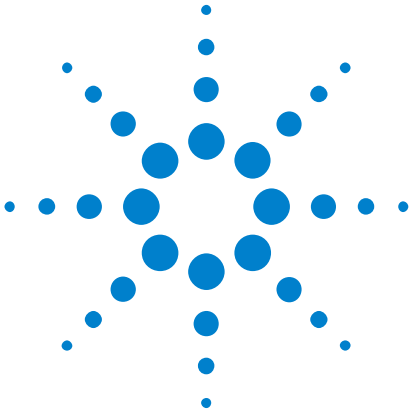
Query `:ACQuire:SRATe:DIGital:AUTO?`

The :ACQuire:SRATe:DIGital:AUTO? query returns the current digital channel acquisition sample rate.

Returned Format `[:ACQuire:SRATe:DIGital:AUTO] {1 | 0}<NL>`

Example This example places the current digital channel sample rate in the variable, varSample, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":ACQuire:SRATe:DIGital:AUTO?"
varSample = myScope.ReadNumber
Debug.Print FormatNumber(varSample, 0)
```



10 Bus Commands

:BUS:B<N>:TYPE 178
:BUS<N>:BIT<M> 179
:BUS<N>:BITS 180
:BUS<N>:CLEar 181
:BUS<N>:CLOCk 182
:BUS<N>:CLOCk:SLOPe 183
:BUS<N>:DISPlay 184
:BUS<N>:LABel 185
:BUS<N>:READout 186

NOTE The :BUS:B<N>:TYPE command applies to oscilloscopes with the serial data analysis option installed. The other :BUS<N> commands apply to mixed-signal oscilloscopes (MSOs).

:BUS:B<N>:TYPE

Command `:BUS:B<N>:TYPE <protocol>`

NOTE This BUS command only applies to oscilloscopes with the serial data analysis option installed.

The :BUS:B<N>:TYPE command sets the type of protocol being analyzed for a serial bus waveform.

<protocol> {CAN | DVI | FIBRechannel | E10GBASEKR | FLEXray | GEN8B10B | GENeric | HOTLink | IIC | INFiniband | JTAG | LIN | MIPI | MPHY | PCI3 | PCIexpress | SAS | SATA | SPI | UART | USB2 | USB3 | XAUI}

<N> An integer, 1-4.

Example This example sets the serial bus waveform number one protocol type to FLEXray.

```
myScope.WriteString ":BUS:B1:TYPE FLEXRAY"
```

Query `:BUS:B<N>:TYPE?`

The :BUS:B<N>:TYPE? query returns the name of the protocol being used for the serial bus.

Returned Format `[:BUS:B<N>:TYPE] <protocol><NL>`

:BUS<N>:BIT<M>

Command `:BUS<N>:BIT<M> {ON | OFF | 1 | 0}`

NOTE The BUS commands only apply to the MSO oscilloscopes.

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) then the bit is included in the definition. If the parameter is a 0 (OFF) then the bit is excluded from the definition. The digital subsystem must be enabled for this command will work. See ENABle command in the root subsystem.

<M> An integer, 0-15.

<N> An integer, 1-4.

Example This example includes bit 1 as part of the bus 1 definition.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":BUS1:BIT1 ON"
```

Query `: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.

Returned Format `[:BUS<N>:BIT<M>] {1 | 0}<NL>`

:BUS<N>:BITS

Command `:BUS<N>:BITS <channel_list>,{ON | OFF| 1 | 0}`

NOTE The BUS commands only apply to the MSO oscilloscopes.

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. The digital subsystem must be enabled for this command will work. See ENABle command in the root subsystem.

<N> An integer, 1- 4.

<channel_list> The channel range is from 0 to 15 in the following format.

(@1,5,7,9)	channels 1, 5, 7, and 9 are turned on.
(@1:15)	channels 1 through 15 are turned on.
(@1:5,8,14)	channels 1 through 5, channel 8, and channel 14 are turned on.

NOTE The parenthesizes are part of the expression and are necessary.

Example This example includes bits 1, 2, 4, 5, 6, 7, 8, and 9 as part of the bus 1 definition.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":BUS1:BITS (@1,2,4:9),ON"
```

Query `:BUS<N>:BITS?`

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

Returned Format `[:BUS<N>:BITS] <channel_list>,{1 | 0}<NL>`

:BUS<N>:CLEar

Command `:BUS<N>:CLEar`

NOTE The BUS commands only apply to the MSO oscilloscopes.

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

<N> An integer, 1-4.

Example This example excludes all the digital channels from the bus 1 definition.

```
myScope.WriteString ":BUS1:CLEAR"
```

:BUS<N>:CLOCk

Command `:BUS<N>:CLOCk {CHANnel<N> | DIGital<M> | NONE}`

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<N>:CLOCk command sets the digital or analog channel used as the clock for decoding the bus values.

<M> An integer, 0-15.

<N> An integer, 1-4.

<O> An integer, 1-4.

Example This example sets the clock to channel 1 for bus 1.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":BUS1:CLOCK CHANNEL1"
```

Query `:BUS<N>:CLOCK?`

The :BUS<N>:CLOCk query returns the channel being used for the specified bus.

Returned Format `[:BUS<N>:CLOCk] {CHANnel<O> | DIGital<M> | NONE}<NL>`

:BUS<N>:CLOCk:SLOPe

Command `:BUS<N>:CLOCk:SLOPe {RISing | FALLing | EITHer}`

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<N>:CLOCk:SLOPe command sets the clock edge used for decoding the bus values.

<0> An integer, 1-4.

Example This example sets the clock edge to falling for bus 1.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":BUS1:CLOCk:SLOPE FALLING"
```

Query `:BUS<N>:CLOCK:SLOPe?`

The :BUS<N>:CLOCk:SLOPe query returns the clock edge being used for the specified bus.

Returned Format `[:BUS<N>:CLOCk:SLOPe] {RISing | FALLing | EITHer}<NL>`

:BUS<N>:DISPlay

Command `:BUS<N>[:DISPlay] {ON | OFF | 1 | 0}`

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<N>:DISPlay command enables or disables the view of the selected bus. The digital subsystem must be enabled before this command will work. See the ENABle command in the root subsystem.

<N> An integer, 1- 4.

Example This example enables the viewing of bus 1.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":BUS1 ON"
```

Query `:BUS<N>[:DISPlay]?`

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

Returned Format `[:BUS<N>] {1 | 0}<NL>`

:BUS<N>:LABel

Command `:BUS<N>:LABel <quoted_string>`

NOTE The BUS commands only apply to the MSO oscilloscopes.

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 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.

<N> An integer, 1- 4.

<quoted_string> A series of 6 or less characters as a quoted ASCII string.

Example This example sets the bus 1 label to Data.

```
myScope.WriteString ":BUS1:LABEL ""Data"""
```

Query `:BUS<N>:LABel?`

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

Returned Format `[:BUS<N>:LABel] <quoted_string><NL>`

:BUS<N>:READout

Command `:BUS<N>:READout {NONE | BINary | DECimal | HEX | OCTal | SIGNed | SYMBol`

NOTE The BUS commands only apply to the MSO oscilloscopes.

The :BUS<N>:READout command changes the format of the numbers displayed in the bus waveform.

<N> An integer, 1-4.

Example This example sets the bus read out to decimal.

```
myScope.WriteString ":BUS1:READOUT DECIMAL"
```

Query `:BUS<N>:READout?`

The :BUS<N>:READout? query returns the format of the readout control.

Returned Format `[:BUS<N>:READout] {NONE | BINary | DECimal | HEX | OCTal | SIGNed | SYMBol}<NL>`

11 Calibration Commands

:CALibrate:DATE? 189
:CALibrate:OUTPut 190
:CALibrate:SKEW 191
:CALibrate:STATus? 192
:CALibrate:TEMP? 193

This chapter briefly explains the calibration of the oscilloscope. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used.

The commands in the CALibration subsystem allow you to change the output of the front-panel Aux Out connector, adjust the skew of channels, and check the status of calibration.

These CALibration commands and queries are implemented in the Infiniium oscilloscopes:

This chapter briefly explains the calibration of the oscilloscope. It is intended to give you and the calibration lab personnel an understanding of the calibration procedure and how the calibration subsystem is intended to be used.

Oscilloscope Calibration

Oscilloscope calibration establishes calibration factors for the oscilloscope. These factors are stored on the oscilloscope's hard disk.

- Initiate the calibration from the "Utilities Calibration" menu.

You should calibrate the oscilloscope periodically (at least annually), or if the ambient temperature since the last calibration has changed more than ±5 °C. The temperature change since the last calibration is shown on the calibration status screen which is found under the "Utilities Calibration" dialog. It is the line labeled "Calibration Δ Temp: _ °C."

See also the oscilloscope's *Service Guide* has more details about the calibration.

Probe Calibration

Probe calibration establishes the gain and offset of a probe that is connected to a channel of the oscilloscope, and applies these factors to the calibration of that channel.

- Initiate probe calibration from the "Setup > Channel > Probes > Calibrate Probe" menu.

  To achieve the specified accuracy (±2%) with a probe connected to a channel, make sure the oscilloscope is calibrated.

  - For probes that the oscilloscope can identify through the probe power connector, like the 1158A, the oscilloscope automatically adjusts the vertical scale factors for that channel even if a probe calibration is not performed.
  - For nonidentified probes, the oscilloscope adjusts the vertical scale factors only if a probe calibration is performed.

- If you do not perform a probe calibration but want to use an unidentified probe, enter the attenuation factor in the "Setup > Channel > Probes > Configure Probing System > User Defined Probe" menu.

  - If the probe being calibrated has an attenuation factor that allows the oscilloscope to adjust the gain (in hardware) to produce even steps in the vertical scale factors, the oscilloscope will do so.
  - If the probe being calibrated has an unusual attenuation, like 3.75, the oscilloscope may have to adjust the vertical scale factors to an unusual number, like 3.75 V/div.

  Typically, probes have standard attenuation factors such as divide by 10, divide by 20, or divide by 100.

:CALibrate:DATE?

Query `:CALibrate:DATE? CHANnel<N>`

The :CALibrate:DATE? query returns the date and time of the last time scale and regular user calibration.

The :CALibrate:DATE? query returns two calibration dates and times:

- The date and time of the last time scale calibration.
- The date and time of the last regular user calibration.

Returned Format `[:CALibrate:DATE] <ts_date_time>,<user_date_time><NL>`

The string returned is formatted like "<day> <month> <year> <hours>:<minutes>:<seconds>,<day> <month> <year> <hours>:<minutes>:<seconds>", for example "31 MAY 2013 12:52:45,4 DEC 2012 10:59:52".

:CALibrate:OUTPut

Command `:CALibrate:OUTPut {AC | TRIGOUT | DC,<dc_value> | ZERO | ONE | DPULse | SOSC | HFOSC}`

The :CALibrate:OUTPut command sets the coupling frequency, trigger output pulse, and dc level of the calibrator waveform output through the front-panel Aux Out connector:

- AC — sets the Aux Out to be the probe compensation square wave (approximately 750 Hz).
- TRIGOUT — outputs a pulse when the trigger event occurs. Use this to trigger other instruments.
- DC,<dc_value> — a DC level value in volts. The <dc_value> is a real number from -2.4 V to +2.4 V dc.
- ZERO — a -450 mV DV level
- ONE — a -150 mV DC level.
- DPULse — A double-pulse signal.
- SOSC — The 100 MHz reference clock output.
- HFOSC — A high-frequency oscillator output.

Example This example puts a DC voltage of 2.0 volts on the oscilloscope front-panel Aux Out connector.

```
myScope.WriteString ":CALibrate:OUTPut DC,2.0"
```

Query `:CALibrate:OUTPut?`

The :CALibrate:OUTPut? query returns the current setup.

Returned Format `[:CALibrate:OUTPut] {AC | TRIGOUT | DC,<dc_value>}`

Example This example places the current selection for the DC calibration to be printed in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   'Dimension variable
myScope.WriteString ":CALibrate:OUTPut?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:CALibrate:SKEW

Command `:CALibrate:SKEW CHANnel<N>,<skew_value>`

The :CALibrate:SKEW command sets the channel-to-channel skew factor for a channel. The numeric argument is a real number in seconds, which is added to the current time base position to shift the position of the channel's data in time. Use this command to compensate for differences in the electrical lengths of input paths due to cabling and probes.

<N> An integer, 1-4.

<skew_value> A real number, in seconds.

Example This example sets the oscilloscope channel 1 skew to 1 µs.

```
myScope.WriteString ":CALIBRATE:SKEW CHANNEL1,1E-6"
```

Query `:CALibrate:SKEW? CHANnel<N>`

The :CALibrate:SKEW? query returns the current skew value.

Returned Format `[:CALibrate:SKEW] <skew_value><NL>`

:CALibrate:STATus?

Query `:CALibrate:STATus?`

The :CALibrate:STATus? query returns the calibration status of the oscilloscope. These are ten, comma-separated integers, with 1, 0, or -1. A "1" indicates pass, a "0" indicates fail and a "-1" indicates unused. This matches the status in the Calibration dialog box in the Utilities menu.

Returned Format `[:CALibrate:STATus] <status>`

<status> <Oscilloscope Frame Status>, <Channel1 Vertical>, <Channel1 Trigger>, <Channel2 Vertical>, <Channel2 Trigger>, <Channel3 Vertical>, <Channel3 Trigger>, <Channel4 Vertical>, <Channel4 Trigger>, <Aux Trigger>

:CALibrate:TEMP?

Query `:CALibrate:TEMP? CHANnel<N>`

The :CALibrate:TEMP? query returns two delta temperature values in Celsius:

- Between the current temp and the temp of the last time scale calibration.
- Between the current temp and the temp of the last regular user calibration.

Returned Format `[:CALibrate:TEMP] <ts_delta_temp>,<user_delta_temp><NL>`

For example, the string returned could be "-1,0". A difference in the two delta values of one degree is not uncommon.

12 Channel Commands

:CHANnel<N>:COMMonmode 197
:CHANnel<N>:DIFFerential 198
:CHANnel<N>:DIFFerential:SKEW 199
:CHANnel<N>:DISPlay 200
:CHANnel<N>:DISPlay:AUTO 201
:CHANnel<N>:DISPlay:OFFSet 202
:CHANnel<N>:DISPlay:RANGe 203
:CHANnel<N>:DISPlay:SCALe 204
:CHANnel<N>:INPut 205
:CHANnel<N>:ISIM:APPLy 206
:CHANnel<N>:ISIM:BANDwidth 207
:CHANnel<N>:ISIM:BWLimit 208
:CHANnel<N>:ISIM:CONVolve 209
:CHANnel<N>:ISIM:CORRection 210
:CHANnel<N>:ISIM:DEConvolve 212
:CHANnel<N>:ISIM:DELay 213
:CHANnel<N>:ISIM:NORMalize 215
:CHANnel<N>:ISIM:PEXTraction 216
:CHANnel<N>:ISIM:SPAN 218
:CHANnel<N>:ISIM:STATe 219
:CHANnel<N>:LABel 220
:CHANnel<N>:OFFSet 221
:CHANnel<N>:PROBe 222
:CHANnel<N>:PROBe:ACCAL 223
:CHANnel<N>:PROBe:ATTenuation 224
:CHANnel<N>:PROBe:AUTozero 225
:CHANnel<N>:PROBe:COUPling 226
:CHANnel<N>:PROBe:EADapter 227
:CHANnel<N>:PROBe:ECOupling 229
:CHANnel<N>:PROBe:EXTernal 230
:CHANnel<N>:PROBe:EXTernal:GAIN 231
:CHANnel<N>:PROBe:EXTernal:OFFSet 232
:CHANnel<N>:PROBe:EXTernal:UNITs 233
:CHANnel<N>:PROBe:GAIN 234

:CHANnel<N>:PROBe:HEAD:ADD 235
:CHANnel<N>:PROBe:HEAD:DELete ALL 236
:CHANnel<N>:PROBe:HEAD:SELect 237
:CHANnel<N>:PROBe:HEAD:VTERm 238
:CHANnel<N>:PROBe:ID? 239
:CHANnel<N>:PROBe:MODE 240
:CHANnel<N>:PROBe:PRECprobe:BANDwidth 241
:CHANnel<N>:PROBe:PRECprobe:CALibration 242
:CHANnel<N>:PROBe:PRECprobe:MODE 243
:CHANnel<N>:PROBe:PRECprobe:ZSRC 244
:CHANnel<N>:PROBe:SKEW 246
:CHANnel<N>:PROBe:STYPe 247
:CHANnel<N>:RANGe 248
:CHANnel<N>:SCALe 249
:CHANnel<N>:UNITs 250

The CHANnel subsystem commands control all vertical (Y axis) functions of the oscilloscope. You may toggle the channel displays on and off with the root level commands :VIEW and :BLANk, or with :CHANnel:DISPlay.

NOTE

In this section, you can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

:CHANnel<N>:COMMonmode

Command `:CHANnel<N>:COMMonmode {{ON | 1} | {OFF | 0}}`

The :CHANnel<N>:COMMonmode command turns on/off common mode for the channel. Channels 2 and 4 may form a common mode channel and Channels 1 and 3 may form a common mode channel.

<N> An integer, 1-4.

Example This example turns channel 1 common mode channel on (channel 1 + channel 3).

```
myScope.WriteString ":CHANnel1:COMMonmode ON"
```

Query `:CHANnel<N>:COMMonmode?`

The :CHANnel<N>:COMMonmode? query returns whether the channel is in commonmode or not.

Returned Format `[:CHANnel<N>:COMMonmode] {1 | 0}<NL>`

Example This example places the current common mode setting of the channel 1 display in the variable varComm, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:COMMonmode?"
varComm = myScope.ReadNumber
Debug.Print FormatNumber(varComm, 0)
```

:CHANnel<N>:DIFFerential

Command `:CHANnel<N>:DIFFerential {{ON | 1} | {OFF | 0}}`

The :CHANnel<N>:DIFFerential command turns on/off differential mode for the channel. Channels 1 and 3 may form a differential channel and Channels 2 and 4 may form a differential channel.

<N> An integer, 1-4.

Example This example turns channel 1 differential on (channel 1 - channel 3).

```
myScope.WriteString ":CHANNEL1:DIFFerential ON"
```

Query `:CHANnel<N>:DIFFerential?`

The :CHANnel<N>:DIFFerential? query returns whether the channel is in differential mode or not.

Returned Format `[:CHANnel<N>:DIFFerential] {1 | 0}<NL>`

Example This example places the current differential setting of the channel 1 display in the variable varDiff, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DIFFerential?"
varDiff = myScope.ReadNumber
Debug.Print FormatNumber(varDiff, 0)
```

:CHANnel<N>:DIFFerential:SKEW

Command `:CHANnel<N>:DIFFerential:SKEW <skew>`

The :CHANnel<N>:DIFFerential:SKEW <skew> command sets the skew that is applied to the differential or common mode pair of channels.

<skew> A real number for the skew value

Example This example sets the skew applied to the channel 1 - channel 3 differential channel to 10 µs.

```
myScope.WriteString ":CHANNEL1:DIFFerential:SKEW 10E-6"
```

Query `:CHANnel<N>:DIFFerential:SKEW?`

The :CHANnel<N>:DIFFerential:SKEW? query returns the skew that is applied to the differential or common mode pair of channels.

Returned Format `[:CHANnel<N>:DIFFerential:SKEW] <skew_value><NL>`

Example This example places the current skew setting of the channel 1 - channel 3 differential channel in the variable varSkew, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DIFFerential:SKEW?"
varSkew = myScope.ReadNumber
Debug.Print FormatNumber(varSkew, 0)
```

:CHANnel<N>:DISPlay

Command `:CHANnel<N>:DISPlay {{ON | 1} | {OFF | 0}}`

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

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example sets channel 1 display to on.

```
myScope.WriteString ":CHANNEL1:DISPLAY ON"
```

Query `:CHANnel<N>:DISPlay?`

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

Returned Format `[:CHANnel<N>:DISPlay] {1 | 0}<NL>`

Example This example places the current setting of the channel 1 display in the variable varDisplay, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DISPLAY?"
varDisplay = myScope.ReadNumber
Debug.Print FormatNumber(varDisplay, 0)
```

:CHANnel<N>:DISPlay:AUTO

Command `:CHANnel<N>:DISPlay:AUTO {{ON | 1} | {OFF | 0}}`

The :CHANnel<N>:DISPlay:AUTO command sets the differential and common mode display scale and offset to track the acquisition scale and offset.

<N> An integer, 1-4.

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example This example sets the channel 1 - channel 3 differential channel display scale and offset to track the acquisition scale and offset.

```
myScope.WriteString ":CHANNEL1:DISPLAY:AUTO ON"
```

Query `:CHANnel<N>:DISPlay:AUTO?`

The :CHANnel<N>:DISPlay:AUTO? query returns whether or not the differential or common mode display scale and offset are tracking the acquisition scale and offset.

Returned Format `[:CHANnel<N>:DISPlay:AUTO] {1 | 0}<NL>`

Example This example places whether or not the channel 1 - channel 3 differential channel display scale and offset is tracking the acquisition scale and offset in the variable varAuto, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DISPLAY:AUTO?"
varAuto = myScope.ReadNumber
Debug.Print FormatNumber(varAuto, 0)
```

:CHANnel<N>:DISPlay:OFFSet

Command `:CHANnel<N>:DISPlay:OFFSet <value>`

The :CHANnel<N>:DISPlay:OFFSet command sets the displayed offset of the selected channel. Setting the display range turns off display auto.

<value> A real number for the value variable

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

---

Example This example sets the displayed offset of channel 1 to

```
myScope.WriteString ":CHANNEL1:DISPLAY:RANGe 10e-6"
```

Query `:CHANnel<N>:DISPlay:OFFSet?`

The :CHANnel<N>:DISPlay:OFFSet? query returns the displayed offset for the selected channel.

Returned Format `[:CHANnel<N>:DISPlay:OFFSet] <value><NL>`

Example This example places the displayed offset of channel 1 in the variable varOffset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DISPLAY:OFFSet?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, 0)
```

:CHANnel<N>:DISPlay:RANGe

Command `:CHANnel<N>:DISPlay:RANGe <range>`

The :CHANnel<N>:DISPlay:RANGe command sets the full scale vertical range of the display of the selected channel. Setting the display range turns off display auto.

<range> A real number for the range value

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

Example This example sets the display range of the display of channel 1 to

```
myScope.WriteString ":CHANNEL1:DISPLAY:RANGe 10e-6"
```

Query `:CHANnel<N>:DISPlay:RANGe?`

The :CHANnel<N>:DISPlay:RANGe? query returns the full scale vertical range of the display for the selected channel.

Returned Format `[:CHANnel<N>:DISPlay:RANGe] <range><NL>`

Example This example places the range of channel 1 in the variable varRange, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":CHANNEL1:DISPLAY:RANGe?"
varRange = myScope.ReadNumber
Debug.Print FormatNumber(varRange, 0)
```

:CHANnel<N>:DISPlay:SCALe

Command `:CHANnel<N>:DISPlay:SCALe <scale>`

The :CHANnel<N>:DISPlay:SCALe command sets the displayed scale of the selected channel per division. Setting the display range turns off display auto.

<scale> A real number for the scale value

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

---

Example This example sets the display scale of channel 1 per division to

```
myScope.WriteString ":CHANNEL1:DISPLAY:SCALe 10e-6"
```

Query `:CHANnel<N>:DISPlay:SCALe?`

The :CHANnel<N>:DISPlay:SCALe? query returns the displayed scale of the selected channel per division.

Returned Format `[:CHANnel<N>:DISPlay:SCALe] <scale><NL>`

Example This example places the display scale of channel 1 in the variable varScale, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:DISPLAY:SCALe?"
varScale = myScope.ReadNumber
Debug.Print FormatNumber(varScale, 0)
```

:CHANnel<N>:INPut

Command `:CHANnel<N>:INPut {DC50 | DCFifty}`

The :CHANnel<N>:INPut command selects the input coupling and impedance for the specified channel and is always DC coupling, 50Ω input impedance. This command is provided for compatibility with other Infiniium oscilloscopes.

<N> An integer, 1-4.

Example This example sets the channel 1 input to DC50.

```
myScope.WriteString ":CHANNEL1:INPut DC50"
```

Query `:CHANnel<N>:INPut?`

The :CHANnel<N>:INPut? query returns the selected channel input parameter and is always DC coupling, 50Ω input impedance. .

Returned Format `[CHANnel<N>:INPut] DC50<NL>`

Example This example puts the current input for channel 1 in the string variable, strInput. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:INPUT?"
strInput = myScope.ReadString
Debug.Print strInput
```

:CHANnel<N>:ISIM:APPLy

Command `:CHANnel<N>:ISIM:APPLy "<transfer_funct_file>"`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:APPLy command applies a pre-computed transfer function to the waveform. If InfiniiSim is in 2 port mode, the file must be a .tf2 file. If in 4 port mode, the file must be a .tf4 file. Use the ISIM:STATe command to enable InfiniiSim before issuing the APPLy command.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<transfer_funct_file> The full path to the .tf2 file name (if in 2 port mode) or the .tf4 file (if in 4 port mode).

Example This example applies the example.tf4 file to the waveform on channel 1.

```
myScope.WriteString _
 ":CHANNEL1:ISIM:APPLy " + _
 """C:\Users\Public\Documents\Infiniium\Filters\example.tf4"""
```

Query `:CHANnel<N>:ISIM:APPLy?`

The :CHANnel<N>:ISIM:APPLy? query returns the currently selected function file name when 2 port or 4 port mode is enabled.

Returned Format `[CHANnel<N>:ISIM:APPLy] <file_name><NL>`

Example This example puts the current transfer function file name in the variable strFile. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":CHANNEL1:ISIM:APPLy?"
strFile = myScope.ReadString
Debug.Print strFile
```

:CHANnel<N>:ISIM:BANDwidth

Command `:CHANnel<N>:ISIM:BANDwidth <bw_value>`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:BANDwidth command sets the Bandwidth Limit field in the InfiniiSim GUI to a desired value (sets the bandwidth limit cutoff frequency). The CHANnel<N>:ISIM:BWLimit command turns this feature on or off. Please refer to the InfiniiSim User's Guide on your oscilloscope or on Agilent.com for more explanation regarding this field.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<bw_value> The maximum value is the sample rate / 2. The minimum value is 1000 Hz.

Example This example sets the channel 1 input bandwidth limit cutoff frequency to 2 GHz.

```
myScope.WriteString ":CHANNEL1:ISIM:BANDwidth 2e9"
```

Query `:CHANnel<N>:ISIM:BANDwidth?`

The :CHANnel<N>:ISIM:BANDwidth? query returns the selected channel input's bandwidth limit cutoff frequency.

Returned Format `[CHANnel<N>:ISIM:BANDwidth] <parameter><NL>`

Example This example puts the current input for channel 1 in the string variable, varBwLimit. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:ISIM:BANDwidth?"
varBwLimit = myScope.ReadNumber
Debug.Print FormatNumber(varBwLimit, 0)
```

:CHANnel<N>:ISIM:BWLimit

Command `:CHANnel<N>:ISIM:BWLimit {{ON | 1} | {OFF | 0}}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:BWLimit command activates or deactivates the Bandwidth Limit field in the InfiniiSim GUI. This field sets the bandwidth limit cutoff frequency. The CHANnel<N>:ISIM:BANDwidth command sets the value to be used when this field is activated. Please refer to the InfiniiSim User's Guide on your oscilloscope or on Agilent.com for more explanation regarding this field.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example turns on the InfiniiSim bandwidth limit feature for channel 1.

```
myScope.WriteString ":CHANNEL1:ISIM:BWLimit ON"
```

Query `:CHANnel<N>:ISIM:BWLimit?`

The :CHANnel<N>:ISIM:BWLimit? query returns the current state of the corresponding channel's InfiniiSim bandwidth limiting feature.

Returned Format `[CHANnel<N>:ISIM:BWLimit] {1 | 0}<NL>`

Example This example puts the current InfiniiSim bandwidth limit state for channel 1 in the string variable, varLimit. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:ISIM:BWLimit?"
varLimit = myScope.ReadNumber
Debug.Print FormatNumber(varLimit, 0)
```

:CHANnel<N>:ISIM:CONVolve

Command `:CHANnel<N>:ISIM:CONVolve <s_parameter_file>, {OFF | ON}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:CONVolve command convolves the indicated S-parameter file with the waveform. This command only uses a single S21 component block. If a .s4p file is indicated, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files. Optionally, include ON to flip the port numbering when reading the s-parameter file.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<s_parameter_file> The name of the s-parameter file.

Example This example convolves the s-parameter file example.s2p with the waveform on channel 1.

```
myScope.WriteString ":CHANNEL1:ISIM:CONVolve example.s2p"
```

:CHANnel<N>:ISIM:CORRection

Command `:CHANnel<N>:ISIM:CORRection <percent>`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:CORRection command sets the amount of linearly scaled correction applied to the non-DC frequency components of the measured signal. This lets you trade off the amount of correction to apply via the transformation function versus the increase in noise it may create at higher frequencies. In other words, you can fine-tune the amount of high-frequency noise versus the sharpness of the step response edge.

<N> An integer, 1-4.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<percent> If you are making averaged mode measurements or applying a transfer function that does not magnify the noise, use the full correction by setting this field to 100%.

However, if you are working with eye diagrams or making jitter measurements and the transfer function is magnifying the noise, you may want to limit the correction by selecting a lower percentage.

Example This example sets the channel 1 InfiniiSim correction factor to 80%.

```
myScope.WriteString ":CHANnel1:ISIM:CORRection 80"
```

Query `:CHANnel<N>:ISIM:CORRection?`

The :CHANnel<N>:ISIM:CORRection? query returns the selected input channel's percent correction factor.

Returned Format `[CHANnel<N>:ISIM:CORRection] <percent><NL>`

Example This example gets the current channel 1 InfiniiSim correction percentage and places it in the numeric variable, varIsimCorrection. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:CORRection?"
varIsimCorrection = myScope.ReadNumber
Debug.Print FormatNumber(varBwLimit, 0)
```

:CHANnel<N>:ISIM:DEConvolve

Command `:CHANnel<N>:ISIM:DEConvolve <s_parameter_file>, {OFF | ON}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:DEConvolve command deconvolves the indicated S-parameter file with the waveform. This command only uses a single S21 component block. If a .s4p file is indicated, ports 1 and 2 are used assuming a 1-2, 3-4 port numbering for 4 port files. Optionally, include ON to flip the port numbering when reading the s-parameter file.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<s_parameter_file> The name of the s-parameter file.

Example This example deconvolves the s-parameter file example.s2p with the waveform on channel 1.

```
myScope.WriteString ":CHANNEL1:ISIM:DEConvolve example.s2p"
```

:CHANnel<N>:ISIM:DELay

Command `:CHANnel<N>:ISIM:DElay {OFF | ON | TRIGger}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

Cables and fixtures, in addition to changing the waveform's shape, delay the signal in time. This can make it difficult to compare the simulated and measured waveforms because as you turn InfiniiSim on and off, the waveforms are not aligned in time. This may make one waveform or the other shift entirely off screen, for example.

When applying transfer functions, use the :CHANnel<N>:ISIM:DELay command to specify whether filter delay is removed or included:

- OFF — No waveform shift is applied.
- ON — The filter delay is included in the resultant waveform. This selection shifts the simulated waveform in time so the two waveforms are approximately aligned which aids in their comparison.
- TRIGger — Applies the trigger's jitter error correction after the de-embedding filter has been applied.

Consult the InfiniiSim User's Guide in the Manuals section of the GUI help system for more information.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example applies the transfer function delay in the resultant waveform.

```
myScope.WriteString ":CHANnel1:ISIM:DELay ON"
```

Query `:CHANnel<N>:ISIM:DELay?`

The :CHANnel<N>:ISIM:DELay? query returns the current state of the transfer function delay feature on the corresponding input channel.

Returned Format `[CHANnel<N>:ISIM:DELay] {OFF | ON | TRIG}<NL>`

Example This example puts whether or not the transfer function delay is included in the resultant waveform for channel 1 in the string variable, strDelay. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:DELay?"
strDelay = myScope.ReadString
Debug.Print strDelay
```

See Also

- ":CHANnel<N>:ISIM:NORMalize" on page 215

:CHANnel<N>:ISIM:NORMalize

Command `:CHANnel<N>:ISIM:NORMalize {{ON | 1} | {OFF | 0}}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:NORMalize command activates or deactivates the "Normalize Gain" option. The InfiniiSim normalize gain option removes any DC gain of the transfer function and can be used when modeling probes.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example turns on the InfiniiSim normalize gain option for channel 1.

```
myScope.WriteString ":CHANnel1:ISIM:NORMalize ON"
```

Query `:CHANnel<N>:ISIM:NORMalize?`

The :CHANnel<N>:ISIM:NORMalize? query returns the current state of the corresponding channel's InfiniiSim normalize gain option.

Returned Format `[CHANnel<N>:ISIM:NORMalize] {1 | 0}<NL>`

Example This example puts the current InfiniiSim normalize gain state for channel 1 in the string variable, varNormalizeGain. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANnel1:ISIM:NORMalize?"
varNormalizeGain = myScope.ReadNumber
Debug.Print FormatNumber(varLimit, 0)
```

See Also

- ":CHANnel<N>:ISIM:DELay" on page 213

:CHANnel<N>:ISIM:PEXTraction

Command `:CHANnel<N>:ISIM:PEXTraction {P12 | P32 | P34 | P14 | DIFFerential | COMMonmode}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:PEXTraction command selects the InfiniiSim port extraction. The selections are:

- P12 — Use ports 1 -> 2, only valid for channels 1 and 2.
- P32 — Use ports 3 -> 2, only valid for channels 1 and 2.
- P34 — Use ports 3 -> 4, only valid for channels 3 and 4.
- P14 — Use ports 1 -> 4, only valid for channels 3 and 4.
- DIFFerential — valid for all channels.
- COMMonmode — valid for all channels.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example selects the channel 1 InfiniiSim differential port extraction.

```
myScope.WriteString ":CHANNEL1:ISIM:PEXTraction DIFFerential"
```

Query `:CHANnel<N>:ISIM:PEXTraction?`

The :CHANnel<N>:ISIM:PEXTraction? query returns the current InfiniiSim port extraction selection.

Returned Format `[CHANnel<N>:ISIM:PEXTraction] {P12 | P32 | P34 | P14 | DIFF | COMM}<NL>`

Example This example puts the current InfiniiSim port extraction selection for channel 1 in the string variable, strMode. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:ISIM:PEXTraction?"
```

```
strMode = myScope.ReadString
Debug.Print strMode
```

:CHANnel<N>:ISIM:SPAN

Command `:CHANnel<N>:ISIM:SPAN <max_time_span>`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:SPAN command sets the maximum time span control in the InfiniiSim Setup dialog box.

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<max_time_span> A real number.

Example This example sets the maximum time span control to 100e-9.

```
myScope.WriteString ":CHANNEL1:ISIM:SPAN 100e-9"
```

Query `:CHANnel<N>:ISIM:SPAN?`

The :CHANnel<N>:ISIM:SPAN? query returns the current InfiniiSim filter maximum time span on the corresponding input channel.

Returned Format `[CHANnel<N>:ISIM:SPAN] <max_time_span><NL>`

Example This example puts the InfiniiSim filter's maximum time span value in the variable varTspan. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:ISIM:SPAN?"
varTspan = myScope.ReadNumber
Debug.Print FormatNumber(varTspan, 0)
```

:CHANnel<N>:ISIM:STATe

Command `:CHANnel<N>:ISIM:STATe {OFF | PORT2 | PORT4 | PORT41}`

NOTE This CHANnel command only applies if you have purchased the InfiniiSim software application.

The :CHANnel<N>:ISIM:STATe command turns InfiniiSim on or off and sets whether 2 port, 4 port (Channels 1&3), or 4 port (Channel 1) mode is being used (if it is turned on).

NOTE You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

Example This example turns on InfiniiSim for channel 1 and puts it in 2 port mode.

```
myScope.WriteString ":CHANNEL1:ISIM:STATe PORT2"
```

Query `:CHANnel<N>:ISIM:STATe?`

The :CHANnel<N>:ISIM:STATe? query returns the current state of InfiniiSim on the corresponding input channel.

Returned Format `[CHANnel<N>:ISIM:STATe] {OFF | PORT2 | PORT4 | PORT41}<NL>`

Example This example puts the current InfiniiSim state for channel 1 in the string variable, strMode. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:ISIM:STATe?"
strMode = myScope.ReadString
Debug.Print strMode
```

:CHANnel<N>:LABel

Command `:CHANnel<N>:LABel <string>`

The :CHANnel<N>:LABel command sets the channel label to the quoted string.

NOTE

You can specify differential and/or common mode channels using the following convention. If you have differential or common mode channels enabled (using either the :CHANnel<N>:DIFFerential or :CHANnel<N>:COMMonmode commands) then:

- :CHANnel1 would refer to the Channel 1 - Channel 3 differential channel
- :CHANnel2 would refer to the Channel 2 - Channel 4 differential channel
- :CHANnel3 would refer to the Channel 1 + Channel 3 common mode channel
- :CHANnel4 would refer to the Channel 2 + Channel 4 common mode channel

<N> An integer, 1-4.

<string> A series of 16 or less characters as a quoted ASCII string

Example This example sets the channel 1 label to Data.

```
myScope.WriteString ":CHANNEL1:LABel ""Data"""
```

Query `:CHANnel<N>:LABel?`

The :CHANnel<N>:LABel? query returns the label of the specified channel.

Returned Format `[CHANnel<N>:LABel] <string><NL>`

:CHANnel<N>:OFFSet

Command `:CHANnel<N>:OFFSet <offset_value>`

The :CHANnel<N>:OFFSet command sets the vertical value that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

<N> An integer, 1-4.

<offset_value> A real number for the offset value at center screen. Usually expressed in volts, but it can also be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:UNITs command or the CHANnel<N>:PROBe:EXTernal:UNITs command.

Example This example sets the offset for channel 1 to 0.125 in the current measurement units:

```
myScope.WriteString ":CHANNEL1:OFFSET 125E-3"
```

Query `:CHANnel<N>:OFFSet?`

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

Returned Format `[CHANnel<N>:OFFSet] <offset_value><NL>`

Example This example places the offset value of the specified channel in the variable, varOffset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:OFFSET?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, "Scientific")
```

:CHANnel<N>:PROBe

Command `:CHANnel<N>:PROBe <attenuation_factor>[,{RATio | DECibel}]`

The :CHANnel<N>:PROBe command sets the probe attenuation factor and the units (ratio or decibels) for the probe attenuation factor for a user-defined probe.

The DECibel and RATio parameters also set the "mode" for the probe attenuation. These parameters, along with attenuation factor, determine the scaling of the display and affect automatic measurements and trigger levels.

This mode also determines the units (ratio or decibels) that may be used for a subsequent command.

<N> An integer, 1-4

<attenuation_factor> A real number from 0.0001 to 1000 for the RATio attenuation units or from -80 dB to 60 dB for the DECibel attenuation units.

Example This example sets the probe attenuation factor for a 10:1 probe on channel 1 in ratio units.

```
myScope.WriteString ":CHANNEL1:PROBE 10,RAT"
```

Query `:CHANnel<N>:PROBe?`

The :CHANnel<N>:PROBe? query returns the current probe attenuation setting and units for the selected channel.

Returned Format `[:CHANnel<N>:PROBe] <attenuation>,{RATio | DECibel}<NL>`

Example This example places the current attenuation setting for channel 1 in the string variable, strAtten, then the program prints the contents.

```
Dim strAtten As String   ' Dimension variable.
myScope.WriteString ":CHANNEL1:PROBE?"
strAtten = myScope.ReadString
Debug.Print strAtten
```

If you use a string variable, the query returns the attenuation value and the factor (decibel or ratio). If you use an integer variable, the query returns the attenuation value. You must then read the attenuation units into a string variable.

:CHANnel<N>:PROBe:ACCAL

Command `:CHANnel<N>:PROBe:ACCAL {AUTO | OFF | PRECprobe}`

The :CHANnel<N>:PROBe:ACCAL command sets the type of AC response probe calibration to use:

- OFF — no AC response probe calibration is used.
- AUTO — the AC response probe calibration is based on the type of probe being used and its general characteristics.
- PRECprobe — PrecisionProbe or PrecisionCable probe calibration is used.

NOTE You are not able to start a PrecisionProbe or PrecisionCable calibration using remote SCPI commands. However, you can enter SCPI commands to use the results of calibrations performed using the front panel wizards.

<N> An integer, 1-4.

Example This example chooses the PrecisionProbe or PrecisionCable AC response calibration for the probe on channel 1.

```
myScope.WriteString ":CHANNEL1:PROBE:ACCAL PRECprobe"
```

Query `:CHANnel<N>:PROBe:ACCAL?`

The :CHANnel<N>:PROBe:ACCAL? query returns the AC response probe calibration setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:ACCAL] {AUTO | OFF | PREC}<NL>`

See Also

- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 243
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 242
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 244
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 241

:CHANnel<N>:PROBe:ATTenuation

Command `:CHANnel<N>:PROBe:ATTenuation {DIV1 | DIV10}`

NOTE This command is only valid for the 1154A probe.

The :CHANnel<N>:PROBe:ATTenuation command sets the 1154A probe's input amplifier attenuation. If the 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example sets the probe attenuation for channel 1 to divide by 10.

```
myScope.WriteString ":CHANNEL1:PROBE:ATTENUATION DIV10"
```

Query `:CHANnel<N>:PROBe:ATTenuation?`

The :CHANnel<N>:PROBe:ATTenuation? query returns the current 1154A probe input amplifier attenuation setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:ATTenuation] {DIV1 | DIV10}<NL>`

:CHANnel<N>:PROBe:AUTozero

Command `:CHANnel<N>:PROBe:AUTozero`

NOTE This command is currently only valid for the N2893A probe.

The :CHANnel<N>:PROBe:AUTozero command initiates the N2893A probe's auto degauss/ offset cal.

If the N2893A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example performs an auto zero operation for the probe on channel 1.

```
myScope.WriteString ":CHANnel1:PROBe:AUTozero"
```

:CHANnel<N>:PROBe:COUPling

Command `:CHANnel<N>:PROBe:COUPling {DC | AC}`

The :CHANnel<N>:PROBe:COUPling command sets the coupling to either AC or DC.

<N> An integer, 1-4.

Example This example sets the probe coupling for channel 1 to AC.

```
myScope.WriteString ":CHANNEL1:PROBE:COUPling AC"
```

Query `:CHANnel<N>:PROBe:COUPling?`

The :CHANnel<N>:PROBe:COUPling? query returns the current probe coupling setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:COUPling] {DC | AC}<NL>`

:CHANnel<N>:PROBe:EADapter

Command `:CHANnel<N>:PROBe:EADapter {NONE | DIV10 | DIV20 | DIV100}`

NOTE This command is valid only for the 1153A, 1154A, and 1159A probes.

The :CHANnel<N>:PROBe:EADapter command sets the probe external adapter control. The 1153A, 1154A, and 1159A probes have external adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the EADapter command to set the external adapter control to match the adapter connected to your probe as follows.

Parameter	Description
NONE	Use this setting when there is no adapter connected to the end of your probe.
DIV10	Use this setting when you have a divide by 10 adapter connected to the end of your probe.
DIV20	Use this setting when you have a divide by 20 adapter connected to the end of your probe. (1159A)
DIV100	Use this setting when you have a divide by 100 adapter connected to the end of your probe.(1153A only)

If an 1153A, 1154A, or 1159A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example sets the external adapter for channel 1 to divide by 10:

```
myScope.WriteString ":CHANNEL1:PROBE:EADAPTER DIV10"
```

Query `:CHANnel<N>:PROBe:EADapter?`

The :CHANnel<N>:PROBe:EADapter? query returns the current external adapter value for the specified channel.

Returned Format `[CHANnel<N>:PROBe:EDApter] {NONE | DIV10 | DIV20 | DIV100}<NL>`

Example This example places the external adapter value of the specified channel in the string variable, strAdapter, then prints the contents of the variable to the computer's screen.

```
Dim strAdapter As String   'Dimension variable
myScope.WriteString ":CHANNEL1:PROBE:EADAPTER?"
strAdapter = myScope.ReadString
Debug.Print strAdapter
```

:CHANnel<N>:PROBe:ECOupling

Command `:CHANnel<N>:PROBe:ECOupling {NONE | AC}`

NOTE This command is valid only for the 1153A, 1154A, and 1159A probes.

The :CHANnel<N>:PROBe:ECOupling command sets the probe external coupling adapter control. The 1154A and 1159A probes have external coupling adapters that you can attach to the end of your probe. When you attach one of these adapters, you should use the ECOupling command to set the external coupling adapter control to match the adapter connected to your probe as follows.

Parameter	Description
NONE	Use this setting when there is no adapter connected to the end of your probe.
AC	Use this setting when you have an ac coupling adapter connected to the end of your probe.

If an 1153A, 1154A, or 1159A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example sets the external coupling adapter for channel 1 to ac:

```
myScope.WriteString ":CHANNEL1:PROBE:ECOUPLING AC"
```

Query `:CHANnel<N>:PROBe:ECOupling?`

The :CHANnel<N>:PROBe:ECoupling? query returns the current external adapter coupling value for the specified channel.

Returned Format `[CHANnel<N>:PROBe:ECOupling] {NONE | AC}<NL>`

Example This example places the external coupling adapter value of the specified channel in the string variable, strAdapter, then prints the contents of the variable to the computer's screen.

```
Dim strAdapter As String   ' Dimension variable.
myScope.WriteString ":CHANNEL1:PROBE:ECOUPLING?"
strAdapter = myScope.ReadString
Debug.Print strAdapter
```

:CHANnel<N>:PROBe:EXTernal

Command `:CHANnel<N>:PROBe:EXTernal {{ON | 1} | {OFF | 0}}`

The :CHANnel<N>:PROBe:EXTernal command sets the external probe mode to on or off.

<N> An integer, 1-4.

Example This example sets channel 1 external probe mode to on.

```
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
```

Query `:CHANnel<N>:PROBe:EXTernal?`

The :CHANnel<N>:PROBe:EXTernal? query returns the current external probe mode for the specified channel.

Returned Format `[:CHANnel<N>:PROBe:EXTernal] {1 | 0}<NL>`

Example This example places the current setting of the external probe mode on channel 1 in the variable varMode, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL?"
varMode = myScope.ReadNumber
Debug.Print FormatNumber(varMode, 0)
```

:CHANnel<N>:PROBe:EXTernal:GAIN

Command `:CHANnel<N>:PROBe:EXTernal:GAIN <gain_factor>[,{RATio | DECibel}]`

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:GAIN command sets the probe external scaling gain factor and, optionally, the units for the probe gain factor. The reference factors that are used for scaling the display are changed with this command, and affect automatic measurements and trigger levels.

The RATio or DECibel also sets the mode for the probe attenuation and also determines the units that may be used for a subsequent command. For example, if you select RATio mode, then the attenuation factor must be given in ratio gain units. In DECibel mode, you can specify the units for the argument as "dB".

<N> An integer, 1-4.

<gain_factor> A real number from 0.001 to 10000 for the RATio gain units, or from -60 dB to 80 dB for the DECibel gain units.

Example This example sets the probe external scaling gain factor for channel 1 to 10.

```
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:GAIN 10,RATIO"
```

Query `:CHANnel<N>:PROBe:EXTernal:GAIN?`

The :CHANnel<N>:PROBe:EXTernal:GAIN? query returns the probe external gain setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:EXTernal:GAIN] <gain_factor><NL>`

Example This example places the external gain value of the probe on the specified channel in the variable, varGain, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:GAIN?"
varGain = myScope.ReadNumber
Debug.Print FormatNumber(varGain, 0)
```

:CHANnel<N>:PROBe:EXTernal:OFFSet

Command `:CHANnel<N>:PROBe:EXTernal:OFFSet <offset_value>`

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect.

The :CHANnel<N>:PROBe:EXTernal:OFFSet command sets the external vertical value for the probe that is represented at the center of the display for the selected channel. Offset parameters are probe and vertical scale dependent.

When using the 113xA series probes, the CHANnel<N>:PROBe:STYPe command determines how the offset is applied. When CHANnel<N>:PROBe:STYPe SINGle is selected, the :CHANnel<N>:PROBe:EXTernal:OFFset command changes the offset value of the probe amplifier. When CHANnel<N>:PROBe:STYPe DIFFerential is selected, the :CHANnel<N>:PROBe:EXTernal:OFFSet command changes the offset value of the channel amplifier.

<N> An integer, 1-4.

<offset_value> A real number for the offset value at center screen. Usually expressed in volts, but can be in other measurement units, such as amperes, if you have specified other units using the :CHANnel<N>:PROBe:EXTernal:UNITs command.

Example This example sets the external offset for the probe on channel 1 to 0.125 in the current measurement units:

```
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:OFFSET 125E-3"
```

Query `:CHANnel<N>:EXTernal:PROBe:OFFSet?`

The :CHANnel<N>:PROBe:EXTernal:OFFSet? query returns the current external offset value for the probe on the specified channel.

Returned Format `[CHANnel<N>:PROBe:EXTernal:OFFSet] <offset_value><NL>`

Example This example places the external offset value of the probe on the specified channel in the variable, Offset, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:OFFSET?"
varOffset = myScope.ReadNumber
Debug.Print FormatNumber(varOffset, 0)
```

:CHANnel<N>:PROBe:EXTernal:UNITs

Command `:CHANnel<N>:PROBe:EXTernal:UNITs {VOLT | AMPere | WATT | UNKNown}`

NOTE CHANnel<N>:PROBe:EXTernal command must be set to ON before issuing this command or query or this command will have no effect. UNITs can also be set using the CHANnel<N>:UNITs command.

The :CHANnel<N>:PROBe:EXTernal:UNITs command sets the probe external vertical units on the specified channel. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel probe external commands and channel commands (such as :CHANnel<N>:PROBe:EXTernal:OFFSet and :CHANnel<N>:RANGe). See the Probe Setup dialog box for more information.

<N> An integer, 1-4.

Example This example sets the external units for the probe on channel 1 to amperes.

```
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:UNITS AMPERE"
```

Query `:CHANnel<N>:PROBe:EXTernal:UNITs?`

The :CHANnel<N>:PROBe:EXTernal:UNITs? query returns the current external units setting for the probe on the specified channel.

Returned Format `[:CHANnel<N>:PROBe:EXTernal:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>`

Example This example places the external vertical units for the probe on the specified channel in the string variable, strUnits, then prints the contents of the variable to the computer's screen.

```
Dim strUnits As String
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL ON"
myScope.WriteString ":CHANNEL1:PROBE:EXTERNAL:UNITS?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

:CHANnel<N>:PROBe:GAIN

Command `:CHANnel<N>:PROBe:GAIN {X1 | X10}`

NOTE This command is valid only for the 1154A probe.

The :CHANnel<N>:PROBe:GAIN command sets the 1154A probe input amplifier gain.

If an 1154A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example sets the probe gain for channel 1 to times 10.

```
myScope.WriteString ":CHANNEL1:PROBE:GAIN X10"
```

Query `:CHANnel<N>:PROBe:GAIN?`

The :CHANnel<N>:PROBe:GAIN? query returns the current probe gain setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:GAIN] {X1 | X10}<NL>`

:CHANnel<N>:PROBe:HEAD:ADD

Command `:CHANnel<N>:PROBe:HEAD:ADD "head", ["label"]`

The :CHANnel<N>:PROBe:HEAD:ADD command adds an entry to the list of probe heads.

<N> An integer, 1-4.

"head" A quoted string matching the probe head model such as "N5381A", "E2678A", etc.

"label" An optional quoted string for the head label.

Example This example adds the probe head N5381A to the list of probe heads for channel 1.

```
myScope.WriteString ":CHANNEL1:PROBE:HEAD:ADD ""N5381A"""
```

:CHANnel<N>:PROBe:HEAD:DELete ALL

Command `:CHANnel<N>:PROBe:HEAD:DELete ALL`

The :CHANnel<N>:PROBe:HEAD:DELete ALL command deletes all the nodes in the list of probe heads except for one default probe head which remains after this command is executed.

<N> An integer, 1-4.

Example This example deletes the entire list of probe heads for channel 1 except for the default head.

```
myScope.WriteString ":CHANNEL1:PROBE:HEAD:DELete ALL"
```

:CHANnel<N>:PROBe:HEAD:SELect

Command `:CHANnel<N>:PROBe:HEAD:SELect {<int> | <quoted_label_string>}`

The :CHANnel<N>:PROBe:HEAD:SELect command selects the probe head being used from a list of possible probe head choices. You can select by the position number in the list of probe heads, or you can select by the label given when the probe head was added.

<N> An integer, 1-4.

<int> Specifies the number of the head (or position) in the configure list. The entry at the top of the list starts at 1.

<quoted_label_string> Specifies the label of the probe head given with the :CHANnel<N>:PROBe:HEAD:ADD command.

Example This example add a couple of probe heads to the list then selects the probe head using a number and a label.

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:ADD 'N5445A:B1.5-2.5S'"
myScope.WriteString ":CHANnel1:PROBe:HEAD:ADD 'N5444A:2.92','foo'"
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect 1"
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect 'foo'"
```

Query `:CHANnel<N>:PROBe:HEAD:SELect? {MODel | LABel}`

The :CHANnel<N>:PROBe:HEAD:SELect? query returns a SCPI formatted string of the selected probe head. Optional parameters are:

- MODel — Returns the model of the probe head.
- LABel — Returns the label of the probe head. This is the same label given with the :CHANnel<N>:PROBe:HEAD:ADD command and that can also be used with the SELect command.

If no parameter is specified, the MODel format is returned.

Example This example shows a few queries of the channel 1 probe head selection.

```
Dim strProbeHead As String
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect?"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "N5444A:2.92".
myScope.WriteString ":CHANnel1:PROBe:HEAD:SELect? LABel"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "foo".
myScope.WriteString ":CHANnel2:PROBe:HEAD:SELect? MODel"
strProbeHead = myScope.ReadString
Debug.Print strProbeHead   ' Prints "N5444A:2.92".
```

See Also

- ":CHANnel<N>:PROBe:HEAD:ADD" on page 235

:CHANnel<N>:PROBe:HEAD:VTERm

Command

```
:CHANnel<N>:PROBe:HEAD:VTERm {FLOating | EXTernal
                              | {INTernal,<voltage>}}
```

The :CHANnel<N>:PROBe:HEAD:VTERm command sets the termination voltage for the N5444A probe head.

<N> An integer, 1-4.

<voltage> A real number for the internal termination voltage setting.

Example To set an internal termination voltage of -1.0 V:

```
myScope.WriteString ":CHANnel1:PROBe:HEAD:VTERm INTernal,-1.0"
```

Query

```
:CHANnel<N>:PROBe:HEAD:VTERm?
```

The :CHANnel<N>:PROBe:HEAD:VTERm? query returns the termination voltage setting.

Returned Format

```
[:CHANnel<N>:PROBe:HEAD:VTERm] {FLO | EXT | {INT,<voltage>}}<NL>
```

:CHANnel<N>:PROBe:ID?

Query `:CHANnel<N>:PROBe:ID?`

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

<N> An integer, 1-4.

Returned Format [:CHANnel<N>:PROBe:ID] <probe_id>

<probe_id> A string of alphanumeric characters. Some of the possible returned values are:

1131A	1132A	1134A
1152A	1154A	1156A
1157A	1158A	1159A
1163A	1168A	1169A
AutoProbe	E2621A	E2622A
E2695A	E2697A	N5380A
N5381A	N5382A	E2695A
No Probe	Unknown	User Defined Probe

Example This example reports the probe type connected to channel 1, if one is connected.

```
myScope.WriteString ":CHANNEL1:PROBE:ID?"
```

:CHANnel<N>:PROBe:MODE

Command `:CHANnel<N>:PROBe:MODE {DIFF | SEA | SEB | CM}`

NOTE This command is currently only valid for the N2750A probe.

The :CHANnel<N>:PROBe:MODE command sets the N2750A probe's InfiniiMode configuration.

If the N2750A probe is not connected to the channel you will get a settings conflict error.

<N> An integer, 1-4.

Example This example sets the probe InfiniiMode for channel 1 to common mode.

```
myScope.WriteString ":CHANNEL1:PROBE:MODE CM"
```

Query `:CHANnel<N>:PROBe:MODE?`

The :CHANnel<N>:PROBe:MODE? query returns the current N2750A probe InfiniiMode setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:MODE] {DIFF | SEA | SEB | CM}<NL>`

:CHANnel<N>:PROBe:PRECprobe:BANDwidth

Command

```
:CHANnel<N>:PROBe:PRECprobe:BANDwidth {AUTO | {MANual, <bandwidth>}
                                      | {BOOSt, <boost_dB>}}
```

The :CHANnel<N>:PROBe:PRECprobe:BANDwidth command specifies how the limit of PrecisionProbe or PrecisionCable correction/boosting is determined.

<N> An integer, 1-4.

AUTO PrecisionProbe or PrecisionCable normally sets the bandwidth to a value that has a small amount of boosting in the frequency response.

MANual, <bandwidth> Let you manually specify a bandwidth limit at which to stop applying correction.

BOOSt, <boost_dB> Lets you specify a dB limit at which to stop applying correction.

Example This example specifies that, for PrecisionProbe or PrecisionCable on channel 1, correction/boosting should stop being applied at a 3 dB limit.

```
myScope.WriteString ":CHANNEL1:PROBE:PRECprobe:BANDwidth BOOSt, 3"
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:BANDwidth?
```

The :CHANnel<N>:PROBe:PRECprobe:BANDwidth? query returns the current PrecisionProbe or PrecisionCable corrected bandwidth setting for the selected channel.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:BANDwidth] {AUTO | {MANual, <bandwidth>}
                                       | {BOOSt, <boost_dB>}}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 223
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 243
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 242
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 244

:CHANnel<N>:PROBe:PRECprobe:CALibration

Command `:CHANnel<N>:PROBe:PRECprobe:CALibration <cal_string>[,<cal_string2]`

The :CHANnel<N>:PROBe:PRECprobe:CALibration command specifies the name of the PrecisionProbe or PrecisionCable calibration to use for the specified channel and probe.

<N> An integer, 1-4.

<cal_string>[,<cal_string2>] A quoted string that is the name of the PrecisionProbe or Precision Cable calibration. The SMA probe heads can use two independent calibration files.

Example This example says to use the PrecisionProbe or PrecisionCable calibration named "2-8-2" for channel 1.

```
myScope.WriteString ":CHANNEL1:PROBE:PRECprobe:CALibration "2-8-2""
```

Query `:CHANnel<N>:PROBe:PRECprobe:CALibration?`

The :CHANnel<N>:PROBe:PRECprobe:CALibration? query returns the currently specified name for the selected channel's PrecisionProbe or PrecisionCable calibration.

Returned Format `[:CHANnel<N>:PROBe:PRECprobe:CALibration] <cal_string>[,<cal_string2]<NL>`

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 223
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 243
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 244
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 241

:CHANnel<N>:PROBe:PRECprobe:MODE

Command `:CHANnel<N>:PROBe:PRECprobe:MODE {PROBe | CABLe}`

The :CHANnel<N>:PROBe:PRECprobe:MODE command chooses between PrecisionProbe or PrecisionCable AC response probe calibration.

<N> An integer, 1-4.

Example This example chooses PrecisionProbe calibration for the probe on channel 1.

```
myScope.WriteString ":CHANNEL1:PROBE:PRECprobe:MODE PROBe"
```

Query `:CHANnel<N>:PROBe:PRECprobe:MODE?`

The :CHANnel<N>:PROBe:PRECprobe:MODE? query returns the current PrecisionProbe/PrecisionCable selection for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:PRECprobe:MODE] {PROBe | CABLe}<NL>`

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 223
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 242
- ":CHANnel<N>:PROBe:PRECprobe:ZSRC" on page 244
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 241

:CHANnel<N>:PROBe:PRECprobe:ZSRC

Command

```
:CHANnel<N>:PROBe:PRECprobe:ZSRC {VIN | {VSRC, <impedance>}
                                 | {VSRC, <file_string>}}
```

The :CHANnel<N>:PROBe:PRECprobe:ZSRC command specifies how PrecisionProbe characterizes the time domain and frequency domain response.

<N> An integer, 1-4.

VIN Selects the VOut/Vin probe transfer function (which characterizes the output of the probe as a function of the input at the probe tips).

Defining the response this way lets you evaluate the probe's accuracy in reproducing the actual signal present in your system with the probe attached. This correction is what you would see with a real band limited probe that has finite input impedance. PrecisionProbe corrects the "VOut/Vin" response to be flat with frequency and phase to your defined bandwidth limit. It does not de-embed the loading effects of the probe. (Agilent's probe corrections are typically defined using Vout/Vin.)

VSRC, <impedance> Selects the VOut/VSrc estimate of probed system response (which corrects the probe as "what would be there if the probe were not present"), and specifies a constant (Zo/2) value (in ohms) as the system source impedance.

One drawback of defining the probe's response in this manner is that if the probe's loading causes your circuit to lose some timing or amplitude margin, you probably want to know that when you make a measurement. VOut/VSource compensation will hide these effects from you. However, this method can be effective if probing at the transmitter.

VSRC, <file_string> Selects the VOut/VSrc estimate of probed system response (which corrects the probe as "what would be there if the probe were not present"), and names an S-parameter file whose S11 is used to specify the system source impedance.

Example This example, for channel 1, tells PrecisionProbe to use the VOut/VSrc characterization and to get the system source impedance from S11 in the "foo.s2p" S-parameter file.

```
myScope.WriteString ":CHANNEL1:PROBE:PRECprobe:ZSRC VSRC, "foo.s2p""
```

Query

```
:CHANnel<N>:PROBe:PRECprobe:ZSRC?
```

The :CHANnel<N>:PROBe:PRECprobe:ZSRC? query returns the current settings for PrecisionProbe time domain and frequency domain response characterization.

Returned Format

```
[:CHANnel<N>:PROBe:PRECprobe:ZSRC] {VIN | {VSRC, <impedance>}
                                   | {VSRC, <file_string>}}<NL>
```

See Also

- ":CHANnel<N>:PROBe:ACCAL" on page 223
- ":CHANnel<N>:PROBe:PRECprobe:MODE" on page 243
- ":CHANnel<N>:PROBe:PRECprobe:CALibration" on page 242
- ":CHANnel<N>:PROBe:PRECprobe:BANDwidth" on page 241

:CHANnel<N>:PROBe:SKEW

Command `:CHANnel<N>:PROBe:SKEW <skew_value>`

The :CHANnel<N>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. You can use the oscilloscope's probe skew control to remove timing differences between probes or cables on different channels.

<N> An integer, 1-4.

<skew_value> A real number for the skew value, in the range -1 ms to +1 ms.

Example This example sets the probe skew for channel 1 to 10 µs.

```
myScope.WriteString ":CHANNEL1:PROBE:SKEW 10E-6"
```

Query `:CHANnel<N>:PROBe:SKEW?`

The :CHANnel<N>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:SKEW] <skew_value><NL>`

:CHANnel<N>:PROBe:STYPe

Command `:CHANnel<N>:PROBe:STYPe {DIFFerential | SINGle}`

NOTE This command is valid only for the 113xA series probes, 1168A probe, and 1169A probe.

The :CHANnel<N>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA series probes, 1168A probe, and 1169A probe. This setting determines how offset is applied.

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

<N> An integer, 1-4.

Example This example sets the probe mode to single-ended.

```
myScope.WriteString ":CHANNEL1:PROBE:STYPE SINGLE"
```

Query `:CHANnel<N>:PROBe:STYPe?`

The :CHANnel<N>:PROBe:STYPe? query returns the current probe mode setting for the selected channel.

Returned Format `[:CHANnel<N>:PROBe:STYPe] {DIFFerential | SINGle}<NL>`

:CHANnel<N>:RANGe

Command `:CHANnel<N>:RANGe <range_value>`

The :CHANnel<N>:RANGe command defines the full-scale vertical axis of the selected channel. It sets up acquisition and display hardware to display the waveform at a given range scale. The values represent the full-scale deflection factor of the vertical axis in volts. These values change as the probe attenuation factor is changed.

<N> An integer, 1-4.

<range_value> A real number for the full-scale voltage of the specified channel number.

Example This example sets the full-scale range for channel 1 to 500 mV.

```
myScope.WriteString ":CHANNEL1:RANGE 500E-3"
```

Query `:CHANnel<N>:RANGe?`

The :CHANnel<N>:RANGe? query returns the current full-scale vertical axis setting for the selected channel.

Returned Format `[:CHANnel<N>:RANGe]<range_value><NL>`

Example This example places the current range value in the number variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":CHANNEL1:RANGE?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:CHANnel<N>:SCALe

Command `:CHANnel<N>:SCALe <scale_value>`

The :CHANnel<N>:SCALe command sets the vertical scale, or units per division, of the selected channel. This command is the same as the front-panel channel scale.

<N> An integer, 1-4.

<scale_value> A real number for the vertical scale of the channel in units per division.

Example This example sets the scale value for channel 1 to 500 mV/div.

```
myScope.WriteString ":CHANNEL1:SCALE 500E-3"
```

Query `:CHANnel<N>:SCALe?`

The :CHANnel<N>:SCALe? query returns the current scale setting for the specified channel.

Returned Format `[:CHANnel<N>:SCALe] <scale_value><NL>`

Example This example places the current scale value in the number variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":CHANNEL1:SCALE?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:CHANnel<N>:UNITs

Command `:CHANnel<N>:UNITs {VOLT | AMPere | WATT | UNKNown}`

NOTE UNITs can also be set using the CHANnel<N>:PROBe:EXTernal:UNITs command when CHANnel<N>:PROBe:EXTernal command has been set to ON.

The :CHANnel<N>:UNITs command sets the vertical units. You can specify Y-axis units of VOLTs, AMPs, WATTs, or UNKNown. The units are implied for other pertinent channel commands (such as :CHANnel<N>:RANGe and :CHANnel<N>:OFFSet). See the Probe Setup dialog box for more information.

<N> An integer, 1-4.

Example This example sets the units for channel 1 to amperes.

```
myScope.WriteString ":CHANNEL1:UNITS AMPERE"
```

Query `:CHANnel<N>:UNITs?`

The :CHANnel<N>:UNITs? query returns the current units setting for the specified channel.

Returned Format `[:CHANnel<N>:UNITs] {VOLT | AMPere | WATT | UNKNown}<NL>`

Example This example places the vertical units for the specified channel in the string variable, strUnits, then prints the contents of the variable to the computer's screen.

```
Dim strUnits As String
myScope.WriteString ":CHANNEL1:UNITS?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

13 Common Commands

*CLS 253
*ESE 254
*ESR? 256
*IDN? 257
*LRN? 258
*OPC 260
*OPT? 261
*PSC 264
*RCL 265
*RST 266
*SAV 267
*SRE 268
*STB? 270
*TRG 272
*TST? 273
*WAI 274

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments. Common commands can be received and processed by the oscilloscope, whether they are sent over the remote interface as separate program messages or within other program messages.

Receiving Common Commands

Common commands can be received and processed by the oscilloscope, whether they are sent over the remote interface as separate program messages or within other program messages. If a subsystem is currently selected and a common command is received by the oscilloscope, the oscilloscope remains in the selected subsystem. For example, if the program message

```
"ACQUIRE:AVERAGE ON;*CLS;COUNT 1024"
```

is received by the oscilloscope, the oscilloscope sets the acquire type, clears the status information, then sets the number of averages without leaving the selected subsystem.

NOTE

Headers and Common Commands.

Headers are not prepended to common commands.

Status Registers

The following two status registers used by common commands have an enable (mask) register. By setting bits in the enable register, you can select the status information for use. Refer to the chapter, "Status Reporting," for a complete discussion of status.

Table 16 Status and Enable Registers

Status Register	Enable Register
Event Status Register	Event Status Enable Register
Status Byte Register	Service Request Enable Register

*CLS

(Clear Status)

Command `*CLS`

The *CLS command clears all status and error registers.

Example This example clears the status data structures of the oscilloscope.

```
myScope.WriteString "*CLS"
```

See Also Refer to the "Status Reporting" chapter for a complete discussion of status.

*ESE

(Event Status Enable)

Command `*ESE <mask>`

The *ESE command sets the Standard Event Status Enable Register bits.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Standard Event Status Register as shown in Table 17.

Example This example enables the User Request (URQ) bit of the Standard Event Status Enable Register. When this bit is enabled and a front-panel key is pressed, the Event Summary bit (ESB) in the Status Byte Register is also set.

```
myScope.WriteString "*ESE 64"
```

Query `*ESE?`

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

Returned Format `<mask><NL>`

<mask> An integer, +0 to +255 (the plus sign is also returned), representing a mask value for the bits enabled in the Standard Event Status Register as shown in Table 17.

Example This example places the current contents of the Standard Event Status Enable Register in the numeric variable, varEvent. The value of the variable is printed on the computer's screen.

```
myScope.WriteString "*ESE?"
varEvent = myScope.ReadNumber
Debug.Print FormatNumber(varEvent, 0)
```

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 "0" in the enable register disables the corresponding bit.

Table 17 Standard Event Status Enable Register Bits

Bit	Weight	Enables	Definition
7	128	PON - Power On	Indicates power is turned on.
6	64		Not Used. Permanently set to zero.
5	32	CME - Command Error	Indicates whether the parser detected an error.

Table 17 Standard Event Status Enable Register Bits (continued)

Bit	Weight	Enables	Definition
4	16	EXE - Execution Error	Indicates whether a parameter was out of range, or was inconsistent with the current settings.
3	8	DDE - Device Dependent Error	Indicates whether the device was unable to complete an operation for device-dependent reasons.
2	4	QYE - Query Error	Indicates if the protocol for queries has been violated.
1	2	RQC - Request Control	Indicates whether the device is requesting control.
0	1	OPC - Operation Complete	Indicates whether the device has completed all pending operations.

See Also Refer to Chapter 6, "Status Reporting," starting on page 105 for a complete discussion of status.

*ESR?

(Event Status Register)

Query `*ESR?`

The *ESR? query returns the contents of the Standard Event Status Register. Reading this register clears the Standard Event Status Register, as does a *CLS.

Returned Format `<status><NL>`

<status> An integer, 0 to 255, representing the total bit weights of all bits that are high at the time you read the register.

Example This example places the current contents of the Standard Event Status Register in the numeric variable, varEvent, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*ESR?"
varEvent = myScope.ReadNumber
Debug.Print FormatNumber(varEvent, 0)
```

Table 18 lists each bit in the Event Status Register and the corresponding bit weights.

Table 18 Standard Event Status Register Bits

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
7	128	PON	1 = OFF to ON transition has occurred.
6	64		Not Used. Permanently set to zero.
5	32	CME	0 = no command errors. 1 = a command error has been detected.
4	16	EXE	0 = no execution error. 1 = an execution error has been detected.
3	8	DDE	0 = no device-dependent errors. 1 = a device-dependent error has been detected.
2	4	QYE	0 = no query errors. 1 = a query error has been detected.
1	2	RQC	0 = request control - NOT used - always 0.
0	1	OPC	0 = operation is not complete. 1 = operation is complete.

*IDN?

(Identification Number)

Query `*IDN?`

The *IDN? query returns the company name, oscilloscope model number, serial number, and software version by returning this string:

```
Agilent Technologies,<Model #>,<USXXXXXXXX>,<Rev #>
[,<Options>]
```

<Model #> Specifies the model number of the oscilloscope.

<USXXXXXXXX> Specifies the serial number of the oscilloscope. The first four digits and letter are the serial prefix, which is the same for all identical oscilloscopes. The last five digits are the serial suffix, which is assigned sequentially, and is different for each oscilloscope.

<Rev #> Specifies the software version of the oscilloscope, and is the revision number.

<Options> Comma separated list of the installed options.

Returned Format `Agilent Technologies,DSO80804B,USXXXXXXXX,A.XX.XX`

Example This example places the oscilloscope's identification information in the string variable, strIdentify, then prints the identification information to the computer's screen.

```
Dim strIdentify As String   ' Dimension variable.
myScope.WriteString "*IDN?"
strIdentify = myScope.ReadString
Debug.Print strIdentify
```

*LRN?

(Learn)

Query `*LRN?`

The *LRN? query returns a block of data that contains the oscilloscope's current setup. You can store the oscilloscope's setup and send it back to the oscilloscope at a later time. This block of setup data should be sent to the oscilloscope just as it is. It works because of its embedded ":SYST:SET" header.

Returned Format `:SYST:SET<setup><NL>`

<setup> This is a definite-length, arbitrary block response specifying the current oscilloscope setup. The block size is subject to change with different firmware revisions.

Example This Python and PyVISA example saves the *LRN? string to a file and then restores the oscilloscope setup from the file.

```
# ********************************************************************
# Using the *LRN? string to save and restore the oscilloscope setup.
# ********************************************************************

# Import modules.
# ---------------------------------------------------------
import visa
import string
import sys

# ==========================================================
# Check for instrument errors:
# ==========================================================
def check_instrument_errors():

 while True:
  error_string = Infiniium.ask(":SYSTem:ERRor? STRing\n")
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    print "ERROR: %s." % error_string
    print "Exited because of error."
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print "ERROR: :SYSTem:ERRor? STRing returned nothing."
   print "Exited because of error."
   sys.exit(1)

# ==========================================================
# Main program:
# ==========================================================
```

```
Infiniium = visa.instrument("TCPIP0::130.29.71.191::inst0::INSTR")
Infiniium.timeout = 20
Infiniium.term_chars = ""
Infiniium.clear()

# Save oscilloscope setup.
sLearn = Infiniium.ask("*LRN?\n")
check_instrument_errors()

f = open("learn.stp", "wb")
f.write("%s\n" % sLearn)
f.close()
print "Learn string bytes saved: %d" % len(sLearn)

# Restore the default setup.
Infiniium.write("*RST\n")

# Set up oscilloscope by loading previously saved learn string.
sLearn = ""
f = open("learn.stp", "rb")
sLearn = f.read()
f.close()

Infiniium.write("%s\n" % sLearn)
check_instrument_errors()

print "Learn string bytes restored: %d" % len(sLearn)
```

See Also :SYSTem:SETup command and query. When HEADers is ON and LONGform is OFF, the :SYSTem:SETup command performs the same function as the *LRN? query. However, *LRN and SETup block setup data are not interchangeable.

NOTE

*LRN? Returns Prefix to Setup Block

The *LRN? query always returns :SYST:SET as a prefix to the setup block. The :SYSTem:HEADer command has no effect on this response.

*OPC

(Operation Complete)

Command `*OPC`

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

Example This example sets the operation complete bit in the Standard Event Status Register when the DIGitize operation is complete.

```
myScope.WriteString ":DIGITIZE CHANNEL1;*OPC"
```

Query `*OPC?`

The *OPC? query places an ASCII character "1" in the oscilloscope's output queue when all pending selected device operations have finished.

Returned Format `1<NL>`

Example This example places an ASCII character "1" in the oscilloscope's output queue when the AUToscale operation is complete. Then the value in the output queue is placed in the numeric variable var"varComplete."

```
myScope.WriteString ":AUTOSCALE;*OPC?"
varComplete = myScope.ReadNumber
Debug.Print FormatNumber(varComplete, 0)
```

The *OPC? query allows synchronization between the computer and the oscilloscope by using the message available (MAV) bit in the Status Byte or by reading the output queue. Unlike the *OPC command, the *OPC query does not affect the OPC Event bit in the Standard Event Status Register.

*OPT?

(Option)

Query `*OPT?`

The *OPT? query returns a string with a list of installed options. If no options are installed, the string will have a 0 as the first character.

The length of the returned string may increase as options become available in the future. Once implemented, an option name will be appended to the end of the returned string, delimited by a comma.

Returned Format `[002,EZP,EZJ,SDA,LSS,ABD,ABC,ABB,NRD,ERC,AIP,PCI1,ETH,DVI,HDM,B30,CAN,SA1,DDR]<NL>`

Table 19 Possible Installed Options and Descriptions

Installed Option	Description
AP2	(U7233A) DDR1 Compliance
B30	(N5416A) USB Compliance
CAN	(N5402A) CAN/FlexRay Protocols
CFL	(N8803A/B) Basic CAN/FlexRay Protocols
D12	(U7232B) Display Port Compliance
DD3	(U7231A) DDR3 Compliance
DD4	(N6462A) DDR4 Compliance
DDR	(N5413A/B) DDR2 Compliance
DEA	(N5465A-002) InfiniiSim Advanced
DEB	(N5465A-001) InfiniiSim Basic
DEQ	(N5461A) Equalization
DPT	(U7232A) Display Port Compliance
DRF	(N8807A) DIGRF4 Protocol
DVI	(N5394A) DVI Compliance
EKR	(N8815A) 10GBASE-KR Ethernet Protocol
EMC	(N6465A) eMMC Compliance
ETH	(N5392A) Gigabit Ethernet Compliance
ETN	(U7236A) 10G Ethernet Compliance
EZC	(N8813A) EZJIT Complete
EZJ	(E2681A) EZJIT
EZP	(N5400A) EZJIT Plus

Table 19 Possible Installed Options and Descriptions (continued)

Installed Option	Description
FBD	(N5409A) FB DIMM Compliance
FBR	(N5410A) Fibre Channel Compliance
GD5	(U7245A) GDDR5 Compliance
H14	(N5399B) HDMI + HEAC Compliance
HDM	(N5399A/B) HDMI Compliance
HSI	(U7248A) HSIC Compliance
LTP	(N8817A) JTAG Protocol
LP2	(N5413B) LPDDR2 Compliance
LPU	(N5413B) LPDDR2 Upgrade
LP3	(U7231B) LPDDR3 Compliance
L3U	(U7231B) LPDDR3 Upgrade
LSS	(N5391A/B) SPI/I2C Protocols
MHL	(N6460A) Mobile HD Link Compliance
MPI	(U7238A) MIPI D-PHY Compliance
MPH	(U7249A) MIPI M-PHY Compliance
MPP	(N8802A) MIPI D-PHY Protocol
MYC	(N5467A) User Defined App
PCI	(N5393B) PCI Express 1.0a Compliance
PC2	(N5393B) PCI Express 2.0 Compliance
PC3	(N5393C) PCI Express 3.0 Compliance
P3U	(N5393C) PCIE 3.0 Upgrade
PEP	(N5463A/B) PCI Express Protocol
PRN	(N2809A) PrecisionProbe
PWR	(U1882A) Power
QPI	(U7241A) QPI Compliance
RSP	(N5462A/B) RS232/UART Protocol
S6G	(N5412B) SAS 6G Compliance
S6U	(N5411B) SATA 3 Compliance Upgrade
SA1	(N5411A) SATA 1 Compliance
SA6	(N5411B) SATA 3 Compliance
SAS	(N5412A) SAS Compliance

Table 19 Possible Installed Options and Descriptions (continued)

Installed Option	Description
SDA	(E2688A, N5384A) Serial Data Analysis
SDC	(U7246A) SD Card Compliance
SSU	(N5412B) SAS 6G Upgrade
STP	(N8801A) SATA/SAS Protocol
SVD	(N8812A) SVID Protocol
SWT	(N5414A/B, N5415B) InfiniiScan
TBL	(N6463A) Thunderbolt Compliance
U3P	(N5464A/B) USB 3.0 Protocol
UDF	(N5430A) User Def Fn
UH2	Ultra HS2 SD Compliance
U23	(U7243A-003) Upgrade from USB 2.0 to USB 3.0 Advanced
US3	(U7243A-001) USB 3.0 Compliance Only (Basic)
USA	(U7243A-002) USB 2.0 and 3.0 Advanced
USU	(U7243A-004) Upgrade from USB 3.0 Compliance Only to USB 3.0 Advanced
USP	(N5464A/B) USB 2.0 Protocol
VSA	UWB VSA
WUB	(U7239A) Wireless USB Compliance
XAI	(N5431A) XAUI Compliance

Example This example places all options into the string variable, strOptions, then prints the option name to the computer's screen.

```
Dim strOptions As String
myScope.WriteString "*OPT?"
strOptions = myScope.ReadString
Debug.Print strOptions
```

*PSC

(Power-on Status Clear)

Command `*PSC {{ON|1} | {OFF|0}}`

The *PSC command determines whether or not the SRQ line is set upon the completion of the oscilloscope's boot process. When the *PSC flag is set to 1, the Power On (PON) bit of the Standard Event Status Register is 0 during the boot process. When the *PSC flag is set to 0, the PON bit is set to a 1 during the boot process.

When the *PSC flag is set to 0, the Standard Event Status Enable Register must be set to 128 decimal and the Service Request Enable Register must be set to 32 decimal. This allows the Power On (PON) bit to set the SRQ line when the oscilloscope is ready to receive commands.

NOTE If you are using a LAN interface rather than a GPIB interface, it is not possible to receive the SRQ during the boot process.

Example This example sets the *PSC flag to 0 which sets the SRQ line during the boot process.

```
myScope.WriteString "*PSC 0;*SRE 32;*ESE 128"
```

Query The *PSC? query returns the value of the *PSC flag.

Returned Format `1<NL>`

Example This example places the *PSC flag into the integer variable varPscflag.

```
myScope.WriteString "*PSC?"
varPscflag = myScope.ReadNumber
Debug.Print FormatNumber(varPscflag, 0)
```

*RCL

(Recall)

Command `*RCL <register>`

The *RCL command restores the state of the oscilloscope to a setup previously stored in the specified save/recall register. An oscilloscope setup must have been stored previously in the specified register. Registers 0 through 9 are general-purpose registers and can be used by the *RCL command.

<register> An integer, 0 through 9, specifying the save/recall register that contains the oscilloscope setup you want to recall.

Example This example restores the oscilloscope to the oscilloscope setup stored in register 3.

```
myScope.WriteString "*RCL 3"
```

See Also *SAV (Save). An error message appears on the oscilloscope's display if nothing has been previously saved in the specified register.

*RST

(Reset)

Command `*RST`

The *RST command performs a default setup which is the same as pressing the oscilloscope front panel default key.

Example This example resets the oscilloscope to a known state.

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

NOTE The default values for all of the Infiniium controls is located in the Infiniium Help System under Default Setup.

*SAV

(Save)

Command `*SAV <register>`

The *SAV command stores the current state of the oscilloscope in a save register.

<register> An integer, 0 through 9, specifying the register used to save the current oscilloscope setup.

Example This example stores the current oscilloscope setup to register 3.

```
myScope.WriteString "*SAV 3"
```

See Also *RCL (Recall).

*SRE

(Service Request Enable)

Command `*SRE <mask>`

The *SRE command sets the Service Request Enable Register bits. By setting the *SRE, when the event happens, you have enabled the oscilloscope's interrupt capability. The oscilloscope will then do an SRQ (service request), which is an interrupt.

<mask> An integer, 0 to 255, representing a mask value for the bits to be enabled in the Service Request Enable Register as shown in Table 20.

Example This example enables a service request to be generated when a message is available in the output queue. When a message is available, the MAV bit is high.

```
myScope.WriteString "*SRE 16"
```

Query `*SRE?`

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

Returned Format `<mask><NL>`

<mask> An integer, 0 to 255, representing a mask value for the bits enabled in the Service Request Enable Register.

Example This example places the current contents of the Service Request Enable Register in the numeric variable, varValue, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*SRE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A "1" in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A "0" disables the bit.

Table 20 Service Request Enable Register Bits

Bit	Weight	Enables
7	128	OPER - Operation Status Register
6	64	Not Used
5	32	ESB - Event Status Bit
4	16	MAV - Message Available
3	8	Not Used

Table 20 Service Request Enable Register Bits (continued)

Bit	Weight	Enables
2	4	MSG - Message
1	2	USR - User Event Register
0	1	TRG - Trigger

*STB?

(Status Byte)

Query `*STB?`

The *STB? query returns the current contents of the Status Byte, including the Master Summary Status (MSS) bit. See Table 21 for Status Byte Register bit definitions.

Returned Format `<value><NL>`

<value> An integer, 0 to 255, representing a mask value for the bits enabled in the Status Byte.

Example This example reads the contents of the Status Byte into the numeric variable, varValue, then prints the value of the variable to the computer's screen.

```
myScope.WriteString "*STB?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

In response to a serial poll (SPOLL), Request Service (RQS) is reported on bit 6 of the status byte. Otherwise, the Master Summary Status bit (MSS) is reported on bit 6. MSS is the inclusive OR of the bitwise combination, excluding bit 6, of the Status Byte Register and the Service Request Enable Register. The MSS message indicates that the oscilloscope is requesting service (SRQ).

Table 21 Status Byte Register Bits

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
7	128	OPER	0 = no enabled operation status conditions have occurred 1 = an enabled operation status condition has occurred
6	64	RQS/MSS	0 = oscilloscope has no reason for service 1 = oscilloscope is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	---	0 = not used

Table 21 Status Byte Register Bits (continued)

Bit	Bit Weight	Bit Name	Condition (0 = False = Low, 1 = True = High)
2	4	MSG	0 = no message has been displayed 1 = message has been displayed
1	2	USR	0 = no enabled user event conditions have occurred 1 = an enabled user event condition has occurred
0	1	TRG	0 = no trigger has occurred 1 = a trigger occurred

*TRG

(Trigger)

Command `*TRG`

The *TRG command has the same effect as the Group Execute Trigger message (GET) or RUN command. It acquires data for the active waveform display, if the trigger conditions are met, according to the current settings.

Example This example starts the data acquisition for the active waveform display according to the current settings.

```
myScope.WriteString "*TRG"
```

NOTE

Trigger Conditions Must Be Met

When you send the *TRG command in Single trigger mode, the trigger conditions must be met before the oscilloscope will acquire data.

*TST?

(Test)

Query `*TST?`

The *TST? query causes the oscilloscope to perform a self-test, and places a response in the output queue indicating whether or not the self-test completed without any detected errors. Use the :SYSTem:ERRor command to check for errors. A zero indicates that the test passed and a non-zero indicates the self-test failed.

NOTE

Disconnect Inputs First

You must disconnect all front-panel inputs before sending the *TST? command.

Returned Format `<result><NL>`

<result> 0 for pass; non-zero for fail.

Example This example performs a self-test on the oscilloscope and places the results in the numeric variable, varResults. The program then prints the results to the computer's screen.

```
myScope.WriteString "*TST?"
varResults = myScope.ReadNumber
Debug.Print FormatNumber(varResults, 0)
```

If a test fails, refer to the troubleshooting section of the service guide.

NOTE

Expanded Error Reporting

The :SELFtest:SCOPETEST command has expanded error reporting. Instead of using *TST?, Agilent recommends that you use the :SELFtest:SCOPETEST command. In either case, be sure you disconnect all front-panel inputs before sending the *TST? command.

*WAI

(Wait)

Command `*WAI`

The *WAI command has no function in the oscilloscope, but is parsed for compatibility with other instruments.

Example

```
myScope.WriteString "*WAI"
```

14 Digital Commands

:DIGital<N>:DISPlay 276
:DIGital<N>:LABel 277
:DIGital<N>:SIZE 278
:DIGital<N>:THReshold 279

NOTE The DIGital commands only apply to the MSO oscilloscopes.

:DIGital<N>:DISPlay

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
:DIGital<N>[:DISPlay] {ON | OFF | 1 | 0}
```

The :DIGital<N>:DISPlay command enables or disables the view for the selected digital channel.

<N> An integer, 0-15.

Example This example turns on the display of bit 5 for the digital channels.

```
myScope.WriteString ":DIGital5:DISPlay ON"
```

Query

```
:DIGital<N>[:DISPlay]?
```

The :DIGital<N>:DISPlay? query returns the value of the display setting for the selected digital channel.

Returned Format

```
[:DIGital<N>:DISPlay] {1 | 0}<NL>
```

:DIGital<N>:LABel

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
:DIGital<N>:LABel <quoted_string>
```

The :DIGital<N>:LABel command sets the digital channel label to the quoted string. Setting a label for a digital channel will also result in the name being added to the label list.

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.

<N> An integer, 1-2.

<quoted_string> A series of 16 or less characters as a quoted ASCII string.

Example This example sets the label for bit 7 to Clock.

```
myScope.WriteString ":DIGital7:LABel ""Clock"""
```

Query

```
:DIGital<N>:LABel?
```

The :DIGital<N>:LABel? query returns the name of the specified digital channel.

Returned Format

```
[:DIGital<N>:LABel] <quoted_string><NL>
```

:DIGital<N>:SIZE

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
DIGital<N>:SIZE {SMALl | MEDium | LARGe}
```

The :DIGital<N>:SIZE command changes the vertical size of all the displayed digital channels. The digital subsystem must be enabled before this command will work. See ENABle command in the root subsystem.

<N> An integer, 0-15.

Example This example changes the size to medium for all displayed digital channels or buses.

```
myScope.WriteString ":ENABLE DIGITAL"
myScope.WriteString ":DIGITAL5:SIZE MEDIUM"
```

Query

```
:DIGital<N>:SIZE?
```

The :DIGital:CHANnel:SIZE? query returns the size of the displayed digital channels.

Returned Format

```
[:DIGital<N>:SIZE] {SMALl | MEDium | LARGe}<NL>
```

:DIGital<N>:THReshold

Command

NOTE The DIGital commands only apply to the MSO oscilloscopes.

```
:DIGital<N>:THReshold {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFFerential | <value>}
```

The :DIGital<N>:THReshold command sets the logic threshold value for a pod. Setting the threshold for digital channels 0 through 7 sets the threshold for pod 1 while setting the threshold for digital channels 8 through 15 sets the threshold for pod 2. This command is equivalent to the POD<N>:THReshold command.

The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold). The voltage values for the predefined thresholds are:

- CMOS50 = 2.5 V
- CMOS33 = 1.65 V
- CMOS25 = 1.25 V
- ECL = -1.3 V
- PECL = 3.7 V
- TTL = 1.4 V
- DIFFerential = 0 V

<N> An integer, 0-15.

<value> A real number representing the voltage value which distinguishes a 1 logic level from a 0 logic level. Waveform voltages greater than the threshold are 1 logic levels while waveform voltages less than the threshold are 0 logic levels.

The range of the threshold voltage is from -3.75 volts to 3.75 volts.

Example This example sets the threshold to 1.8 volts for bits D15 through D8.

```
myScope.WriteString ":DIGital8:THReshold 1.8"
```

Query

```
:DIGital<N>:THREShold?
```

The :DIGital<N>:THReshold? query returns the threshold value for the specified pod.

Returned Format

```
[:DIGital<N>:THReshold] {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFF | <value>}<NL>
```

15 Disk Commands

:DISK:CDIRectory 282
:DISK:COPY 283
:DISK:DELete 284
:DISK:DIRectory? 285
:DISK:LOAD 286
:DISK:MDIRectory 287
:DISK:PWD? 288
:DISK:SAVE:COMPosite 289
:DISK:SAVE:IMAGe 290
:DISK:SAVE:JITTer 291
:DISK:SAVE:LISTing 292
:DISK:SAVE:MEASurements 293
:DISK:SAVE:PRECprobe 294
:DISK:SAVE:SETup 295
:DISK:SAVE:WAVeform 296
:DISK:SEGMented 298

The DISK subsystem commands perform the disk operations as defined in the File menu. This allows saving and loading of waveforms and setups, as well as saving screen images to bitmap files.

NOTE

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

NOTE

Filenames are Not Case Sensitive.

The filename that you use is not case sensitive.

:DISK:CDIRectory

Command `:DISK:CDIRectory "<directory>"`

The :DISK:CDIRectory command changes the present working directory to the designated directory name. An error occurs when the requested directory does not exist. You can then view the error with the :SYSTem:ERRor? [{NUMBer | STRing}] query.

<directory> A character-quoted ASCII string, which can include the subdirectory designation. You must separate the directory name and any subdirectories with a backslash (\).

Example This example sets the present working directory to C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

```
myScope.WriteString ":DISK:CDIRECTORY ""C:\Document and Settings\
All Users\Shared Documents\Infiniium\Data"""
```

:DISK:COPY

Command `:DISK:COPY "<source_file>","<dest_file>"`

The :DISK:COPY command copies a source file from the disk to a destination file on the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<source_file>
<dest_file>
A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example copies FILE1.SET to NEWFILE.SET on the disk.

```
myScope.WriteString ":DISK:COPY ""FILE1.SET"","NEWFILE.SET"""
```

:DISK:DELete

Command `:DISK:DELete "<file_name>"`

The :DISK:DELete command deletes a file from the disk. An error is displayed on the oscilloscope screen if the requested file does not exist. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<file_name> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example deletes FILE1.SET from the disk.

```
myScope.WriteString ":DISK:DELETE ""FILE1.SET"""
```

:DISK:DIRectory?

Query `:DISK:DIRectory? ["<directory>"]`

The :DISK:DIRectory? query returns the requested directory listing. Each entry is 63 bytes long, including a carriage return and line feed. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<directory> The list of filenames and directories.

Returned Format `[:DISK:DIRectory]<n><NL><directory>`

<n> The specifier that is returned before the directory listing, indicating the number of lines in the listing.

<directory> The list of filenames and directories. Each line is separated by a <NL>.

Example This example displays a number, then displays a list of files and directories in the current directory. The number indicates the number of lines in the listing.

```
Dim varResults As Variant
Dim lngI As Long

myScope.WriteString ":DISK:DIR?"
varResults = myScope.ReadList(ASCIIType_BSTR, vbLf)
Debug.Print FormatNumber(varResults(0), 0)

For lngI = 1 To (varResults(0) - 2)
  Debug.Print CStr(varResults(lngI))
Next lngI
```

:DISK:LOAD

Command `:DISK:LOAD "<file_name>"[,<destination>]`

The :DISK:LOAD command restores a setup or a waveform from the disk. The type of file is determined by the filename suffix if one is present, or by the destination field if one is not present. You can load .WFM, .CSV, .TSV, .TXT, .BIN, .H5, and .SET file types. The destination is only used when loading a waveform memory.

CAUTION Setups saved from Infiniium software versions prior to 2.00 may not load correctly in software versions 4.30 and greater.

You can remedy this by re-saving any pre-2.00 setups using any version of software from version 2.00 to version 4.20.

Setups saved from software versions between 2.00 and 4.20 should load correctly into version 4.30 and greater.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. You can use either .WFM, .CSV, .TSV, .TXT, .BIN, .H5, or .SET as a suffix after the filename. If no file suffix is specified, the default is .wfm.

The present working directory is assumed, or you can specify the entire path. For example, you can load the standard setup file "SETUP0.SET" using the command:

:DISK:LOAD "C:\Document and Settings\All Users\Shared Documents\Infiniium\Setups\SETUP0.SET"

Or, you can use :DISK:CDIRectory to change the present working directory to C:\Document and Settings\All Users\Shared Documents\Infiniium\Setups, then just use the file name ("SETUP0.SET", for example). The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<destination> WMEMory<N>.

Where <N> is an integer from 1-4.

If a destination is not specified, waveform memory 1 is used.

Example This example restores the waveform in FILE1.WFM to waveform memory 1.

```
myScope.WriteString ":DISK:LOAD ""FILE1.WFM"",WMEM1"
```

:DISK:MDIRectory

Command `:DISK:MDIRectory "<directory>"`

The :DISK:MDIRectory command creates a directory in the present working directory which has been set by the :DISK:CDIRectory command. If the present working directory has not been set by the :DISK:CDIRectory command, you must specify the full path in the <directory> parameter as shown in Example 1 below.

An error is displayed if the requested subdirectory does not exist.

<directory> A quoted ASCII string which can include subdirectories. You must separate the directory name and any subdirectories with a backslash (\).

Example 1 This example creates the directory CPROGRAMS in the C:\Document and Settings\All Users\Shared Documents\Infiniium\Data directory.

```
myScope.WriteString _
  ":DISK:MDIRECTORY ""C:\Document and Settings\All Users\
Shared Documents\Infiniium\Data\CPROGRAMS"""
```

Example 2 This example creates the directory CPROGRAMS in the present working directory set by the :DISK:CDIRectory command.

```
myScope.WriteString ":DISK:MDIRECTORY ""CPROGRAMS"""
```

You can check your path with the :DISK:DIRectory? query.

:DISK:PWD?

Query `:DISK:PWD?`

The :DISK:PWD? query returns the name of the present working directory (including the full path). If the default path (C:\Document and Settings\All Users\Shared Documents\Infiniium\Data) has not been changed by the :DISK:CDIRectory command, the :DISK:PWD? query will return an empty string.

Returned Format `:DISK:PWD? <present_working_directory><NL>`

Example This example places the present working directory in the string variable strWdir, then prints the contents of the variable to the computer's screen.

```
Dim strWdir As String
myScope.WriteString ":DISK:PWD?"
str Wdir = myScope.ReadString
Debug.Print strWdir
```

:DISK:SAVE:COMPosite

Command `:DISK:SAVE:COMPosite "<file_name>"`

The :DISK:SAVE:COMPosite command lets you save oscilloscope composite files to Infiniium's hard disk or to a network drive. Composite files contain setups and waveform data.

The file will have an .osc extension.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

The filename assumes the present working directory if a path does not precede the file name.

Example This example saves the oscilloscope's setup and waveform data to a composite file named "C:\Scope\Setup\Comp001.osc".

```
myScope.WriteString ":DISK:SAVE:COMPosite ""C:\Scope\Setup\Comp001"""
```

:DISK:SAVE:IMAGe

Command

```
:DISK:SAVE:IMAGe "<file_name>" [,<format>
[,{SCReen | GRATicule}
[,{ON | 1} | {OFF | 0}
[,{NORMal | INVert}
[,{ON | 1} | {OFF | 0}]]]]]
```

The DISK:SAVE:IMAGe command saves a screen image. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<format> The image format can be: BMP, GIF, TIF, PNG, or JPEG. The extension is supplied by the oscilloscope depending on the selected file format.

If you do not include the format in the command, the file is saved in the format shown in the Save Screen dialog box.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

(First) ON | OFF ON means that compression is on for the bitmap format (BMP). OFF means compression is off.

(Second) ON | OFF The second ON/OFF selection indicates to save the setup information in the image or not.

<format> {BMP | GIF | TIF | JPEG | PNG}

Examples

```
myScope.WriteString ":DISK:SAVE:IMAGE ""FILE1"",BMP,SCR,ON,INVERT"
```

or:

```
myScope.WriteString ":DISK:SAVE:IMAGE ""FILE1"",TIF,GRAT,ON"
```

or:

```
myScope.WriteString ":DISK:SAVE:IMAGE ""FILE1"""
```

:DISK:SAVE:JITTer

Command `:DISK:SAVE:JITTer "<file_name>"`

The DISK:SAVE:JITTer command saves the jitter measurements shown in the RJDJ tab at the bottom of the oscilloscope screen along with the RJDJ histograms in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
myScope.WriteString ":DISK:SAVE:JITTER ""FILE1"""
```

:DISK:SAVE:LISTing

Command `:DISK:SAVE:LISTing [<source>,] "<file_name>"[, <format>]`

The DISK:SAVE:LISTing command saves the contents of the bus listing window to a file in either a .csv or .txt format. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<source> {SERial<N>} — The default serial bus is the one currently displayed in the listing window.

<N> An integer 1 - 4.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

<format> {CSV | TXT}

Example

```
myScope.WriteString ":DISK:SAVE:LISTing SERial3, ""FILE1"", CSV"
```

:DISK:SAVE:MEASurements

Command `:DISK:SAVE:MEASurements "<file_name>"`

The DISK:SAVE:MEASurements command saves the measurements shown in the measurements tab at the bottom of the oscilloscope screen in a comma separated variables (CSV) file format. The csv extension is supplied by the oscilloscope. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
myScope.WriteString ":DISK:SAVE:MEASURMENTS ""FILE1"""
```

:DISK:SAVE:PRECprobe

Command `:DISK:SAVE:PRECprobe "<file_name>.csv", {CHAN1 | CHAN2 | CHAN3 | CHAN4}`

The DISK:SAVE:PRECprobe command saves PrecisionProbe/Cable data in a comma separated variables (CSV) file format. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used.

Example

```
myScope.WriteString ":DISK:SAVE:PRECprobe ""PPch1data.csv""", CHAN1
```

:DISK:SAVE:SETup

Command `:DISK:SAVE:SETup "<file_name>"`

The :DISK:SAVE:SETup command saves the current oscilloscope setup to a disk. The file will have a .set extension.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\SCOPE\SETUP.

Example This example saves the channel 1 waveform to SETUP1 on the disk.

```
myScope.WriteString ":DISK:SAVE:SETUP ""SEUP1"""
```

:DISK:SAVE:WAVeform

Command `:DISK:SAVE:WAVeform <source>,"<file_name>" [,<format>[,<header>]]`

The :DISK:SAVE:WAVeform command saves a waveform to a disk. If the source is ALL, all of the currently displayed waveforms are saved to the file. If you use a file extension as shown below in the <format> variable, then the type of file saved defaults to the extension type. If no format is specified and no extension is used, the file is saved in the INTernal format.

NOTE See the ":WAVeform:VIEW" on page 1029 command to determine how much data is saved.

<source> {ALL | CHANnel<N> | CLOCk | FUNCtion<F> | HISTogram | MTRend | MSPectrum | EQUalized | WMEMory<N> | PODALL | POD1 | POD2}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

If you have an MSO model oscilloscope, you can save digital channel waveform data by using the sources PODALL (all digital channels), POD1 (D0-D7), or POD2 (D8-D15).

<N> An integer, 1-4.

<F> An integer, 1-16.

<file_name> A quoted ASCII string with a maximum of 254 characters including the entire path name, if used. The filename assumes the present working directory if a path does not precede the file name. The default path is C:\Document and Settings\All Users\Shared Documents\Infiniium\Data.

<format> {BIN | CSV | INTernal | TSV | TXT | H5 | H5INt}

The following file name extensions are used for the different formats:

- BIN = file_name.bin
- CSV (comma separated values) = file_name.csv
- INTernal = file_name.wfm

- TSV (tab separated values) = file_name.tsv
- TXT = file_name.txt
- H5, H5INt (HDF5) = file_name.h5

In the H5 format, data is saved as floats. In the H5INt format, data is saved as integers.

<header> {{ON | 1} | {OFF | 0}}

Example This example saves the channel 1 waveform to FILE1 on the disk in the CSV format with header on.

```
myScope.WriteString ":DISK:SAVE:WAVeform CHANnel1,""FILE1"",CSV,ON"
```

:DISK:SEGMented

Command `:DISK:SEGMented {ALL | CURRent}`

The :DISK:SEGMented command sets whether all segments or just the current segment are saved to a file when the :DISK:SAVE:WAVeform command is issued and the source is a channel but not a waveform memory or function. Before segments can be saved, the :ACQuire:MODE must be set to the SEGMented mode and segments must be acquired.

Example This example sets the disk segmented memory store method to CURRent.

```
myScope.WriteString ":DISK:SEGMENTED CURRENT"
```

Query `:DISK:SEGMented?`

The :DISK:SEGMented? query returns disk segmented memory store method value.

Returned Format `[:DISK:SEGMented] {ALL | CURRent}<NL>`

Example This example places the disk store method in the string variable strMethod, then prints the contents of the variable to the computer's screen.

```
Dim strMethod As String
myScope.WriteString ":DISK:SEGMENTED?"
strMethod = myScope.ReadString
Debug.Print strMethod
```

16 Display Commands

:DISPlay:CGRade 300
:DISPlay:CGRade:LEVels? 301
:DISPlay:CGRade:SCHeme 303
:DISPlay:COLumn 305
:DISPlay:CONNect 306
:DISPlay:DATA? 307
:DISPlay:GRATicule 308
:DISPlay:GRATicule:INTensity 309
:DISPlay:GRATicule:NUMBer 310
:DISPlay:GRATicule:SETGrat 311
:DISPlay:GRATicule:SIZE 312
:DISPlay:LABel 313
:DISPlay:LINE 314
:DISPlay:PERSistence 315
:DISPlay:ROW 316
:DISPlay:SCOLor 317
:DISPlay:STATus:COL 319
:DISPlay:STATus:ROW 320
:DISPlay:STRing 321
:DISPlay:TAB 322
:DISPlay:TEXT 323

The DISPlay subsystem controls the display of data, text, and graticules, and the use of color.

:DISPlay:CGRade

Command `:DISPlay:CGRade {{ON | 1} | {OFF | 0}}`

The :DISPlay:CGRade command sets the color grade persistence on or off.

When in the color grade persistence mode, all waveforms are mapped into a database and shown with different colors representing varying number of hits in a pixel. "Connected dots" display mode (:DISPlay:CONNect) is disabled when the color grade persistence is on.

The oscilloscope has three features that use a specific database. This database uses a different memory area than the waveform record for each channel. The three features that use the database are:

- Histograms.
- Mask testing.
- Color grade persistence.

When any one of these three features is turned on, the oscilloscope starts building the database. The database is the size of the graticule area and varies in size. Behind each pixel is a 53-bit counter. Each counter is incremented each time a pixel is hit by data from a channel or function. The maximum count (saturation) for each counter is 9,007,199,254,740,991. You can check for counter saturation by using the DISPlay:CGRade:LEVels? query.

The color grade persistence uses colors to represent the number of hits on various areas of the display. The default color-grade state is off.

Example This example sets the color grade persistence on.

```
myScope.WriteString ":DISPlay:CGRade ON"
```

Query `:DISPlay:CGRade?`

The DISPlay:CGRade query returns the current color-grade state.

Returned Format `[:DISPlay:CGRade] {1 | 0}<NL>`

Example This example returns the current color grade state.

```
Dim strCgrade As String   ' Dimension variable.
myScope.WriteString ":DISPlay:CGRade?"
strCgrade = myScope.ReadString
Debug.Print strCgrade
```

See Also

- ":DISPlay:CGRade:LEVels?" on page 301
- ":DISPlay:CGRade:SCHeme" on page 303

:DISPlay:CGRade:LEVels?

Query `:DISPlay:CGRade:LEVels?`

The :DISPlay:CGRade:LEVels? query returns the range of hits represented by each color. Fourteen values are returned, representing the minimum and maximum count for each of seven colors. In the CLASsic color grade scheme, the values are returned in the following order:

- Green minimum value
- Green maximum value
- Blue minimum value
- Blue maximum value
- Pink minimum value
- Pink maximum value
- Red minimum value
- Red maximum value
- Orange minimum value
- Orange maximum value
- Yellow minimum value
- Yellow maximum value
- White minimum value
- White maximum value

Returned Format `[DISPlay:CGRade:LEVels] <color format><NL>`

<color format> <intensity color min/max> is an integer value from 0 to 9,007,199,254,740,991

Example This example gets the range of hits represented by each color and prints it on the computer screen:

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:CGRADE:LEVELS?"
strCgrade = myScope.ReadString
Debug.Print strCgrade
```

In the CLASsic color grade scheme, colors start at green minimum, maximum, then blue, pink, red, orange, yellow, white. The format is a string where commas separate minimum and maximum values. The largest number in the string can be 9,007,199,254,740,991

An example of a possible returned string is as follows:

1,414,415,829,830,1658,1659,3316,3317,6633,6634,13267,13268,26535

See Also

- ":DISPlay:CGRade" on page 300
- ":DISPlay:CGRade:SCHeme" on page 303

:DISPlay:CGRade:SCHeme

Command `:DISPlay:CGRade:SCHeme {CLASsic | TEMP}`

The :DISPlay:CGRade:SCHeme command sets the color grade scheme to CLASsic or TEMP.

Color grade persistence is displayed in seven different colors which represent the range of the counters in the database. In the CLASsic color grade scheme, the counters with the largest counts are displayed using a white pixel while the counters with the smallest counts are displayed using green pixels.

The following table shows the counter ranges for each color for both the CLASsic and TEMP color grade schemes.

Color Grade Scheme		Range
Classic	Temperature	
		$Counter_{Max} \rightarrow \frac{Counter_{Max}}{2} + 1$
		$Counter_{Max} - \frac{Counter_{Max}}{2} \rightarrow \frac{Counter_{Max}}{2} - \frac{Counter_{Max}}{4} + 1$
		$\frac{Counter_{Max}}{2} - \frac{Counter_{Max}}{4} \rightarrow \frac{Counter_{Max}}{4} - \frac{Counter_{Max}}{8} + 1$
		$\frac{Counter_{Max}}{4} - \frac{Counter_{Max}}{8} \rightarrow \frac{Counter_{Max}}{8} - \frac{Counter_{Max}}{16} + 1$
		$\frac{Counter_{Max}}{8} - \frac{Counter_{Max}}{16} \rightarrow \frac{Counter_{Max}}{16} - \frac{Counter_{Max}}{32} + 1$
		$\frac{Counter_{Max}}{16} - \frac{Counter_{Max}}{32} \rightarrow \frac{Counter_{Max}}{32} - \frac{Counter_{Max}}{64} + 1$
		$\frac{Counter_{Max}}{32} - \frac{Counter_{Max}}{64} \rightarrow 1$

Example This example sets the color grade scheme to "classic".

```
myScope.WriteString ":DISPlay:CGRade:SCHeme CLASsic"
```

Query `:DISPlay:CGRade:SCHeme?`

The :DISPlay:CGRade:SCHeme? query returns the specified color scheme.

Returned Format `[DISPlay:CGRade:SCHeme] {CLASsic | TEMP}<NL>`

Example This example gets the specified color scheme and prints it on the computer screen:

```
Dim strCgradeScheme As String   ' Dimension variable.
myScope.WriteString ":DISPlay:CGRade:SCHeme?"
strCgradeScheme = myScope.ReadString
Debug.Print strCgradeScheme
```

See Also

- ":DISPlay:CGRade" on page 300
- ":DISPlay:CGRade:LEVels?" on page 301

:DISPlay:COLumn

Command `:DISPlay:COLumn <column_number>`

The :DISPlay:COLumn command specifies the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands.

<column_number> An integer representing the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 0 to 90.

Example This example sets the starting column for subsequent :DISPlay:STRing and :DISPlay:LINE commands to column 10.

```
myScope.WriteString ":DISPLAY:COLUMN 10"
```

Query `:DISPlay:COLumn?`

The :DISPlay:COLumn? query returns the column where the next :DISPlay:LINE or :DISPlay:STRing starts.

Returned Format `[:DISPlay:COLumn] <value><NL>`

Example This example returns the current column setting to the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:COLUMN?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:CONNect

Command `:DISPlay:CONNect {{ON | 1} | {OFF | 0}}`

When enabled, :DISPlay:CONNect draws a line between consecutive waveform data points. This is also known as linear interpolation.

:DISPlay:CONNect is forced to OFF when color grade (:DISPlay:CGRade) persistence is on.

Example This example turns on the connect-the-dots feature.

```
myScope.WriteString ":DISPLAY:CONNECT ON"
```

Query `:DISPlay:CONNect?`

The :DISPlay:CONNect? query returns the status of the connect-the-dots feature.

Returned Format `[:DISPlay:CONNect] {1 | 0}<NL>`

:DISPlay:DATA?

Query `:DISPlay:DATA? [<type>[,<screen_mode>[,<compression> [,<inversion>]]]]`

The :DISPlay:DATA? query returns information about the captured data. If no options to the query are specified, the default selections are BMP file type, SCReen mode, compression turned ON, and inversion set to NORMal.

<type> The bitmap type: BMP | JPG | GIF | TIF | PNG.

<screen_mode> The display setting: SCReen | GRATicule. Selecting GRATicule displays a 10-by-8 (unit) display graticule on the screen. See also :DISPlay:GRATicule.

<compression> The file compression feature: ON | OFF.

<inversion> The inversion of the displayed file: NORMal | INVert.

Returned Format `[:DISPlay:DATA] <binary_block_data><NL>`

<binary_block_data> Data in the IEEE 488.2 definite block format.

:DISPlay:GRATicule

Commands

```
:DISPlay:GRATicule {GRID | FRAMe}
```

The :DISPlay:GRATicule command selects the type of graticule that is displayed. Infiniium oscilloscopes have a 10-by-8 (unit) display graticule grid GRID), a grid line is place on each vertical and horizontal division. When it is off (FRAMe), a frame with tic marks surrounds the graticule edges.

Example This example sets up the oscilloscope's display background with a frame that is separated into major and minor divisions.

```
myScope.WriteString ":DISPlay:GRATicule FRAMe"
```

Queries

```
:DISPlay:GRATicule?
```

The :DISPlay:GRATicule? query returns the type of graticule currently displayed.

Returned Format

```
[:DISPlay:GRATicule] {GRID | FRAMe}<NL>
```

Example This example places the current display graticule setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule:INTensity" on page 309
- ":DISPlay:GRATicule:NUMBer" on page 310
- ":DISPlay:GRATicule:SETGrat" on page 311
- ":DISPlay:GRATicule:SIZE" on page 312

:DISPlay:GRATicule:INTensity

Commands

```
:DISPlay:GRATicule:INTensity <intensity_value>
```

You can dim the grid's intensity or turn the grid off to better view waveforms that might be obscured by the graticule lines using the :DISPlay:GRATicule:INTensity command. Otherwise, you can use the grid to estimate waveform measurements such as amplitude and period.

When printing, the grid intensity control does not affect the hard copy. To remove the grid from a printed hard copy, you must turn off the grid before printing.

<intensity_value> A integer from 0 to 100, indicating the percentage of grid intensity.

Example This example sets the graticule intensity to 50%.

```
myScope.WriteString ":DISPlay:GRATicule:INTensity 50"
```

Queries

```
:DISPlay:GRATicule:INTensity?
```

The :DISPlay:GRATicule:INTensity? query returns the intensity.

Returned Format

```
[:DISPlay:GRATicule:INTensity] <value><NL>
```

Example This example places the current graticule intensity setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:INTensity?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 308
- ":DISPlay:GRATicule:NUMBer" on page 310
- ":DISPlay:GRATicule:SETGrat" on page 311
- ":DISPlay:GRATicule:SIZE" on page 312

:DISPlay:GRATicule:NUMBer

Commands

```
:DISPlay:GRATicule:NUMBer {1 | 2 | 4 | 8 | 16}
```

You can divide the waveform viewing area from one area into two, four, eight, or sixteen separate viewing areas using the :DISPlay:GRATicule:NUMBer command. This lets you separate waveforms without having to adjust the vertical position controls.

Example This example sets up two viewing areas.

```
myScope.WriteString ":DISPlay:GRATicule:NUMBer 2"
```

Queries

```
:DISPlay:GRATicule:NUMBer?
```

The :DISPlay:GRATicule:NUMBer? query returns the the number of viewing areas.

Returned Format

```
[:DISPlay:GRATicule:NUMBer {1 | 2 | 4 | 8 | 16}<NL>
```

Example This example places the current number of viewing areas in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:NUMBer?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 308
- ":DISPlay:GRATicule:INTensity" on page 309
- ":DISPlay:GRATicule:SETGrat" on page 311
- ":DISPlay:GRATicule:SIZE" on page 312

:DISPlay:GRATicule:SETGrat

Commands `:DISPlay:GRATicule:SETGrat <DispGratChan>,<number>`

The :DISPlay:GRATicule:SETGrat command assigns the corresponding waveform to a specific grid on the display.

<DispGratChan> Can be:

- CHN<N>
- DIFF1, DIFF3
- COMM2, COMM4
- MEM<N> where N is between 1 and 4
- FN<N> where N is between 1 and 16 (function)
- HIST
- D<M> where M is between 0 and 15 (on MSO models with 16 digital channels)
- BUS<Y> where Y is between 1 and 4 (on MSO models)

<number> 1- 16, the number of the grid you want to assign the waveform to.

Example This example assigns the histogram to grid 2.

```
myScope.WriteString ":DISPlay:GRATicule:SETGrat HIST,2"
```

See Also

- ":DISPlay:GRATicule" on page 308
- ":DISPlay:GRATicule:INTensity" on page 309
- ":DISPlay:GRATicule:NUMBer" on page 310
- ":DISPlay:GRATicule:SIZE" on page 312

:DISPlay:GRATicule:SIZE

Commands

```
:DISPlay:GRATicule:SIZE {STANdard | EXTended | MAXimized | MINimized}
```

The :DISPlay:GRATicule:SIZE command sets the graticule size.

The opposite effect of this command is to size the Measurements tab. This is, to get the biggest Measurements tab area, you minimize the graticule size.

Example This example sets the extended graticule size.

```
myScope.WriteString ":DISPlay:GRATicule:SIZE EXTended"
```

Queries

```
:DISPlay:GRATicule:SIZE?
```

The :DISPlay:GRATicule:SIZE? query returns the graticule size.

Returned Format

```
[:DISPlay:GRATicule:SIZE]
   {STANdard | EXTended | MAXimized | MINimized}<NL>
```

Example This example places the current graticule size setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:GRATicule:SIZE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":DISPlay:GRATicule" on page 308
- ":DISPlay:GRATicule:INTensity" on page 309
- ":DISPlay:GRATicule:NUMBer" on page 310
- ":DISPlay:GRATicule:SETGrat" on page 311

:DISPlay:LABel

Command `:DISPlay:LABel {{ON | 1} | {OFF | 0}}`

The :DISPlay:LABel command turns on or off the display of analog channel labels. Label names can be up to 6 characters long. The label name is assigned by using the CHANnel<n>:LABel command:

Example This example turns on the display of all labels.

```
myScope.WriteString ":DISPLAY:LABEL ON"
```

Query `:DISPlay:LABel?`

The :DISPlay:LABel? query returns the current state of the labels.

Returned Format `[:DISPlay:LABel] {1 | 0}<NL>`

Example This example places the current label state into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:LABEL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:LINE

Command `:DISPlay:LINE "<string_argument>"`

The :DISPlay:LINE command writes a quoted string to the screen, starting at the location specified by the :DISPlay:ROW and :DISPlay:COLumn commands.

<string_argument> Any series of ASCII characters enclosed in quotation marks.

Example This example writes the message "Infiniium Test" to the screen, starting at the current row and column location.

```
myScope.WriteString ":DISPLAY:LINE ""Infiniium Test"""
```

When using the C programming language, quotation marks within a string are escaped using the backslash (\) character as shown in the next example. This example writes the message "Infiniium Test" to the screen.

```
printf("\"Infiniium Test\"");
```

You may write text up to column 94. If the characters in the string do not fill the line, the rest of the line is blanked. If the string is longer than the space available on the current line, the excess characters are discarded.

In any case, the ROW is incremented and the COLumn remains the same. The next :DISPlay:LINE command will write on the next line of the display. After writing the last line in the display area, the ROW is reset to 0.

:DISPlay:PERSistence

Command `:DISPlay:PERSistence {MINimum | INFinite | <time>}`

`<time> ::= seconds in in NR3 format from 100E-3 to 200E0`

The :DISPlay:PERSistence command sets the display persistence. The parameter for this command can be:

- MINimum — indicates zero persistence.
- INFinite — indicates infinite persistence.
- <time> — for variable persistence, that is, you can specify how long acquisitions remain on the screen.

Example This example sets the persistence to infinite.

```
myScope.WriteString ":DISPlay:PERSistence INFinite"
```

Query `:DISPlay:PERSistence?`

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

Returned Format `[:DISPlay:PERSistence] {MINimum | INFinite | <time>}<NL>`

Example This example places the current persistence setting in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPlay:PERSistence?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:ROW

Command `:DISPlay:ROW <row_number>`

The :DISPlay:ROW command specifies the starting row on the screen for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The row number remains constant until another :DISPlay:ROW command is received, or the row is incremented by the :DISPlay:LINE command.

<row_number> An integer representing the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands. The range of values is 9 to 23.

Example This example sets the starting row for subsequent :DISPlay:STRing and :DISPlay:LINE commands to 10.

```
myScope.WriteString ":DISPLAY:ROW 10"
```

Query `:DISPlay:ROW?`

The :DISPlay:ROW? query returns the current value of the row.

Returned Format `[:DISPlay:ROW] <row_number><NL>`

Example This example places the current value for row in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:ROW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:SCOLor

Command `:DISPlay:SCOLor <color_name>, <hue>, <saturation>, <luminosity>`

The :DISPlay:SCOLor command sets the color of the specified display element. The display elements are described in Table 22.

<color_name> `{CGLevel1 | CGLevel2 | CGLevel3 | CGLevel4 | CGLevel5 | CGLevel6 | CGLevel7 | CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4 | DBACkgrnd | GRID | MARKers | MEASurements | MICons | MTPolygons | STEXt | WBACkgrnd | TINPuts | WOVerlap | TSCale | WMEMories | WINText | WINBackgrnd}`

Table 22 Color Names

Color Name	Definition
CGLevel1	Color Grade Level 1 waveform display element.
CGLevel2	Color Grade Level 2 waveform display element.
CGLevel3	Color Grade Level 3 waveform display element.
CGLevel4	Color Grade Level 4 waveform display element.
CGLevel5	Color Grade Level 5 waveform display element.
CGLevel6	Color Grade Level 6 waveform display element.
CGLevel7	Color Grade Level 7 waveform display element.
CHANnel1	Channel 1 waveform display element.
CHANnel2	Channel 2 waveform display element.
CHANnel3	Channel 3 waveform display element.
CHANnel4	Channel 4 waveform display element.
DBACkgrnd	Display element for the border around the outside of the waveform viewing area.
GRID	Display element for the grid inside the waveform viewing area.
MARKers	Display element for the markers.
MEASurements	Display element for the measurements text.
MICons	Display element for measurement icons to the left of the waveform viewing area.
STEXt	Display element for status messages displayed in the upper left corner of the display underneath the menu bar. Changing this changes the memory bar's color.
WBACkgrnd	Display element for the waveform viewing area's background.
TINPuts	Display element for line and aux trig colors.

Table 22 Color Names (continued)

Color Name	Definition
WOVerlap	Display element for waveforms when they overlap each other.
TSCale	Display element for horizontal scale and offset control text.
WMEMories	Display element for waveform memories.
WINText	Display element used in dialog box controls and pull-down menus.
WINBackgrnd	Display element for the background color used in dialog boxes and buttons.

<hue> An integer from 0 to 100. The hue control sets the color of the chosen display element. As hue is increased from 0%, the color changes from red, to yellow, to green, to blue, to purple, then back to red again at 100% hue. For color examples, see the sample color settings table in the Infiniium Oscilloscope online help file. Pure red is 100%, pure blue is 67%, and pure green is 33%.

<saturation> An integer from 0 to 100. The saturation control sets the color purity of the chosen display element. The saturation of a color is the purity of a color, or the absence of white. A 100% saturated color has no white component. A 0% saturated color is pure white.

<luminosity> An integer from 0 to 100. The luminosity control sets the color brightness of the chosen display element. A 100% luminosity is the maximum color brightness. A 0% luminosity is pure black.

Example This example sets the hue to 50, the saturation to 70, and the luminosity to 90 for the markers.

```
myScope.WriteString ":DISPLAY:SCOLOR MARKERS,50,70,90"
```

Query

```
:DISPlay:SCOLor? <color_name>
```

The :DISPlay:SCOLor? query returns the hue, saturation, and luminosity for the specified color.

Returned Format

```
[:DISPlay:SCOLor] <color_name>, <hue>, <saturation>, <luminosity><NL>
```

Example This example places the current settings for the graticule color in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:SCOLOR? GRATICULE"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:STATus:COL

Command `:DISPlay:STATus:COL <column>`

The :DISPlay:STATus:COL command is used to position the real time eye and InfiniiScan Zone Trigger status labels.

This and the :DISPlay:STATus:ROW commands specify the upper left corner of the box relative to the screen.

<column> A value of 0 to 1 may be given for the column where 0 is the far left and 1 the far right.

Example For example, a column of 0.5 will place the upper left of the status label at the center screen.

```
myScope.WriteString ":DISPLAY:STATus:COL 0.5"
```

Query `:DISPlay:STATus:COL?`

The :DISPlay:STATus:COL? query returns the current value of the status label column location.

Returned Format `[:DISPlay:STATus:COL] <column><NL>`

Example This example places the current value for the status label column location in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:STATus:COL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:STATus:ROW

Command `:DISPlay:STATus:ROW <row>`

The :DISPlay:STATus:ROW command is used to position the real time eye and InfiniiScan Zone Trigger status labels.

This and the :DISPlay:STATus:COL commands specify the upper left corner of the box relative to the screen.

<row> A value of 0 to 1 may be given for the row where 0 is the far top and 1 the far bottom.

Example For example, a row and column of 0.5 will place the upper left of the status label at the center screen.

```
myScope.WriteString ":DISPLAY:STATus:ROW 0.5"
```

Query `:DISPlay:STATus:ROW?`

The :DISPlay:STATus:ROW? query returns the current value of the status label row location.

Returned Format `[:DISPlay:STATus:ROW] <row><NL>`

Example This example places the current value for the status label row location in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":DISPLAY:STATus:ROW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:DISPlay:STRing

Command `:DISPlay:STRing "<string_argument>"`

The :DISPlay:STRing command writes text to the oscilloscope screen. The text is written starting at the current row and column settings. If the column limit is reached, the excess text is discarded. The :DISPlay:STRing command does not increment the row value, but :DISPlay:LINE does.

<string_argument> Any series of ASCII characters enclosed in quotation marks.

Example This example writes the message "Example 1" to the oscilloscope's display starting at the current row and column settings.

```
myScope.WriteString ":DISPLAY:STRING ""Example 1"""
```

:DISPlay:TAB

Command `:DISPlay:TAB <tab>`

The :DISPlay:TAB command displays the corresponding tab indicated by the <tab> parameter.

<tab> MEASurement | MARKer | LIMittest | JITTer | NOISe | HISTogram | MASKtest | EYE | COLorgrade | NAVigation | STATus | SCALe

Example This example sets the Status tab as the displayed one.

```
myScope.WriteString ":DISPlay:TAB STATus"
```

Query `:DISPlay:TAB?`

The :DISPlay:TAB? query returns the tab that is currently displayed.

Returned Format `[:DISPlay:TAB] {MEAS | MARK | LIM | JITT | NOIS | HIST | MASK | EYE | COL | NAV | STAT | SCAL}<NL>`

Example This example places the currently displayed tab into the string variable, strTab, then prints the contents of the variable to the computer's screen.

```
Dim strTab As String   ' Dimension variable.
myScope.WriteString ":DISPlay:TAB?"
strTab = myScope.ReadString
Debug.Print strTab
```

:DISPlay:TEXT

Command `:DISPlay:TEXT BLANk`

The :DISPlay:TEXT command blanks the user text area of the screen.

Example This example blanks the user text area of the oscilloscope's screen.

```
myScope.WriteString ":DISPLAY:TEXT BLANK"
```

17 Function Commands

:FUNCtion<F>? 328
:FUNCtion<F>:ABSolute 329
:FUNCtion<F>:ADD 330
:FUNCtion<F>:ADEMod 331
:FUNCtion<F>:AVERage 332
:FUNCtion<F>:COMMonmode 333
:FUNCtion<F>:DELay 334
:FUNCtion<F>:DIFF 335
:FUNCtion<F>:DISPlay 336
:FUNCtion<F>:DIVide 337
:FUNCtion<F>:FFT:FREQuency 338
:FUNCtion<F>:FFT:REFerence 339
:FUNCtion<F>:FFT:RESolution? 340
:FUNCtion<F>:FFT:TDELay 341
:FUNCtion<F>:FFT:WINDow 342
:FUNCtion<F>:FFTMagnitude 344
:FUNCtion<F>:FFTPhase 345
:FUNCtion<F>:HIGHpass 347
:FUNCtion<F>:HORizontal 348
:FUNCtion<F>:HORizontal:POSition 349
:FUNCtion<F>:HORizontal:RANGe 350
:FUNCtion<F>:INTegrate 351
:FUNCtion<F>:INVert 352
:FUNCtion<F>:LOWPass 353
:FUNCtion<F>:MAGNify 354
:FUNCtion<F>:MAXimum 355
:FUNCtion<F>:MHIStogram 356
:FUNCtion<F>:MINimum 357
:FUNCtion<F>:MTRend 358
:FUNCtion<F>:MULTiply 359
:FUNCtion<F>:OFFSet 360
:FUNCtion<F>:RANGe 361
:FUNCtion<F>:SMOoth 362
:FUNCtion<F>:SQRT 363

:FUNCtion<F>:SQUare 364
:FUNCtion<F>:SUBTract 365
:FUNCtion<F>:VERSus 366
:FUNCtion<F>:VERTical 367
:FUNCtion<F>:VERTical:OFFSet 368
:FUNCtion<F>:VERTical:RANGe 369

The FUNCtion subsystem defines functions 1-16. The operands of these functions can be:

- Any of the installed channels in the oscilloscope (see page 326)
- Differential channels or common mode channels (see page 327)
- Waveform memories (see page 327)
- Functions (see page 327)
- A constant (see page 327)
- Jitter measurement trend or jitter spectrum (see page 327)

You can control the vertical scaling and offset functions remotely using the RANGe and OFFSet commands in this subsystem. You can obtain the horizontal scaling and position values of the functions using the :HORizontal:RANge? and :HORizontal:POSition? queries in this subsystem.

If a channel is not on but is used as an operand, that channel will acquire waveform data.

If the operand waveforms have different memory depths, the function uses the shorter of the two.

If the two operands have the same time scales, the resulting function has the same time scale. If the operands have different time scales, the resulting function has no valid time scale. This is because operations are performed based on the displayed waveform data position, and the time relationship of the data records cannot be considered. When the time scale is not valid, delta time pulse parameter measurements have no meaning, and the unknown result indicator is displayed on the screen.

Constant operands take on the same time scale as the associated waveform operand.

Channel Operands

CHANnel<N>, where N is an integer, 1-4.

Differential and Common Mode Channel Operands

DIFF<P>, where P is an integer, 1-2.

COMMonmode<P>, where P is an integer, 3-4.

The COMMonmode and DIFF sources are just aliases that can be used in place of the channel names to apply to differential or common mode signals. These are just aliases - no state change occurs if you refer to a differential channel and you are not in differential mode. DIFF1 refers to the differential signal between channels 1 and 3 (and COMMonmode3 refers to the common mode channel between these same channels). DIFF2 refers to the differential signal between channels 2 and 4 (and COMMonmode4 refers to the common mode channel between these same channels).

Waveform Memory Operands

WMEMory<N>, where N is an integer, 1-4.

Function Operands

FUNCtion<F>, where F is an integer, 1-16.

Another function can be a function's source as long as the other function doesn't use the function being defined. In other words, circular expressions are not allowed.

Constant Operands

Constant operands can be a real number from -1E6 to 1E12.

Jitter Measurement Trend and Jitter Spectrum Operands

The jitter measurement trend, MTRend, and jitter spectrum, MSPectrum, operands are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

:FUNCtion<F>?

Query `:FUNCtion<F>?`

The :FUNCtion<F>? query returns the currently defined source(s) for the function.

Returned Format `[:FUNCtion<F>:<operator>] {<operand>[,<operand>]}<NL>`

<F> An integer, 1-16, representing the selected function.

<operator> Active math operation for the selected function. For example, ADD, AVERage, COMMonmode, DIFF, DIVide, FFTMagnitude, FFTPhase, HIGHpass, INTegrate, INVert, LOWPass, MAGNify, MAXimum, MINimum, MULTiply, SMOoth, SUBTract, or VERSus.

<operand> Any allowable source for the selected FUNCtion, including channels, differential channels, common mode channels. waveform memories 1-4, functions 1-4, a constant, jitter measurement trend, and jitter spectrum. If the function is applied to a constant, the source returns the constant.

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example returns the currently defined source for function 1.

```
myScope.WriteString ":FUNCTION1?"
```

If the headers are off (see :SYSTem:HEADer), the query returns only the operands, not the operator.

```
myScope.WriteString ":SYST:HEAD ON"
myScope.WriteString ":FUNC1:ADD CHAN1,CHAN2"
myScope.WriteString ":FUNC1?"
strSettings = myScope.ReadString   ' Returns ":FUNC1:ADD CHAN1,CHAN2".
myScope.WriteString ":SYST:HEAD OFF"
myScope.WriteString ":FUNC1?"
strSettings = myScope.ReadString   ' Returns "CHAN1,CHAN2".
```

:FUNCtion<F>:ABSolute

Command `:FUNCtion<F>:ABSolute <operand>`

The :FUNCtion<F>:ABSolute command takes the absolute value an operand.

<operand> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIFF<P> | COMMonmode<P> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example turns on the absolute value command using channel 3.

```
myScope.WriteString ":FUNCtion1:ABSOLUTE CHANNEL3"
```

:FUNCtion<F>:ADD

Command `:FUNCtion<F>:ADD <operand>,<operand>`

The :FUNCtion<F>:ADD command defines a function that takes the algebraic sum of the two operands.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 1 to add channel 1 to channel 2.

```
myScope.WriteString ":FUNCTION1:ADD CHANNEL1,CHANNEL2"
```

:FUNCtion<F>:ADEMod

Command `:FUNCtion<F>:ADEMod <source>`

The :FUNCtion<F>:ADEMod command sets the math function to show the amplitude envelope for an amplitude modulated (AM) input signal.

This function uses a Hilbert transform to get the real (in-phase, I) and imaginary (quadrature, Q) parts of the input signal and then performs a square root of the sum of the real and imaginary parts to get the demodulated amplitude envelope waveform.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 1 to perform the amplitude demodulation function on channel 1.

```
myScope.WriteString ":FUNCtion1:ADEMod CHANnel1"
```

:FUNCtion<F>:AVERage

Command `:FUNCtion<F>:AVERage <operand>[,<averages>]`

The :FUNCtion<F>:AVERage command defines a function that averages the operand based on the number of specified averages.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

<averages> An integer, 2 to 65534 specifying the number of waveforms to be averaged

Example This example sets up function 1 to average channel 1 using 16 averages.

```
myScope.WriteString ":FUNCTION1:AVERAGE CHANNEL1,16"
```

:FUNCtion<F>:COMMonmode

Command `:FUNCtion<F>:COMMonmode <operand>,<operand>`

The :FUNCtion<F>:COMMonmode command defines a function that adds the voltage values of the two operands and divides by 2, point by point.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 1 to view the common mode voltage value of channel 1 and channel 2.

```
myScope.WriteString ":FUNCTION1:COMMONMODE CHANNEL1,CHANNEL2"
```

:FUNCtion<F>:DELay

(Delay)

Command `:FUNCtion<F>:DELay <operand>,<delay_time>`

The :FUNCtion<F>:DELay command adds the provided time to the X origin of the source waveform, effectively shifting the function waveform in time.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

<delay_time> Time, in seconds, set for the delay.

Example This example sets function 2 to be the waveform from channel1, delayed by 100 ps.

```
myScope.WriteString ":FUNCtion2:DELay CHANnel1,100E-12"
```

:FUNCtion<F>:DIFF

(Differentiate)

Command `:FUNCtion<F>:DIFF <operand>[,<low_pass_phase_align>]`

The :FUNCtion<F>:DIFF command defines a function that computes the discrete derivative of the operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

<low_pass_phase_align> {{ON | 1} | {OFF | 0}

This parameter turns on or off the low pass and phase align filter.

Example This example sets up function 2 to take the discrete derivative of the waveform on channel 2.

```
myScope.WriteString ":FUNCTION2:DIFF CHANNEL2"
```

:FUNCtion<F>:DISPlay

Command `:FUNCtion<F>:DISPlay {{ON|1} | {OFF|0}}`

The :FUNCtion<F>:DISPlay command either displays the selected function or removes it from the display.

<F> An integer, 1-16, representing the selected function.

Example This example turns function 1 on.

```
myScope.WriteString ":FUNCTION1:DISPLAY ON"
```

Query `:FUNCtion<F>:DISPlay?`

The :FUNCtion<F>:DISPlay? query returns the displayed status of the specified function.

Returned Format `[:FUNCtion<F>:DISPlay] {1|0}<NL>`

Example This example places the current state of function 1 in the variable, strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":FUNCTION1:DISPLAY?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:FUNCtion<F>:DIVide

Command `:FUNCtion<F>:DIVide <operand>,<operand>`

The :FUNCtion<F>:DIVide command defines a function that divides the first operand by the second operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 2 to divide the waveform on channel 1 by the waveform in waveform memory 4.

```
myScope.WriteString ":FUNCTION2:DIVIDE CHANNEL1,WMEMORY4"
```

:FUNCtion<F>:FFT:FREQuency

Command `:FUNCtion<F>:FFT:FREQuency <center_frequency_value>`

The :FUNCtion<F>:FFT:FREQuency command sets the center frequency for the FFT when :FUNCtion<F>:FFTMagnitude is defined for the selected function.

<F> An integer, 1-16, representing the selected function.

<center_frequency_value> A real number for the value in Hertz, from -1E12 to 1E12.

Query `:FUNCtion<F>:FFT:FREQuency?`

The :FUNCtion<F>:FFT:FREQuency? query returns the center frequency value.

Returned Format `[FUNCtion<F>:FFT:FREQuency] <center_frequency_value><NL>`

:FUNCtion<F>:FFT:REFerence

Command `:FUNCtion<F>:FFT:REFerence {DISPlay | TRIGger}`

The :FUNCtion<F>:FFT:REFerence command sets the reference point for calculating the FFT phase function.

<F> An integer, 1-16, representing the selected function.

Example This example sets the reference point to DISPlay.

```
myScope.WriteString ":FUNCTION1:FFT:REFERENCE DISPLAY"
```

Query `:FUNCtion<F>:FFT:REFerence?`

The :FUNCtion<F>:FFT:REFerence? query returns the currently selected reference point for the FFT phase function.

Returned Format `[:FUNCtion<F>:FFT:REFerence] {DISPlay | TRIGger}<NL>`

Example This example places the current state of the function 1 FFT reference point in the string variable, strREF, then prints the contents of the variable to the computer's screen.

```
Dim strREF As String
myScope.WriteString ":FUNCTION1:FFT:REFERENCE?"
strREF = myScope.ReadString
Debug.Print strREF
```

:FUNCtion<F>:FFT:RESolution?

Query `:FUNCtion<F>:FFT:RESolution?`

The :FUNCtion<F>:FFT:RESolution? query returns the current resolution of the FFT function.

Returned Format `[FUNCtion<F>:FFT:RESolution] <resolution_value><NL>`

<F> An integer, 1-16, representing the selected function.

<resolution_value> Resolution frequency.

The FFT resolution is determined by the sample rate and memory depth settings. The FFT resolution is calculated using the following equation:

$$\text{FFT Resolution} = \frac{\text{Sample Rate}}{\text{Effective Memory Depth}}$$

The effective memory depth is the highest power of 2 less than or equal to the number of sample points across the display. The memory bar in the status area at the top of the display indicates how much of the actual memory depth is across the display.

:FUNCtion<F>:FFT:TDELay

Command `:FUNCtion<F>:FFT:TDELay <time_delay>`

The :FUNCtion<F>:FFT:TDELay command sets the time delay for the FFT phase function.

<time_delay> Time, in seconds, set for the time delay.

Example This example sets the time delay to one millisecond.

```
myScope.WriteString ":FUNCtion1:FFT:TDELay 1E-3"
```

Query `:FUNCtion<F>:FFT:TDELay?`

The :FUNCtion<F>:FFT:TDELay? query returns the time delay for the FFT phase function.

Returned Format `[:FUNCtion<F>:FFT:TDELay] <time_delay><NL>`

Example This example places the FFT phase function's time delay value in the variable, varFftPhaseTimeDelay, then prints the contents of the variable to the computer's screen.

```
Dim varFftPhaseTimeDelay As Variant
myScope.WriteString ":FUNCtion1:FFT:TDELay?""
varFftPhaseTimeDelay = myScope.ReadNumber
Debug.Print FormatNumber(varFftPhaseTimeDelay, 0)
```

See Also

- ":FUNCtion<F>:FFTPhase" on page 345

:FUNCtion<F>:FFT:WINDow

Command `:FUNCtion<F>:FFT:WINDow {RECTangular | HANNing | FLATtop | BHARris | HAMMing}`

The :FUNCtion<F>:FFT:WINDow command sets the window type for the FFT function.

The FFT function assumes that the time record repeats. Unless there is an integral number of cycles of the sampled waveform in the record, a discontinuity is created at the beginning of the record. This introduces additional frequency components into the spectrum about the actual peaks, which is referred to as spectral leakage. To minimize spectral leakage, windows that approach zero smoothly at the beginning and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input waveforms.

- RECTangular – is essentially no window, and all points are multiplied by 1. This window is useful for transient waveforms and waveforms where there are an integral number of cycles in the time record.
- HANNing – 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.
- FLATtop – is best for making accurate amplitude measurements of frequency peaks.
- BHARris – (Blackman-Harris) is best used when you want to looks at signals with a strong interference component that is fairly distant from the frequency you want to see. It can be used as a general purpose window as its main lobe is not too wide (decent frequency discrimination) and the side lobes drop off by 90 dB.
- HAMMing –is a "raised cosine" function like the HANNing window but with different coefficients. It has slightly better frequency resolution than the HANNing window.

<F> An integer, 1-16, representing the selected function. This command presently selects all functions, regardless of which integer (1-16) is passed.

Example This example sets the window type for the FFT function to RECTangular.

```
myScope.WriteString ":FUNCtion1:FFT:WINDow RECTangular"
```

Query `:FUNCtion<F>:FFT:WINDow?`

The :FUNCtion<F>:FFT:WINDow? query returns the current selected window for the FFT function.

Returned Format `[:FUNCtion<F>:FFT:WINDow] {RECTangular | HANNing | FLATtop | BHARris | HAMMing}<NL>`

Example This example places the current state of the function 1 FFT window in the string variable, strWND, then prints the contents of the variable to the computer's screen.

```
Dim strWND As String
myScope.WriteString ":FUNCtion1:FFT:WINDow?""
strWND = myScope.ReadString
Debug.Print strWND
```

:FUNCtion<F>:FFTMagnitude

Command `:FUNCtion<F>:FFTMagnitude <operand>`

The :FUNCtion<F>:FFTMagnitude command computes the Fast Fourier Transform (FFT) of the specified channel, function, or memory. The FFT takes the digitized time record and transforms it to magnitude and phase components as a function of frequency.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 1 to compute the FFT of waveform memory 3.

```
myScope.WriteString ":FUNCTION1:FFTMAGNITUDE WMEMORY3"
```

:FUNCtion<F>:FFTPhase

Command `:FUNCtion<F>:FFTPhase <source>`

The :FUNCtion<F>:FFTPhase command computes the Fast Fourier Transform (FFT) of the specified channel, function, or waveform memory. The FFT takes the digitized time record and transforms it into magnitude and phase components as a function of frequency.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example sets up function 1 to compute the FFT of waveform memory 3.

```
myScope.WriteString ":FUNCTION1:FFTPHASE WMEMORY3"
```

See Also

- ":FUNCtion<F>:FFT:TDELay" on page 341

:FUNCtion<F>:GATing

(Gating)

Command `:FUNCtion<F>:GATing <operand>[,<gating_start>,<gating_stop>]`

The :FUNCtion<F>:GATing command defines a horizontal gating function of another waveform (similar to horizontal zoom). Measurements on horizontal gating functions are essentially gated measurements.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

<gating_start> Time, in seconds, relative to the source waveform that specifies where the gating window begins.

<gating_stop> Time, in seconds, relative to the source waveform that specifies where the gating window ends.

Example This example sets function 4 to be a horizontal gating of the channel1 waveform beginning at -8 ns and ending at -5 ns.

```
myScope.WriteString ":FUNCtion4:GATing CHANnel1,-8E-9,-5E-9"
```

:FUNCtion<F>:HIGHpass

Command `:FUNCtion<F>:HIGHpass <source>,<bandwidth>`

The :FUNCtion<F>:HIGHpass command applies a single-pole high pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

<bandwidth> A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a high pass filter with a bandwidth of 1 MHz.

```
myScope.WriteString ":FUNCTION2:HIGHPASS CHANNEL4,1E6"
```

:FUNCtion<F>:HORizontal

Command `:FUNCtion<F>:HORizontal {AUTO | MANual}`

The :FUNCtion<F>:HORizontal command sets the horizontal tracking to either AUTO or MANual.

The HORizontal command also includes the following commands and queries, which are described on the following pages:

- POSition
- RANGe

<F> An integer, 1-16, representing the selected function.

Query `:FUNCtion<F>:HORizontal?`

The :FUNCtion<F>:HORizontal? query returns the current horizontal scaling mode of the specified function.

Returned Format `[:FUNCtion<F>:HORizontal] {AUTO | MANual}<NL>`

Example This example places the current state of the function 1 horizontal tracking in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":FUNCTION1:HORIZONTAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:FUNCtion<F>:HORizontal:POSition

Command `:FUNCtion<F>:HORizontal:POSition <position_value>`

The :FUNCtion<F>:HORizontal:POSition command sets the time value at center screen for the selected function. If the oscilloscope is not already in manual mode when you execute this command, it puts the oscilloscope in manual mode.

When you select :FUNCtion<F>:FFTMagnitude, the horizontal position is equivalent to the center frequency. This also automatically selects manual mode.

<F> An integer, 1-16, representing the selected function.

<position_value> A real number for the position value in time, in seconds, from -10E15 to 10E15.

Query `:FUNCtion<F>:HORizontal:POSition?`

The :FUNCtion<F>:HORizontal:POSition? query returns the current time value at center screen of the selected function.

Returned Format `[:FUNCtion<F>:HORizontal:POSition] <position><NL>`

Example This example places the current horizontal position setting for function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:HORIZONTAL:POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:FUNCtion<F>:HORizontal:RANGe

Command `:FUNCtion<F>:HORizontal:RANGe <range_value>`

The :FUNCtion<F>:HORizontal:RANGe command sets the current time range for the specified function. This automatically selects manual mode.

<F> An integer, 1-16, representing the selected function.

<range_value> A real number for the width of screen in current X-axis units (usually seconds), from -100E-15 to 100E15.

Query `:FUNCtion<F>:HORizontal:RANGe?`

The :FUNCtion<F>:HORizontal:RANGe? query returns the current time range setting of the specified function.

Returned Format `[:FUNCtion<F>:HORizontal:RANGe] <range><NL>`

Example This example places the current horizontal range setting of function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:HORIZONTAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:FUNCtion<F>:INTegrate

Command `:FUNCtion<F>:INTegrate <operand>`

The :FUNCtion<F>:INTegrate command defines a function that computes the integral of the specified operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example sets up function 1 to compute the integral of waveform memory 3.

```
myScope.WriteString ":FUNCTION1:INTEGRATE WMEMORY3"
```

:FUNCtion<F>:INVert

Command `:FUNCtion<F>:INVert <operand>`

The :FUNCtion<F>:INVert command defines a function that inverts the defined operand's waveform by multiplying by -1.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example sets up function 2 to invert the waveform on channel 1.

```
myScope.WriteString ":FUNCTION2:INVERT CHANNEL1"
```

:FUNCtion<F>:LOWPass

Command `:FUNCtion<F>:LOWPass <source>,<bandwidth>`

The :FUNCtion<F>:LOWPass command applies a 4th order Bessel-Thompson pass filter to the source waveform. The bandwidth that you set is the 3 dB bandwidth of the filter.

<F> An integer, 1-16, representing the selected function.

<source> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

<bandwidth> A real number in the range of 50 to 50E9.

Example This example sets up function 2 to compute a low pass filter with a bandwidth of 1 MHz.

```
myScope.WriteString ":FUNCTION2:LOWPASS CHANNEL4,1E6"
```

:FUNCtion<F>:MAGNify

Command `:FUNCtion<F>:MAGNify <operand>`

The :FUNCtion<F>:MAGNify command defines a function that is a copy of the operand. The magnify function is a software magnify. No hardware settings are altered as a result of using this function. It is useful for scaling channels, another function, or memories with the RANGe and OFFSet commands in this subsystem.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example creates a function (function 1) that is a magnified version of channel 1.

```
myScope.WriteString ":FUNCTION1:MAGNIFY CHANNEL1"
```

:FUNCtion<F>:MAXimum

Command `:FUNCtion<F>:MAXimum <operand>`

The :FUNCtion<F>:MAXmum command defines a function that computes the maximum of each time bucket for the defined operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example sets up function 2 to compute the maximum of each time bucket for channel 4.

```
myScope.WriteString ":FUNCTION2:MAXIMUM CHANNEL4"
```

:FUNCtion<F>:MHIStogram

Command `:FUNCtion<F>:MHIStogram {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}`

The :FUNCtion<F>:MHIStogram command adds a Meas Histogram function that shows a histogram of measurement values. Measurement values are captured and the histogram is updated as new acquisitions are made.

You can display statistics for the histogram in the Measurements tab using the :MEASure:HISTogram commands and you can get histogram statistics using the :MEASure:HISTogram queries.

<F> An integer, 1-16, representing the selected function.

Example This example sets up a histogram function of the first measurement.

```
myScope.WriteString ":FUNCtion2:MHIStogram MEAS1"
```

See Also

- ":MEASure:HISTogram:HITS" on page 550
- ":MEASure:HISTogram:M1S" on page 551
- ":MEASure:HISTogram:M2S" on page 552
- ":MEASure:HISTogram:M3S" on page 553
- ":MEASure:HISTogram:MAX" on page 554
- ":MEASure:HISTogram:MEAN" on page 555
- ":MEASure:HISTogram:MEDian" on page 556
- ":MEASure:HISTogram:MIN" on page 557
- ":MEASure:HISTogram:MODE" on page 558
- ":MEASure:HISTogram:PEAK" on page 559
- ":MEASure:HISTogram:PP" on page 560
- ":MEASure:HISTogram:RESolution" on page 561
- ":MEASure:HISTogram:STDDev" on page 562

:FUNCtion<F>:MINimum

Command `:FUNCtion<F>:MINimum <operand>`

The :FUNCtion<F>:MINimum command defines a function that computes the minimum of each time bucket for the defined operand's waveform.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example sets up function 2 to compute the minimum of each time bucket for channel 4.

```
myScope.WriteString ":FUNCTION2:MINIMUM CHANNEL4"
```

:FUNCtion<F>:MTRend

Command `:FUNCtion<F>:MTRend {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}`

The :FUNCtion<F>:MTRend command adds a Meas Trend function that shows measurement values for a waveform (based on measurement threshold settings) as the waveform progresses across the screen. For every cycle, a measurement is made, and the value is displayed on the screen for the cycle.

If a measurement cannot be made for part of a waveform, the trend function output is a hole (that is, no value) until a measurement can be made.

<F> An integer, 1-16, representing the selected function.

Example This example sets up a trend function of the first measurement.

```
myScope.WriteString ":FUNCtion2:MTRend MEAS1"
```

:FUNCtion<F>:MULTiply

Command `:FUNCtion<F>:MULTiply <operand>,<operand>`

The :FUNCtion<F>:MULTiply command defines a function that algebraically multiplies the first operand by the second operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example defines a function that multiplies channel 1 by waveform memory 1.

```
myScope.WriteString ":FUNCTION1:MULTIPLY CHANNEL1,WMEMORY1"
```

:FUNCtion<F>:OFFSet

Command `:FUNCtion<F>:OFFSet <offset_value>`

The :FUNCtion<F>:OFFSet command sets the voltage represented at the center of the screen for the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<offset_value> A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited to being within the vertical range that can be represented by the function data.

Example This example sets the offset voltage for function 1 to 2 mV.

```
myScope.WriteString ":FUNCTION1:OFFSET 2E-3"
```

Query `:FUNCtion<F>:OFFSet?`

The :FUNCtion<F>:OFFSet? query returns the current offset value for the selected function.

Returned Format `[:FUNCtion<F>:OFFSet] <offset_value><NL>`

Example This example places the current setting for offset on function 2 in the numeric variable, varValue, then prints the result to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:FUNCtion<F>:RANGe

Command `:FUNCtion<F>:RANGe <full_scale_range>`

The :FUNCtion<F>:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<full_scale_range> A real number for the full-scale vertical range, from -100E15 to 100E15.

Example This example sets the full-scale range for function 1 to 400 mV.

```
myScope.WriteString ":FUNCTION1:RANGE 400E-3"
```

Query `:FUNCtion<F>:RANGe?`

The :FUNCtion<F>:RANGe? query returns the current full-scale range setting for the specified function.

Returned Format `[:FUNCtion<F>:RANGe] <full_scale_range><NL>`

Example This example places the current range setting for function 2 in the numeric variable "varValue", then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:FUNCtion<F>:SMOoth

Command `:FUNCtion<F>:SMOoth <operand>[,<points>]`

The :FUNCtion<F>:SMOoth command defines a function that assigns the smoothing operator to the operand with the number of specified smoothing points.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

<points> An integer, odd numbers from 3 to 4001 specifying the number of smoothing points.

Example This example sets up function 1 using assigning smoothing operator to channel 1 using 5 smoothing points.

```
myScope.WriteString ":FUNCTION1:SMOOTH CHANNEL1,5"
```

:FUNCtion<F>:SQRT

Command `:FUNCtion<F>:SQRT <operand>`

The :FUNCtion<F>:SQRT command takes the square root of the operand.

<operand> {CHANnel<N> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<N> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example turns on the square root function using channel 3.

```
myScope.WriteString ":FUNCtion1:SQRT CHANNEL3"
```

:FUNCtion<F>:SQUare

Command `:FUNCtion<F>:SQUare <operand>`

The :FUNCtion<F>:SQUare command takes the square value of the operand.

<operand> {CHANnel<N> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<N> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, “Function Commands,” starting on page 325.

Example This example turns on the square value command using channel 3.

```
myScope.WriteString ":FUNCtion1:SQUARE CHANNEL3"
```

:FUNCtion<F>:SUBTract

Command `:FUNCtion<F>:SUBTract <operand>,<operand>`

The :FUNCtion<F>:SUBTract command defines a function that algebraically subtracts the second operand from the first operand.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example defines a function that subtracts waveform memory 1 from channel 1.

```
myScope.WriteString ":FUNCTION1:SUBTRACT CHANNEL1,WMEMORY1"
```

:FUNCtion<F>:VERSus

Command `:FUNCtion<F>:VERSus <operand>,<operand>`

The :FUNCtion<F>:VERSus command defines a function for an X-versus-Y display. The first operand defines the Y axis and the second defines the X axis. The Y-axis range and offset are initially equal to that of the first operand, and you can adjust them with the RANGe and OFFSet commands in this subsystem.

<F> An integer, 1-16, representing the selected function.

<operand> {CHANnel<n> | DIFF<P> | COMMonmode<P> | FUNCtion<F> | WMEMory<n> | <float_value> | MTRend | MSPectrum}

See the discussion of possible operands in the introduction to Chapter 17, "Function Commands," starting on page 325.

Example This example defines function 1 as an X-versus-Y display. Channel 1 is the X axis and waveform memory 2 is the Y axis.

```
myScope.WriteString ":FUNCTION1:VERSUS WMEMORY2,CHANNEL1"
```

:FUNCtion<F>:VERTical

Command `:FUNCtion<F>:VERTical {AUTO | MANual}`

The :FUNCtion<F>:VERTical command sets the vertical scaling mode of the specified function to either AUTO or MANual.

This command also contains the following commands and queries:

- OFFset
- RANge

<F> An integer, 1-16, representing the selected function.

Query `:FUNCtion<F>:VERTical?`

The :FUNCtion<F>:VERTical? query returns the current vertical scaling mode of the specified function.

Returned Format `[:FUNCtion<F>:VERTical] {AUTO | MANual}<NL>`

Example This example places the current state of the vertical tracking of function 1 in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":FUNCTION1:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:FUNCtion<F>:VERTical:OFFSet

Command `:FUNCtion<F>:VERTical:OFFSet <offset_value>`

The :FUNCtion<F>:VERTical:OFFSet command sets the voltage represented at center screen for the selected function. This automatically changes the mode from auto to manual.

<F> An integer, 1-16, representing the selected function.

<offset_value> A real number for the vertical offset in the currently selected Y-axis units (normally volts). The offset value is limited only to being within the vertical range that can be represented by the function data.

Query `:FUNCtion<F>:VERTical:OFFset?`

The :FUNCtion<F>:VERTical:OFFSet? query returns the current offset value of the selected function.

Returned Format `[:FUNCtion<F>:VERTical:OFFset] <offset_value><NL>`

Example This example places the current offset setting for function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:FUNCtion<F>:VERTical:RANGe

Command `:FUNCtion<F>:VERTical:RANGe <full_scale_range>`

The :FUNCtion<F>:VERTical:RANGe command defines the full-scale vertical axis of the selected function. This automatically changes the mode from auto to manual, if the oscilloscope is not already in manual mode.

<F> An integer, 1-16, representing the selected function.

<full_scale_range> A real number for the full-scale vertical range, from -100E15 to 100E15.

Query `:FUNCtion<F>:VERTical:RANGe?`

The :FUNCtion<F>:VERTical:RANGe? query returns the current range setting of the specified function.

Returned Format `[:FUNCtion<F>:VERTical:RANGe] <range><NL>`

Example This example places the current vertical range setting of function 2 in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":FUNCTION2:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

18 Hardcopy Commands

:HARDcopy:AREA 372
:HARDcopy:DPRinter 373
:HARDcopy:FACTors 374
:HARDcopy:IMAGe 375
:HARDcopy:PRINters? 376

The HARDcopy subsystem commands set various parameters for printing the screen. The print sequence is activated when the root level command :PRINt is sent.

:HARDcopy:AREA

Command `:HARDcopy:AREA {GRATicule | SCReen}`

The :HARDcopy:AREA command selects which data from the screen is to be printed. When you select GRATicule, only the graticule area of the screen is printed (this is the same as choosing Waveforms Only in the Configure Printer dialog box). When you select SCReen, the entire screen is printed.

Example This example selects the graticule for printing.

```
myScope.WriteString ":HARDCOPY:AREA GRATICULE"
```

Query `:HARDcopy:AREA?`

The :HARDcopy:AREA? query returns the current setting for the area of the screen to be printed.

Returned Format `[:HARDcopy:AREA] {GRATicule | SCReen}<NL>`

Example This example places the current selection for the area to be printed in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":HARDCOPY:AREA?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:HARDcopy:DPRinter

Command `:HARDcopy:DPRinter {<printer_number> | <printer_string>}`

The :HARDcopy:DPRinter command selects the default printer to be used.

<printer _number> An integer representing the attached printer. This number corresponds to the number returned with each printer name by the :HARDcopy:PRINters? query.

<printer _string> A string of alphanumeric characters representing the attached printer.

The :HARDcopy:DPRinter command specifies a number or string for the printer attached to the oscilloscope. The printer string must exactly match the character strings in the File->Print Setup dialog boxes, or the strings returned by the :HARDcopy:PRINters? query.

Examples This example sets the default printer to the second installed printer returned by the :HARDcopy:PRINters? query.

```
myScope.WriteString ":HARDCOPY:DPRINTER 2"
```

This example sets the default printer to the installed printer with the name "HP Laser".

```
myScope.WriteString ":HARDCOPY:DPRINTER ""HP Laser"""
```

Query `:HARDcopy:DPRinter?`

The :HARDcopy:DPRinter? query returns the current printer number and string.

Returned Format `[:HARDcopy:DPRinter?] {<printer_number>,<printer_string>,DEFAULT}<NL>`

Or, if there is no default printer (no printers are installed), only a <NL> is returned.

Example This example places the current setting for the hard copy printer in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDCOPY:DPRinter?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

NOTE It takes several seconds to change the default printer. Any programs that try to set the default printer must wait (10 seconds is a safe amount of time) for the change to complete before sending other commands. Otherwise, the oscilloscope will become unresponsive.

:HARDcopy:FACTors

Command `:HARDcopy:FACTors {{ON | 1} | {OFF | 0}}`

The :HARDcopy:FACTors command determines whether the oscilloscope setup factors will be appended to screen or graticule images. FACTors ON is the same as choosing Include Setup Information in the Configure Printer dialog box.

Example This example turns on the setup factors.

```
myScope.WriteString ":HARDCOPY:FACTORS ON"
```

Query `:HARDcopy:FACTors?`

The :HARDcopy:FACTors? query returns the current setup factors setting.

Returned Format `[:HARDcopy:FACTors] {1 | 0}<NL>`

Example This example places the current setting for the setup factors in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDCOPY:FACTORS?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:HARDcopy:IMAGe

Command `:HARDcopy:IMAGe {NORMal | INVert}`

The :HARDcopy:IMAGe command prints the image normally, inverted, or in monochrome. IMAGe INVert is the same as choosing Invert Waveform Colors in the Configure Printer dialog box.

Example This example sets the hard copy image output to normal.

```
myScope.WriteString ":HARDCOPY:IMAGE NORMAL"
```

Query `:HARDcopy:IMAGe?`

The :HARDcopy:IMAGe? query returns the current image setting.

Returned Format `[:HARDcopy:IMAGe] {NORMal | INVert}<NL>`

Example This example places the current setting for the hard copy image in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":HARDCOPY:IMAGE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:HARDcopy:PRINters?

Query `:HARDcopy:PRINters?`

The :HARDcopy:PRINters? query returns the currently available printers.

Returned Format

```
[:HARDcopy:PRINters?]
<printer_count><NL><printer_data><NL>[,<printer_data><NL>]
```

<printer_count> The number of printers currently installed.

<printer _data> The printer number and the name of an installed printer. The word DEFAULT appears next to the printer that is the currently selected default printer.

The <printer_data> return string has the following format: <printer_number>,<printer_string>{,DEFAULT}

Example This example places the number of installed printers into the variable varCount, loops through it that number of times, and prints the installed printer names to the computer's screen.

```
Dim varResults As Variant
Dim lngI As Long

myScope.WriteString ":HARDcopy:PRINters?"
varResults = myScope.ReadList(ASCIIType_BSTR, vbLf)
Debug.Print FormatNumber(varResults(0), 0)

For lngI = 1 To varResults(0)
  Debug.Print CStr(varResults(lngI))
Next lngI
```

19 Histogram Commands

:HISTogram:AXIS 379
:HISTogram:MODE 380
:HISTogram:SCALe:SIZE 381
:HISTogram:WINDow:DEFault 382
:HISTogram:WINDow:SOURce 383
:HISTogram:WINDow:LLIMit 384
:HISTogram:WINDow:RLIMit 385
:HISTogram:WINDow:BLIMit 386
:HISTogram:WINDow:TLIMit 387

The HISTogram commands and queries control the histogram features. A histogram is a probability distribution that shows the distribution of acquired data within a user-definable histogram window.

You can display the histogram either vertically, for voltage measurements, or horizontally, for timing measurements.

The most common use for histograms is measuring and characterizing noise or jitter on displayed waveforms. Noise is measured by sizing the histogram window to a narrow portion of time and observing a vertical histogram that measures the noise on a waveform. Jitter is measured by sizing the histogram window to a narrow portion of voltage and observing a horizontal histogram that measures the jitter on an edge.

Histograms and the database

The histograms, mask testing, and color grade persistence use a specific database that uses a different memory area from the waveform record for each channel. When any of these features are turned on, the oscilloscope starts building the database. The database is the size of the graticule area. Behind each pixel is a 21-bit counter that is incremented each time data from a channel or function hits a pixel. The maximum count (saturation) for each counter is 2,097,151. You can use the DISPlay:CGRade:LEVels command to see if any of the counters are close to saturation.

The database continues to build until the oscilloscope stops acquiring data or all both features (color grade persistence and histograms) are turned off. You can clear the database by turning off all three features that use the database.

The database does not differentiate waveforms from different channels or functions. If three channels are on and the waveform from each channel happens to light the same pixel at the same time, the counter is incremented by three. However, it is not possible to tell how many hits came from each waveform. To separate waveforms, you can position the waveforms vertically with the channel offset. By separating the waveforms, you can avoid overlapping data in the database caused by multiple waveforms. Even if the display is set to show only the most recent acquisition, the database keeps track of all pixel hits while the database is building.

Remember that color grade persistence, mask testing, and histograms all use the same database. Suppose that the database is building because color grade persistence is ON; when mask testing or histograms are turned on, they can use the information already established in the database as though they had been turned on the entire time.

To avoid erroneous data, clear the display after you change oscilloscope setup conditions or DUT conditions and acquire new data before extracting measurement results.

:HISTogram:AXIS

Command `:HISTogram:AXIS {VERTical | HORizontal}`

The :HISTogram:AXIS command selects the type of histogram. A horizontal histogram can be used to measure time related information like jitter. A vertical histogram can be used to measure voltage related information like noise.

Example This example defines a vertical histogram.

```
myScope.WriteString ":HISTOGRAM:AXIS VERTICAL"
```

Query `:HISTogram:AXIS?`

The :HISTogram:AXIS? query returns the currently selected histogram type.

Returned Format `[:HISTogram:AXIS] {VERTical | HORizontal}<NL>`

Example This example returns the histogram type and prints it to the computer's screen.

```
Dim strAxis As String
myScope.WriteString ":HISTOGRAM:AXIS?"
strAxis = myScope.ReadString
Debug.Print strAxis
```

:HISTogram:MODE

Command `:HISTogram:MODE {OFF | MEASurement | WAVeforms}`

NOTE The MEASurement parameter is only available when the E2681A Jitter Analysis option is installed.

The :HISTogram:MODE command selects the histogram mode. The histogram may be off, set to track the waveforms, or set to track the measurement when the E2681A Jitter Analysis Software is installed. When the E2681A Jitter Analysis Software is installed, sending the :MEASure:JITTer:HISTogram ON command will automatically set :HISTOgram:MODE to MEASurement.

Example This example sets the histogram mode to track the waveform.

```
myScope.WriteString ":HISTOGRAM:MODE WAVEFORM"
```

Query `:HISTogram:MODE?`

The :HISTogram:MODE? query returns the currently selected histogram mode.

Returned Format `[:HISTogram:MODE] {OFF | MEASurement | WAVeform}<NL>`

Example This example returns the result of the mode query and prints it to the computer's screen.

```
Dim strMode As String
myScope.WriteString ":HISTOGRAM:MODE?"
strMode = myScope.ReadString
Debug.Print strMode
```

:HISTogram:SCALe:SIZE

Command `:HISTogram:SCALe:SIZE <size>`

The :HISTogram:SCALe:SIZE command sets histogram size for vertical and horizontal mode.

<size> The size is from 1.0 to 8.0 for the horizontal mode and from 1.0 to 10.0 for the vertical mode.

Example This example sets the histogram size to 3.5.

```
myScope.WriteString ":HISTOGRAM:SCALE:SIZE 3.5"
```

Query `:HISTogram:SCALe:SIZE?`

The :HISTogram:SCALe:SIZE? query returns the correct size of the histogram.

Returned Format `[:HISTogram:SCALe:SIZE] <size><NL>`

Example This example returns the result of the size query and prints it to the computer's screen.

```
Dim strSize As String
myScope.WriteString ":HISTOGRAM:SCALE:SIZE?"
strSize = myScope.ReadString
Debug.Print strSize
```

:HISTogram:WINDow:DEFault

Command `:HISTogram:WINDow:DEFault`

The :HISTogram:WINDow:DEFault command positions the histogram markers to a default location on the display. Each marker will be positioned one division off the left, right, top, and bottom of the display.

Example This example sets the histogram window to the default position.

```
myScope.WriteString ":HISTOGRAM:WINDOW:DEFAULT"
```

:HISTogram:WINDow:SOURce

Command `:HISTogram:WINDow:SOURce {CHANnel<N> | FUNCtion<F> | WMEMory<N>}`

The :HISTogram:WINDow:SOURce command selects the source of the histogram window. The histogram window will track the source's vertical and horizontal scale.

<N> An integer, 1-4.

<F> An integer, 1-16.

Example This example sets the histogram window's source to Channel 1.

```
myScope.WriteString ":HISTOGRAM:WINDOW:SOURCE CHANNEL1"
```

Query `:HISTogram:WINDow:SOURce?`

The :HISTogram:WINDow:SOURce? query returns the currently selected histogram window source.

Returned Format `[:HISTogram:WINDow:SOURce] {CHANnel<N> | FUNCtion<F> | WMEMory<N>}<NL>`

Example This example returns the result of the window source query and prints it to the computer's screen.

```
Dim strWinsour As String
myScope.WriteString ":HISTOGRAM:WINDOW:SOURCE?"
strWinsour = myScope.ReadString
Debug.Print strWinsour
```

:HISTogram:WINDow:LLIMit

Command `:HISTogram:WINDow:LLIMit <left_limit>`

The :HISTogram:WINDow:LLIMit command moves the Ax marker (left limit) of the histogram window. The histogram window determines the portion of the display used to build the database for the histogram. The histogram window markers will track the scale of the histogram window source.

<left_limit> A real number that represents the left boundary of the histogram window.

Example This example sets the left limit position to -200 microseconds.

```
myScope.WriteString ":HISTOGRAM:WINDOW:LLIMit -200E-6"
```

Query `:HISTogram:WINDow:LLIMit?`

The :HISTogram:WINDow:LLIMit? query returns the value of the left limit histogram window marker.

Returned Format `[:HISTogram:WINDow:LLIMit] <left_limit><NL>`

Example This example returns the result of the left limit position query and prints it to the computer's screen.

```
Dim strLL As String
myScope.WriteString ":HISTOGRAM:WINDOW:LLIMIT?"
strLL = myScope.ReadString
Debug.Print strLL
```

:HISTogram:WINDow:RLIMit

Command `:HISTogram:WINDow:RLIMit <right_limit>`

The :HISTogram:WINDow:RLIMit command moves the Bx marker (right limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<right_limit> A real number that represents the right boundary of the histogram window.

Example This example sets the Bx marker to 200 microseconds.

```
myScope.WriteString ":HISTOGRAM:WINDOW:RLIMit 200E-6"
```

Query `:HISTogram:WINDow:RLIMit?`

The :HISTogram:WINDow:RLIMit? query returns the value of the right histogram window marker.

Returned Format `[:HISTogram:WINDow:RLIMit] <right_limit><NL>`

Example This example returns the result of the Bx position query and prints it to the computer's screen.

```
Dim strRL As String
myScope.WriteString ":HISTOGRAM:WINDOW:RLIMit?"
strRL = myScope.ReadString
Debug.Print strRL
```

:HISTogram:WINDow:BLIMit

Command `:HISTogram:WINDow:BLIMit <bottom_limit>`

The :HISTogram:WINDow:BLIMit command moves the Ay marker (bottom limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<bottom_limit> A real number that represents the bottom boundary of the histogram window.

Example This example sets the position of the Ay marker to -250 mV.

```
myScope.WriteString ":HISTOGRAM:WINDOW:BLIMit -250E-3"
```

Query `:HISTogram:WINDow:BLIMit?`

The :HISTogram:WINDow:BLIMit? query returns the value of the Ay histogram window marker.

Returned Format `[:HISTogram:WINDow:BLIMit] <bottom_limit><NL>`

Example This example returns the result of the Ay position query and prints it to the computer's screen.

```
Dim strBL As String
myScope.WriteString ":HISTOGRAM:WINDOW:BLIMit?"
strBL = myScope.ReadString
Debug.Print strBL
```

:HISTogram:WINDow:TLIMit

Command `:HISTogram:WINDow:TLIMit <top_limit>`

The :HISTogram:WINDow:TLIMit command moves the By marker (top limit) of the histogram window. The histogram window determines the portion of the display used to build the database used for the histogram. The histogram window markers will track the scale of the histogram window source.

<top_limit> A real number that represents the top boundary of the histogram window.

Example This example sets the position of the By marker to 250 mV.

```
myScope.WriteString ":HISTOGRAM:WINDOW:TLIMit 250E-3"
```

Query `:HISTogram:WINDow:TLIMit?`

The :HISTogram:WINDow:TLIMit? query returns the value of the By histogram window marker.

Returned Format `[:HISTogram:WINDow:TLIMit] <top_limit><NL>`

Example This example returns the result of the By position query and prints it to the computer's screen.

```
Dim strTL As String
myScope.WriteString ":HISTOGRAM:WINDOW:TLIMit?"
strTL = myScope.ReadString
Debug.Print strTL
```

20 InfiniiScan (ISCan) Commands

:ISCan:DELay 390
:ISCan:MEASurement:FAIL 391
:ISCan:MEASurement:LLIMit 392
:ISCan:MEASurement 393
:ISCan:MEASurement:ULIMit 394
:ISCan:MODE 395
:ISCan:NONMonotonic:EDGE 396
:ISCan:NONMonotonic:HYSTeresis 397
:ISCan:NONMonotonic:SOURce 398
:ISCan:RUNT:HYSTeresis 399
:ISCan:RUNT:LLEVel 400
:ISCan:RUNT:SOURce 401
:ISCan:RUNT:ULEVel 402
:ISCan:SERial:PATTern 403
:ISCan:SERial:SOURce 404
:ISCan:ZONE:HIDE 405
:ISCan:ZONE:SOURce 406
:ISCan:ZONE<N>:MODE 407
:ISCan:ZONE<N>:PLACement 408
:ISCan:ZONE<N>:STATe 409

The ISCan commands and queries control the InfiniiScan feature of the oscilloscope. InfiniiScan provides several ways of searching through the waveform data to find unique events.

:ISCan:DELay

Command `:ISCan:DELay {OFF | <delay_time>}`

The :ISCan:DELay command sets the delay time from when the hardware trigger occurs and when InfiniiScan tries to find the waveform event that has been defined.

OFF Turns off the delay from the hardware trigger.

<delay_time> Sets the amount of time that the InfiniiScan trigger is delayed from the hardware trigger.

Example The following example causes the oscilloscope to delay by 1 ms.

```
myScope.WriteString ":ISCAN:DELay 1E-06"
```

Query `:ISCan:DELay?`

The query returns the current set delay value.

Returned Format `[:ISCan:DELay] {OFF | <delay_time>}<NL>`

Example The following example returns the current delay value and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:DELAY?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:MEASurement:FAIL

Command `:ISCan:MEASurement:FAIL {INSide | OUTSide}`

The :ISCan:MEASurement:FAIL command sets the fail condition for an individual measurement. The conditions for a test failure are set on the measurement selected by the :ISCan:MEASurement command.

When a measurement failure is detected by the limit test the oscilloscope triggers and the trigger action is executed.

INSide INside causes the oscilloscope to fail a test when the measurement results are within the parameters set by the :ISCan:MEASurement:LIMit and :ISCan:MEASurement:ULIMit commands.

OUTSide OUTside causes the oscilloscope to fail a test when the measurement results exceed the parameters set by the :ISCan:MEASurement:LLIMit and the :ISCan:MEASurement:ULIMit commands.

Example The following example causes the oscilloscope to trigger when the measurements are outside the lower or upper limits.

```
myScope.WriteString ":ISCAN:MEASUREMENT:FAIL OUTSIDE"
```

Query `:ISCan:MEASurement:FAIL?`

The query returns the current set fail condition.

Returned Format `[:ISCan:MEASurement:FAIL] {INSide | OUTSide}<NL>`

Example The following example returns the current fail condition and prints the result to the controller's screen.

```
Dim strFAIL As String
myScope.WriteString ":ISCAN:MEASUREMENT:FAIL?"
strFAIL = myScope.ReadString
Debug.Print strFAIL
```

:ISCan:MEASurement:LLIMit

Command `:ISCan:MEASurement:LLIMit <lower_value>`

The :ISCan:MEASurement:LLIMit (lower limit) command sets the lower test limit for the currently selected measurement. The :ISCan:MEASurement command selects the measurement used.

<lower_value> A real number.

Example The following example sets the lower test limit to 1.0.

```
myScope.WriteString ":ISCAN:MEASUREMENT:LLIMIT 1.0"
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the measurement limit to trigger when the signal is outside the specified limit.

Query `:ISCan:MEASurement:LLIMit?`

The query returns the current value set by the command.

Returned Format `[:ISCan:MEASurement:LLIMit]<lower_value><NL>`

Example The following example returns the current lower test limit and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:MEASUREMENT:LLIMIT?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:MEASurement

Command `:ISCan:MEASurement {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}`

The :ISCan:MEASurement command selects the current source for Measurement Limit Test Trigger. It selects one of the active measurements as referred to by their position in the Measurement tab area at the bottom of the screen. Measurements are numbered from left to right in the Measurements tab area of the screen.

Example The following example selects the first measurement as the source for the limit testing commands.

```
myScope.WriteString ":ISCAN:MEASUREMENT MEAS1"
```

Query `:ISCan:MEASurement?`

The query returns the currently selected measurement source.

Returned Format `[:ISCan:MEASurement]{MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5} <NL>`

Example The following example returns the currently selected measurement source for the limit testing commands.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:MEASUREMENT?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also Measurements are started by the commands in the Measurement Subsystem.

:ISCan:MEASurement:ULIMit

Command `:ISCan:MEASurement:ULIMit <upper_value>`

The :ISCan:MEASurement:ULIMit (upper limit) command sets the upper test limit for the active measurement currently selected by the :ISCan:MEASurement command.

<upper_value> A real number.

Example The following example sets the upper limit of the currently selected measurement to 500 mV.

```
myScope.WriteString ":ISCAN:MEASUREMENT:ULIMIT 500E-3"
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the :ISCan:MEASurement:FAIL OUTside command to specify that the oscilloscope will trigger when the voltage exceeds 500 mV.

Query `:ISCan:MEASurement:ULIMit?`

The query returns the current upper limit of the limit test.

Returned Format `[:ISCan:MEASurement:ULIMit] <upper_value><NL>`

Example The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:MEASUREMENT:ULIMit?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:MODE

Command `:ISCan:MODE {OFF | MEASurement | NONMonotonic | RUNT | SERial | ZONE}`

The :ISCan:MODE command selects the type of InfiniiScan trigger mode. The Measurement, Runt, Zone Qualify, and Non-monotonic Edge InfiniiScan modes can be set using this command.

OFF Turns off the InfiniiScan trigger mode.

MEASurement Sets the Measurement Limit trigger mode.

NONMonotonic Sets the Non-monotonic edge trigger mode.

RUNT Sets the Runt trigger mode.

SERial Sets the Serial trigger mode.

ZONE Sets the Zone Qualify trigger mode.

Example The following example selects the runt trigger.

```
myScope.WriteString ":ISCAN:MODE RUNT"
```

Query `:ISCan:MODE?`

The query returns the currently selected IniniiScan trigger mode.

Returned Format `[:ISCan:MEASurement]{OFF | MEASurement | NONMonotonic | RUNT | SERial | ZONE}<NL>`

Example The following example returns the currently selected InfiniiScan trigger mode.

```
Dim strMODE As String
myScope.WriteString ":ISCAN:MODE?"
strMODE = myScope.ReadString
Debug.Print strMODE
```

:ISCan:NONMonotonic:EDGE

Command `:ISCan:NONMonotonic:EDGE {EITHer | FALLing | RISing}`

The :ISCan:NONMonotonic:EDGE command selects the rising edge, the falling edge, or either edge for the Non-monotonic edge trigger mode.

EITHer Sets the edge used by the Non-monotonic edge trigger to both rising and falling edges.

FALLing Sets the edge used by the Non-monotonic edge trigger to falling edges.

RISing Sets the edge used by the Non-monotonic edge trigger to rising edges.

Example The following example selects the falling edge non-monotonic trigger.

```
myScope.WriteString ":ISCAN:NONMONOTONIC:EDGE FALLING"
```

Query `:ISCan:NONMonotonic:EDGE?`

The query returns the currently selected edge type for the Non-Monotonic Edge trigger.

Returned Format `[:ISCan:NONMonotonic:EDGE]{EITHer | FALLing | RISing}<NL>`

Example The following example returns the currently selected edge type used for the Non-monotonic Edge trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:NONMONOTONIC:EDGE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

:ISCan:NONMonotonic:HYSTeresis

Command `:ISCan:NONMonotonic:HYSTeresis <value>`

The :ISCan:NONMonotonic:HYSTeresis command sets the hysteresis value used for the Non-monotonic Edge trigger.

<value> is a real number for the hysteresis.

Example The following example sets the hysteresis value used by the Non-monotonic trigger mode to 10 mV.

```
myScope.WriteString ":ISCAN:NONMONOTONIC:HYSTERESIS 1E-2"
```

Query `:ISCan:NONMonotonic:HYSTersis?`

The query returns the hysteresis value used by the Non-monotonic Edge trigger mode.

Returned Format `[:ISCan:NONMonotonic:HYSTeresis]<value><NL>`

Example The following example returns and prints the value of the hysteresis.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:NONMONOTONIC:HYSTERESIS?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:NONMonotonic:SOURce

Command `:ISCan:NONMonotonic:SOURce CHANnel<N>`

The :ISCan:NONMonotonic:SOURce command sets the source used for the Non-monotonic Edge trigger.

<N> is an integer from 1-4.

Example The following example sets the source used by the Non-monotonic trigger mode to channel 1.

```
myScope.WriteString ":ISCAN:NONMONOTONIC:SOURCE CHANNEL1"
```

Query `:ISCan:NONMonotonic:SOURce?`

The query returns the source used by the Non-monotonic Edge trigger mode.

Returned Format `[:ISCan:NONMonotonic:SOURce]CHANnel<N><NL>`

Example The following example returns the currently selected source for the Non-monotonic Edge trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:NONMONOTONIC:SOURCE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

:ISCan:RUNT:HYSTeresis

Command `:ISCan:RUNT:HYSTeresis <value>`

The :ISCan:RUNT:HYSTeresis command sets the hysteresis value used for the Runt trigger.

<value> is a real number for the hysteresis.

Example The following example sets the hysteresis value used by the Runt trigger mode to 10 mV.

```
myScope.WriteString ":ISCAN:RUNT:HYSTERESIS 1E-2"
```

Query `:ISCan:RUNT:HYSTersis?`

The query returns the hysteresis value used by the Runt trigger mode.

Returned Format `[:ISCan:RUNT:HYSTeresis]<value><NL>`

Example The following example returns and prints the value of the hysteresis.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:RUNT:HYSTERESIS?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:RUNT:LLEVel

Command `ISCan:RUNT:LLEVel <lower_level>`

The :ISCan:RUNT:LLEVel (lower level) command sets the lower level limit for the Runt trigger mode.

<lower_level> A real number.

Example The following example sets the lower level limit to 1.0 V.

```
myScope.WriteString ":ISCAN:RUNT:LLEVel 1.0"
```

Query `:ISCan:RUNT:LLEVel?`

The query returns the lower level limit set by the command.

Returned Format `[:ISCan:RUNT:LLEVel] <lower_level><NL>`

Example The following example returns the current lower level used by the Runt trigger and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:RUNT:LLEVel?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:RUNT:SOURce

Command `:ISCan:RUNT:SOURce CHANnel<N>`

The :ISCan:RUNT:SOURce command sets the source used for the Runt trigger.

<N> is an integer from 1-4.

Example The following example sets the source used by the Runt trigger mode to channel 1.

```
myScope.WriteString ":ISCAN:RUNT:SOURCE CHANNEL1"
```

Query `:ISCan:RUNT:SOURce?`

The query returns the source used by the Runt trigger mode.

Returned Format `[:ISCan:RUNT:SOURce]CHANnel<N><NL>`

Example The following example returns the currently selected source for the Runt trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:RUNT:SOURCE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

:ISCan:RUNT:ULEVel

Command `:ISCan:RUNT:ULEVel <upper_level>`

The :ISCan:RUNT:ULEVel (upper level) command sets the upper level limit for the Runt trigger mode.

<upper_level> A real number.

Example The following example sets the upper level value used by the Runt trigger mode to 500 mV.

```
myScope.WriteString ":ISCAN:RUNT:ULEVEL 500E-3"
```

Query `:ISCan:RUNT:ULEVel?`

The query returns the current upper level value used by the Runt trigger.

Returned Format `[:ISCan:RUNT:ULEVel] <upper_level><NL>`

Example The following example returns the current upper level used by the Runt trigger and prints the result to the controller's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":ISCAN:RUNT:ULEVel?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:ISCan:SERial:PATTern

Command `:ISCan:SERial:PATTern <pattern>`

The :ISCan:SERial:PATTern command sets the pattern used for the Serial trigger.

<pattern> is a 1, 0, or X binary character string of up to 80 characters. The pattern can only be expressed in the binary format.

Example The following example sets the pattern used by the Serial trigger to 101100.

```
myScope.WriteString ":ISCAN:SERIAL:PATTERN ""101100"""
```

Query `:ISCan:SERial:PATTern?`

The query returns the pattern used by the Serial trigger mode.

Returned Format `[:ISCan:SERial:PATTern]<pattern><NL>`

Example The following example returns the currently selected pattern for the Serial trigger mode.

```
Dim strPATTERN As String
myScope.WriteString ":ISCAN:SERIAL:PATTERN?"
strPATTERN = myScope.ReadString
Debug.Print strPATTERN
```

:ISCan:SERial:SOURce

Command `:ISCan:SERial:SOURce CHANnel<N>`

The :ISCan:SERial:SOURce command sets the source used for the Serial trigger.

<N> is an integer from 1-4.

Example The following example sets the source used by the Serial trigger mode to channel 1.

```
myScope.WriteString ":ISCAN:SERIAL:SOURCE CHANNEL1"
```

Query `:ISCan:SERial:SOURce?`

The query returns the source used by the Serial trigger mode.

Returned Format `[:ISCan:SERial:SOURce]CHANnel<N><NL>`

Example The following example returns the currently selected source for the Serial trigger mode.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:SERIAL:SOURCE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

:ISCan:ZONE:HIDE

Command `:ISCan:ZONE:HIDE {{ON | 1} | {OFF | 0}}`

The :ISCan:ZONE:HIDE command lets you hide or show all InfiniiScan zones on the display.

Example The following example hides all InfiniiScan zones on the display.

```
myScope.WriteString ":ISCAN:ZONE:HIDE ON"
```

Query `:ISCan:ZONE:HIDE?`

The query returns the current zone hide setting.

Returned Format `[:ISCan:ZONE:HIDE]{1 | 0};`

Example The following example returns the current zone hide setting.

```
Dim strHide As String
myScope.WriteString ":ISCAN:ZONE:HIDE?"
strHide = myScope.ReadString
Debug.Print strHide
```

:ISCan:ZONE:SOURce

Command `:ISCan:ZONE:SOURce CHANnel<N>`

The :ISCan:ZONE:SOURce command sets the source used for the zone qualify trigger.

<N> is an integer from 1-4.

Example The following example sets the source used by the zone qualify trigger to channel 1.

```
myScope.WriteString ":ISCAN:ZONE:SOURCE CHANNEL1"
```

Query `:ISCan:ZONE:SOURce?`

The query returns the source used by the zone qualify trigger.

Returned Format `[:ISCan:ZONE:SOURce]CHANnel<N><NL>`

Example The following example returns the currently selected source for zone qualify trigger.

```
Dim strSOURCE As String
myScope.WriteString ":ISCAN:ZONE:SOURCE?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

:ISCan:ZONE<N>:MODE

Command `:ISCan:ZONE<N>:MODE {INTersect | NOTintersect}`

The :ISCan:ZONE<N>:MODE command sets the Zone Qualify trigger mode. For the INTersect mode, the waveform must enter the zone region to qualify as a valid waveform. For NOTintersect mode, the waveform cannot enter a zone region to qualify as a valid waveform.

<N> is an integer from 1-4.

Example The following example sets the mode to intersect for zone 1.

```
myScope.WriteString ":ISCAN:ZONE1:MODE INTERSECT"
```

Query `:ISCan:ZONE<N>:MODE?`

The query returns the mode used by zone 1.

Returned Format `[:ISCan:ZONE<N>:MODE]{INTersect | NOTintersect}<NL>`

Example The following example returns the currently selected mode for zone 1.

```
Dim strMODE As String
myScope.WriteString ":ISCAN:ZONE1:MODE?"
strMODE = myScope.ReadString
Debug.Print strMODE
```

:ISCan:ZONE<N>:PLACement

Command `:ISCan:ZONE<N>:PLACement <width>,<height>,<x_center>,<y_center>`

The :ISCan:ZONE<N>:PLACement command sets the location and size of a zone for the zone qualify trigger mode.

<N> is an integer from 1-4.

<width> a real number defining the width of a zone in seconds.

<height> is a real number defining the height of a zone in volts.

<x_center> is a real number defining the x coordinate of the center of the zone in seconds.

<y_center> is a real number defining the y coordinate of the center of the zone in volts.

Example The following example sets the size of zone 1 to be 500 ps wide and 0.5 volts high and centered about the xy coordinate of 1.5 ns and 1 volt.

```
myScope.WriteString ":ISCAN:ZONE1:PLACEMENT 500e-12,0.5,1.5e-9,1"
```

Query `:ISCan:ZONE<N>:PLACement?`

The query returns the placement values used by zone 1.

Returned Format `[:ISCan:ZONE<N>:PLACement]<width>,<height>,<x_center>,<y_center><NL>`

Example The following example returns the current placement values for zone 1.

```
Dim strPLACEMENT As String
myScope.WriteString ":ISCAN:ZONE1:PLACEMENT?"
strPLACEMENT = myScope.ReadString
Debug.Print strPLACEMENT
```

:ISCan:ZONE<N>:STATe

Command `:ISCan:ZONE<N>:STATe {{ON | 1} | {OFF | 0}}`

The :ISCan:ZONE<N>:STATe command turns a zone off or on for the zone qualify trigger.

<N> is an integer from 1-4.

Example The following example turns on zone 2.

```
myScope.WriteString ":ISCAN:ZONE2:STATE ON"
```

Query `:ISCan:ZONE<N>:STATe?`

The query returns the state value for a zone.

Returned Format `[:ISCan:ZONE<N>:STATe]{1 | 0}<NL>`

Example The following example returns the current state value for zone 2.

```
Dim strSTATE As String
myScope.WriteString ":ISCAN:ZONE2:STATE?"
strSTATE = myScope.ReadString
Debug.Print strSTATE
```

21 Limit Test Commands

:LTESt:FAIL 412
:LTESt:LLIMit 413
:LTESt:MEASurement 414
:LTESt:RESults? 415
:LTESt:TEST 416
:LTESt:ULIMit 417

The Limit Test commands and queries control the limit test features of the oscilloscope. Limit testing automatically compares measurement results with pass or fail limits. The limit test tracks up to four measurements. The action taken when the test fails is also controlled with commands in this subsystem.

:LTESt:FAIL

Command `:LTESt:FAIL {INSide | OUTSide}`

The :LTESt:FAIL command sets the fail condition for an individual measurement. The conditions for a test failure are set on the source selected with the last LTESt:MEASurement command.

When a measurement failure is detected by the limit test, the fail action conditions are executed, and there is the potential to generate an SRQ.

INSide FAIL INside causes the oscilloscope to fail a test when the measurement results are within the parameters set by the LLTESt:LIMit and LTESt:ULIMit commands.

OUTSide FAIL OUTside causes the oscilloscope to fail a test when the measurement results exceed the parameters set by LTESt:LLIMit and LTESt:ULIMit commands.

Example The following example causes the oscilloscope to fail a test when the measurements are outside the lower and upper limits.

```
myScope.WriteString ":LTEST:FAIL OUTSIDE"
```

Query `:LTESt:FAIL?`

The query returns the current set fail condition.

Returned Format `[:LTESt:FAIL] {INSide | OUTSide}<NL>`

Example The following example returns the current fail condition and prints the result to the controller's screen.

```
Dim strFAIL As String
myScope.WriteString ":LTEST:FAIL?"
strFAIL = myScope.ReadString
Debug.Print strFAIL
```

:LTESt:LLIMit

(Lower Limit)

Command `:LTESt:LLIMit <lower_value>`

The :LTESt:LLIMit (Lower LIMit) command sets the lower test limit for the active measurement currently selected by the :LTESt:MEASurement command.

<lower_value> A real number.

Example The following example sets the lower test limit to 1.0.

```
myScope.WriteString ":LTEST:LLIMIT 1.0"
```

If, for example, you chose to measure volts peak-peak and want the smallest acceptable signal swing to be one volt, you could use the above command, then set the limit test to fail when the signal is outside the specified limit.

Query `:LTESt:LLIMit?`

The query returns the current value set by the command.

Returned Format `[:LTESt:LLIMit]<lower_value><NL>`

Example The following example returns the current lower test limit and prints the result to the controller's screen.

```
Dim strLLIM As String
myScope.WriteString ":LTEST:LLIMIT?"
strLLIM = myScope.ReadString
Debug.Print strLLIM
```

:LTESt:MEASurement

Command `:LTESt:MEASurement {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}`

The :LTESt:MEASurement command selects the current source for Limit Test for the ULIMit and LLIMit commands. It selects one of the active measurements as referred to by their position in the measurement window on the bottom of the screen. Measurements are numbered from left to right.

Example The following example selects the first measurement as the source for the limit testing commands.

```
myScope.WriteString ":LTEST:MEASUREMENT MEAS1"
```

Query `:LTESt:MEASurement?`

The query returns the currently selected measurement source.

Returned Format `[:LTESt:MEASurement]{MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5} <NL>`

Example The following example returns the currently selected measurement source for the limit testing commands.

```
Dim strSOURCE As String
myScope.WriteString ":LTEST:MEASUREMENT?"
strSOURCE = myScope.ReadString
Debug.Print strSOURCE
```

See Also Measurements are started in the Measurement Subsystem.

:LTESt:RESults?

Query `:LTESt:RESults? {MEAS1 | MEAS2 | MEAS3 | MEAS4 | MEAS5}`

The query returns the measurement results for selected measurement. The values returned are the failed minimum value (Fail Min), the failed maximum value (Fail Max), and the total number of measurements made (# of Meas).

Returned Format `[:LTESt:RESults] <fail_min>,<fail_max>,<num_meas><NL>`

<fail_min> A real number representing the total number of measurements that have failed the minimum limit.

<fail_max> A real number representing the total number of measurements that have failed the maximum limit.

<num_meas> A real number representing the total number of measurements that have been made.

Example The following example returns the values for the limit test of measurement 1.

```
Dim strRESULTS As String
myScope.WriteString ":LTEST:RESults? MEAS1"
strRESULTS = myScope.ReadString
Debug.Print strRESULTS
```

See Also Measurements are started in the Measurement Subsystem.

:LTESt:TEST

Command `:LTESt:TEST {{ON | 1} {OFF | 0}}`

The LTESt:TEST command controls the execution of the limit test function. ON allows the limit test to run over all of the active measurements. When the limit test is turned on, the limit test results are displayed on screen in a window below the graticule.

Example The following example turns off the limit test function.

```
myScope.WriteString ":LTEST:TEST OFF"
```

Query `:LTESt:TEST?`

The query returns the state of the TEST control.

Returned Format `[:LTESt:TEST] {1 | 0} <NL>`

Example The following example returns the current state of the limit test and prints the result to the controller's screen.

```
Dim strTEST As String
myScope.WriteString ":LTEST:TEST?"
strTEST = myScope.ReadString
Debug.Print strTEST
```

The result of the MEAS:RESults? query has two extra fields when LimitTESt:TEST is ON (failures, total). Failures is a number and total is the total number of measurements made.

:LTESt:ULIMit

(Upper Limit)

Command `:LTESt:ULIMit <upper_value>`

The :LTESt:ULIMit (Upper LIMit) command sets the upper test limit for the active measurement currently selected by the last :LTESt:MEASurement command.

<upper_value> A real number.

Example The following example sets the upper limit of the currently selected measurement to 500 milli.

```
myScope.WriteString ":LTEST:ULIMIT 500E-3"
```

Suppose you are measuring the maximum voltage of a signal with Vmax, and that voltage should not exceed 500 mV. You can use the above program and set the LTESt:FAIL OUTside command to specify that the limit subsystem will fail a measurement when the voltage exceeds 500 mV.

Query `:LTESt:ULIMit?`

The query returns the current upper limit of the limit test.

Returned Format `[:LTESt:ULIMit] <upper_value><NL>`

Example The following example returns the current upper limit of the limit test and prints the result to the controller's screen.

```
Dim strULIM As String
myScope.WriteString ":LTEST:ULIMIT?"
strULIM = myScope.ReadString
Debug.Print strULIM
```

22 Lister Commands

:LISTer:DATA 420
:LISTer:DISPlay 421

The LISTer subsystem is used to turn on/off the serial decode Lister display and return data from the Lister display.

:LISTer:DATA

Query `:LISTer:DATA? {SBUS1 | SBUS2 | SBUS3 | SBUS4}`

The :LISTer:DATA? query returns the lister data.

Returned Format

```
<binary block><NL>

<binary_block> ::= comma-separated data with newlines at the
                   end of each row
```

See Also

- ":LISTer:DISPlay" on page 421

:LISTer:DISPlay

Command `:LISTer:DISPlay <value>`

`<value> ::= {OFF | ON | SBUS1 | SBUS2 | SBUS3 | SBUS4}`

The :LISTer:DISPlay command configures which of the serial buses to display in the Lister, or whether the Lister is off. "ON" or "1" is the same as "SBUS1".

When set to "ALL", the decode information for different buses is interleaved in time.

Serial bus decode must be on before it can be displayed in the Lister.

Query `:LISTer:DISPlay?`

The :LISTer:DISPlay? query returns the Lister display setting.

Returned Format `<value><NL>`

`<value> ::= {OFF | ON | SBUS1 | SBUS2 | SBUS3 | SBUS4}`

See Also

- ":SBUS<N>[:DISPlay]" on page 785
- ":LISTer:DATA" on page 420

23 Marker Commands

:MARKer:CURSor? 424
:MARKer:MEASurement:MEASurement 425
:MARKer:MODE 426
:MARKer:X1Position 427
:MARKer:X2Position 428
:MARKer:X1Y1source 429
:MARKer:X2Y2source 430
:MARKer:XDELta? 431
:MARKer:Y1Position 432
:MARKer:Y2Position 433
:MARKer:YDELta? 434

The commands in the MARKer subsystem specify and query the settings of the time markers (X axis) and current measurement unit markers (volts, amps, and watts for the Y axis). You typically set the Y-axis measurement units using the :CHANnel:UNITs command.

NOTE

Guidelines for Using Queries in Marker Modes

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries. If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected.

:MARKer:CURSor?

Query `:MARKer:CURSor? {DELTa | STARt | STOP}`

The :MARKer:CURSor? query returns the time and current measurement unit values of the specified marker (if markers are in Track Waveforms mode) as an ordered pair of time and measurement unit values.

- If DELTA is specified, the value of delta Y and delta X are returned.
- If START is specified, marker A's x-to-y positions are returned.
- If STOP is specified, marker B's x-to-y positions are returned.

Returned Format

```
[:MARKer:CURSor] {DELTa | STARt | STOP}
{<Ax, Ay> | <Bx, By> | <deltaX, deltaY>}<NL>
```

Example This example returns the current position of the X cursor and measurement unit marker 1 to the string variable, strPosition. The program then prints the contents of the variable to the computer's screen.

```
Dim strPosition As String   ' Dimension variable.
myScope.WriteString ":MARKER:CURSOR? START"
strPosition = myScope.ReadString
Debug.Print strPosition
```

CAUTION

The :MARKer:CURSor? query may change marker mode and results.

In Track Waveforms mode, use :MARKer:CURSor? to track the position of the waveform. In Manual Markers and Track Measurements Markers modes, use other marker queries, such as the X1Position? and X2Position?, and VSTArt? and VSTOp? queries.

If you use :MARKer:CURSor? when the oscilloscope is in either Manual Markers or Track Measurements Markers modes, it will put the oscilloscope in Track Waveforms mode, regardless of the mode previously selected. In addition, measurement results may not be what you expected. In addition, measurement results may not be what you expected.

:MARKer:MEASurement:MEASurement

Command `:MARKer:MEASurement:MEASurement {MEASurement<N>}`

The :MARKer:MEASurement:MEASurement command specifies which measurement markers track. This setting is only used when the :MARKer:MODE is set to MEASurement.

<N> MEASurement<N> is an integer, 1-10.

Example This example sets the markers to track the fourth measurement.

```
myScope.WriteString ":MARKER:MEASurement:MEASurement MEASurement4"
```

Query `:MARKer:MEASurement:MEASurement?`

The :MARKer:MEASurement:MEASurement? query returns the currently specified measurement for marker tracking.

Returned Format `[:MARKer:MEASurement:MEASurement] {MEAS<N>}<NL>`

Example This example places the current marker mode in the string variable, strTrackMeas, then prints the contents of the variable to the computer's screen.

```
Dim strTrackMeas As String   ' Dimension variable.
myScope.WriteString ":MARKER:MEASurement:MEASurement?"
strTrackMeas = myScope.ReadString
Debug.Print strTrackMeas
```

See Also

- ":MARKer:MODE" on page 426

:MARKer:MODE

Command `:MARKer:MODE {OFF | MANual | WAVeform | MEASurement | FFTPeak}`

The :MARKer:MODE command sets the marker mode:

- OFF – Removes the marker information from the display.
- MANual – Enables manual placement of markers A and B.
- WAVeform – Tracks the current waveform.
- MEASurement – Tracks the most recent measurement.
- FFTPeak – Tracks the current FFT peak that has been navigated to.

Example This example sets the marker mode to waveform.

```
myScope.WriteString ":MARKer:MODE WAVeform"
```

Query `:MARKer:MODE?`

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

Returned Format `[:MARKer:MODE] {OFF | MAN | WAV | MEAS | FFTP}<NL>`

Example This example places the current marker mode in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:MODE?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:MARKer:X1Position

Command `:MARKer:X1Position <Ax_position>`

The :MARKer:X1Position command sets the Ax marker position, and moves the Ax marker to the specified time with respect to the trigger time.

<Ax_position> A real number for the time at the Ax marker in seconds.

Example This example sets the Ax marker to 90 ns.

```
myScope.WriteString ":MARKER:X1POSITION 90E-9"
```

Query `:MARKer:X1Position?`

The :MARKer:X1Position? query returns the time at the Ax marker position.

Returned Format `[:MARKer:X1Position] <Ax_position><NL>`

Example This example returns the current setting of the Ax marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:X1POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also :MARKer:TSTArt

:MARKer:X2Position

Command `:MARKer:X2Position <Bx_position>`

The :MARKer:X2Position command sets the Bx marker position and moves the Bx marker to the specified time with respect to the trigger time.

<Bx_position> A real number for the time at the Bx marker in seconds.

Example This example sets the Bx marker to 90 ns.

```
myScope.WriteString ":MARKER:X2POSITION 90E-9"
```

Query `:MARKer:X2Position?`

The :MARKer:X2Position? query returns the time at Bx marker in seconds.

Returned Format `[:MARKer:X2Position] <Bx_position><NL>`

Example This example returns the current position of the Bx marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:X2POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MARKer:X1Y1source

Command `:MARKer:X1Y1source {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized | DIGital<M> | BUS<N>}`

The :MARKer:X1Y1source command sets the source for the Ax and Ay markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

BUS<N> is an integer, 1-4. Buses are available on mixed-signal oscilloscopes.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 1 as the source for markers Ax and Ay.

```
myScope.WriteString ":MARKer:X1Y1source CHANnel1"
```

Query `:MARKer:X1Y1source?`

The :MARKer:X1Y1source? query returns the current source for markers Ax and Ay.

Returned Format `[:MARKer:X1Y1source] {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized | DIGital<M> | BUS<N>}<NL>`

Example This example returns the current source selection for the Ax and Ay markers to the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:X1Y1source?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:MARKer:X2Y2source

Command `:MARKer:X2Y2source {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized | DIGital<M> | BUS<N>}`

The :MARKer:X2Y2source command sets the source for the Bx and By markers. The channel you specify must be enabled for markers to be displayed. If the channel, function, or waveform memory that you specify is not on, an error message is issued and the query will return channel 1.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

BUS<N> is an integer, 1-4. Buses are available on mixed-signal oscilloscopes.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 1 as the source for markers Bx and By.

```
myScope.WriteString ":MARKer:X2Y2source CHANnel1"
```

Query `:MARKer:X2Y2source?`

The :MARKer:X2Y2source? query returns the current source for markers Bx and By.

Returned Format `[:MARKer:X2Y2source] {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized | DIGital<M> | BUS<N>}<NL>`

Example This example returns the current source selection for the Bx and By markers to the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":MARKer:X2Y2source?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:MARKer:XDELta?

Query `:MARKer:XDELta?`

The :MARKer:XDELta? query returns the time difference between Ax and Bx time markers.

Xdelta = time at Bx - time at Ax

Returned Format `[:MARKer:XDELta] <time><NL>`

<time> Time difference between Ax and Bx time markers in seconds.

Example This example returns the current time between the Ax and Bx time markers to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:XDELTA?"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MARKer:Y1Position

Command `:MARKer:Y1Position <Ay_position>`

The :MARKer:Y1Position command sets the Ay marker position on the specified source.

<Ay_position> A real number for the current measurement unit value at Ay (volts, amps, or watts).

Example This example sets the Ay marker to 10 mV.

```
myScope.WriteString ":MARKER:Y1POSITION 10E-3"
```

Query `:MARKer:Y1Position?`

The :MARKer:Y1Position? query returns the current measurement unit level at the Ay marker position.

Returned Format `[:MARKer:Y1Position] <Ay_position><NL>`

Example This example returns the current setting of the Ay marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:Y1POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MARKer:Y2Position

Command `:MARKer:Y2Position <By_position>`

The :MARKer:Y2Position command sets the By marker position on the specified source.

<By_position> A real number for the current measurement unit value at By (volts, amps, or watts).

Example This example sets the By marker to-100 mV.

```
myScope.WriteString ":MARKER:Y2POSITION -100E-3"
```

Query `:MARKer:Y2Position?`

The :MARKer:Y2Position? query returns the current measurement unit level at the By marker position.

Returned Format `[:MARKer:Y2Position] <By_position><NL>`

Example This example returns the current setting of the By marker to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:Y2POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MARKer:YDELta?

Query `:MARKer:YDELta?`

The :MARKer:YDELta? query returns the current measurement unit difference between Ay and By.

Ydelta = value at By - value at Ay

Returned Format `[:MARKer:YDELta] <value><NL>`

<value> Measurement unit difference between Ay and By.

Example This example returns the voltage difference between Ay and By to the numeric variable, varVolts, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MARKER:YDELTA?"
varVolts = myScope.ReadNumber
Debug.Print FormatNumber(varVolts, 0)
```

24
Mask Test Commands

:MTESt:ALIGn 437
:MTESt:AlignFIT 438
:MTESt:AMASk:CREate 440
:MTESt:AMASk:SOURce 441
:MTESt:AMASk:SAVE 442
:MTESt:AMASk:UNITs 443
:MTESt:AMASk:XDELta 444
:MTESt:AMASk:YDELta 445
:MTESt:AUTO 446
:MTESt:AVERage 447
:MTESt:AVERage:COUNt 448
:MTESt:COUNt:FAILures? 449
:MTESt:COUNt:FUI? 450
:MTESt:COUNt:FWAVeforms? 451
:MTESt:COUNt:UI? 452
:MTESt:COUNt:WAVeforms? 453
:MTESt:DELete 454
:MTESt:ENABle 455
:MTESt:FOLDing 456
:MTESt:FOLDing:BITS 457
:MTESt:FOLDing:FAST 458
:MTESt:HAMPlitude 459
:MTESt:IMPedance 460
:MTESt:INVert 461
:MTESt:LAMPlitude 462
:MTESt:LOAD 463
:MTESt:NREGions? 464
:MTESt:PROBe:IMPedance? 465
:MTESt:RUMode 466
:MTESt:RUMode:SOFailure 467
:MTESt:SCALe:BIND 468
:MTESt:SCALe:X1 469
:MTESt:SCALe:XDELta 470
:MTESt:SCALe:Y1 471

:MTESt:SCALe:Y2 472
:MTESt:SOURce 473
:MTESt:STARt 474
:MTESt:STOP 475
:MTESt:STIMe 476
:MTESt:TITLe? 477
:MTESt:TRIGger:SOURce 478

The MTESt subsystem commands and queries control the mask test features. Mask Testing automatically compares measurement results with the boundaries of a set of polygons that you define. Any waveform or sample that falls within the boundaries of one or more polygons is recorded as a failure.

The FOLDing command is only available when the E2688A Clock Recovery Software is installed.

:MTESt:ALIGn

Command `:MTESt:ALIGn`

The :MTESt:ALIGn command automatically aligns and scales the mask to the current waveform on the display. The type of mask alignment performed depends on the current setting of the Use File Setup When Aligning control. See the :MTESt:AUTO command for more information.

Example This example aligns the current mask to the current waveform.

```
myScope.WriteString ":MTEST:ALIGN"
```

:MTESt:AlignFIT

Command

```
:MTESt:AlignFIT {EYEAMI | EYECMI | EYENRZ | FANWidth | FAPeriod
                | FAPWidth | FYNWidth | FYPWidth | NONE | NWIDth
                | PWIDth | TMAX | TMIN}
```

The :MTESt:AlignFIT command specifies the alignment type for aligning a mask to a waveform. The pulse mask standard has rules that determine which controls the oscilloscope can adjust or change during the alignment process. In the following table of alignment types, an X in a column indicates that the control can be adjusted.

Table 23 Available Alignment Types

Alignment Type	Waveform Type	Horizontal Position	0 Level Voltage	1 Level Voltage	Vertical Offset	Invert Waveform
EYEAMI	AMI	X	X	X		
EYECMI	CMI	X	X	X		
EYENRZ	NRZ	X	X	X		
FANWidth	Negative	X			X	X
FAPeriod	Full Period	X	X			
FAPWidth	Positive	X			X	X
FYNWidth	Negative	X	X			X
FYPWidth	Positive	X	X			X
NONE	Automask					
NWIDth	Negative Pulse	X	X	X		X
PWIDth	Positive Pulse	X	X	X		X
TMAX	Positive Sine Pulse	X	X	X		X
TMIN	Negative Sine Pulse	X	X	X		X

Example This example specifies the alignment type to be EYEAMI.

```
myScope.WriteString ":MTEST:ALIGNFIT EYEAMI"
```

Query

```
:MTESt:AlignFIT?
```

The :MTEST:AlignFIT? query returns the alignment type used for the mask.

Returned Format `[:MTESt:AlignFIT] {EYEAMI | EYECMI | EYENRZ | FANWidth | FAPeriod | FAPWidth | FYNWidth | FYPWidth | NONE | NWIDth | PWIDth | TMAX | TMIN}<NL>`

:MTESt:AMASk:CREate

Command `:MTESt:AMASk:CREate`

The :MTESt:AMASk:CREate command automatically constructs a mask around the current selected channel, using the tolerance parameters defined by the AMASk:XDELta, AMASk:YDELta, and AMASk:UNITs commands. The mask only encompasses the portion of the waveform visible on the display, so you must ensure that the waveform is acquired and displayed consistently to obtain repeatable results.

The :MTESt:SOURce command selects the channel and should be set before using this command.

Example This example creates an automask using the current XDELta and YDELta units settings.

```
myScope.WriteString ":MTEST:AMASK:CREATE"
```

:MTESt:AMASk:SOURce

Command `:MTESt:AMASk:SOURce CHANnel<N>`

The :MTESt:AMASk:SOURce command selects the source for the interpretation of the AMASk:XDELta and AMASk:YDELta parameters when AMASk:UNITs is set to CURRent. When UNITs are CURRent, the XDELta and YDELta parameters are defined in terms of the channel units, as set by the :CHANnel:UNITs command, of the selected source. Suppose that UNITs are CURRent and that you set SOURce to CHANNEL1, which is using units of volts. Then you can define AMASk:XDELta in terms of volts and AMASk:YDELta in terms of seconds.

<N> An integer, 1-4.

Example This example sets the automask source to Channel 1.

```
myScope.WriteString ":MTEST:AMASK:SOURCE CHANNEL1"
```

Query `:MTESt:AMASk:SOURce?`

The :MTESt:AMASk:SOURce? query returns the currently set source.

Returned Format `[:MTESt:AMASk:SOURce] CHANnel<N><NL>`

Example This example gets the source setting for automask and prints the result on the computer display.

```
Dim strAmask_source As String
myScope.WriteString ":MTEST:AMASK:SOURCE?"
strAmask_source = myScope.ReadString
Debug.Print strAmask_source
```

:MTESt:AMASk:SAVE

Command `:MTESt:AMASk:{SAVE | STORe} "<filename>"`

NOTE The :MTESt:AMASk:STORe command is equivalent to the :MTESt:AMASk:SAVE command.

The :MTESt:AMASk:SAVE command saves the automask generated mask to a file. If an automask has not been generated, an error occurs.

<filename> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The filename assumes the present working directory if a path does not precede the file name. The default save path is C:\Documents and Settings\All Users\Documents\Infiniium\masks. The filename must have a .msk or .MSK extension or the command will fail.

Example This example saves the automask generated mask to a file named "FILE1.MSK".

```
myScope.WriteString ":MTEST:AMASK:SAVE""FILE1.MSK"""
```

:MTESt:AMASk:UNITs

Command `:MTESt:AMASk:UNITs {CURRent | DIVisions}`

The :MTESt:AMASk:UNITs command alters the way the mask test subsystem interprets the tolerance parameters for automasking as defined by AMASk:XDELta and AMASk:YDELta commands.

CURRent When set to CURRent, the mask test subsystem uses the units as set by the :CHANnel:UNITs command, usually time for ΔX and voltage for ΔY.

DIVisions When set to DIVisions, the mask test subsystem uses the graticule as the measurement system, so tolerance settings are specified as parts of a screen division. The mask test subsystem maintains separate XDELta and YDELta settings for CURRent and DIVisions. Thus, XDELta and YDELta are not converted to new values when the UNITs setting is changed.

Example This example sets the measurement units for automasking to the current :CHANnel:UNITs setting.

```
myScope.WriteString ":MTEST:AMASK:UNITS CURRENT"
```

Query `:MTESt:AMASk:UNITs?`

The AMASk:UNITs query returns the current measurement units setting for the mask test automask feature.

Returned Format `[:MTESt:AMASk:UNITs] {CURRent | DIVision}<NL>`

Example This example gets the automask units setting, then prints the setting on the screen of the computer.

```
Dim strAutomask_units As String
myScope.WriteString ":MTEST:AMASK:UNITS?"
strAutomask_units = myScope.ReadString
Debug.Print strAutomask_units
```

:MTESt:AMASk:XDELta

Command `:MTESt:AMASk:XDELta <xdelta_value>`

The :MTESt:AMASk:XDELta command sets the tolerance in the X direction around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to horizontal values of the waveform to determine the boundaries of the mask.

<xdelta_value> A value for the horizontal tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies time in the horizontal direction, the tolerance will be ±250 ms. If the setting for AMASk:UNITs is DIVisions, the same xdelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example This example sets the units to divisions and sets the ΔX tolerance to one-eighth of a division.

```
myScope.WriteString ":MTEST:AMASK:UNITS DIVISIONS"
myScope.WriteString ":MTEST:AMASK:XDELTA 125E-3"
```

Query `:MTESt:AMASk:XDELta?`

The AMASk:XDELta? query returns the current setting of the ΔX tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format `[:MTESt:AMASk:XDELta] <xdelta_value><NL>`

Example This example gets the measurement system units and ΔX settings for automasking from the oscilloscope and prints the results on the computer screen.

```
Dim strAutomask_units As String
Dim strAutomask_xdelta As String
myScope.WriteString ":MTEST:AMASK:UNITS?"
strAutomask_units = myScope.ReadString
myScope.WriteString ":MTEST:AMASK:XDELTA?"
strAutomask_xdelta = myScope.ReadString
Debug.Print strAutomask_units
Debug.Print strAutomask_xdelta
```

:MTESt:AMASk:YDELta

Command `:MTESt:AMASk:YDELta <ydelta_value>`

The :MTESt:AMASk:YDELta command sets the vertical tolerance around the waveform for the automasking feature. The absolute value of the tolerance will be added and subtracted to vertical values of the waveform to determine the boundaries of the mask.

This command requires that mask testing be enabled, otherwise a settings conflict error message is displayed. See :MTESt:ENABle for information on enabling mask testing.

<ydelta_value> A value for the vertical tolerance. This value is interpreted based on the setting specified by the AMASk:UNITs command; thus, if you specify 250-E3, the setting for AMASk:UNITs is CURRent, and the current setting specifies voltage in the vertical direction, the tolerance will be ±250 mV. If the setting for AMASk:UNITs is DIVisions, the same ydelta_value will set the tolerance to ±250 millidivisions, or 1/4 of a division.

Example This example sets the units to current and sets the ΔY tolerance to 30 mV, assuming that the current setting specifies volts in the vertical direction.

```
myScope.WriteString ":MTEST:AMASK:UNITS CURRENT"
myScope.WriteString ":MTEST:AMASK:YDELTA 30E-3"
```

Query `:MTESt:AMASk:YDELta?`

The AMASk:YDELta? query returns the current setting of the ΔY tolerance for automasking. If your computer program will interpret this value, it should also request the current measurement system using the AMASk:UNITs query.

Returned Format `[:MTESt:AMASk:YDELta] <ydelta_value><NL>`

Example This example gets the measurement system units and ΔY settings for automasking from the oscilloscope and prints the results on the computer screen.

```
Dim strAutomask_units As String
Dim strAutomask_ydelta As String
myScope.WriteString ":MTEST:AMASK:UNITS?"
strAutomask_units = myScope.ReadString
myScope.WriteString ":MTEST:AMASK:YDELTA?"
strAutomask_ydelta = myScope.ReadString
Debug.Print strAutomask_units
Debug.Print strAutomask_ydelta
```

:MTESt:AUTO

Command `:MTESt:AUTO {{ON | 1} | {OFF | 0}}`

The :MTESt:AUTO command enables (ON) or disables (OFF) the Use File Setup When Aligning control. This determines which type of mask alignment is performed when the :MTESt:ALIGn command is sent. When enabled, the oscilloscope controls are changed to the values which are determined by the loaded mask file. This alignment guarantees that the aligned mask and any subsequent mask tests meet the requirements of the standard.

When disabled, the alignment is performed using the current oscilloscope settings. This may be useful when troubleshooting problems during the design phase of a project.

Example This example enables the Use File Settings When Aligning control.

```
myScope.WriteString ":MTEST:AUTO ON"
```

Query `:MTESt:AUTO?`

The :MTESt:AUTO? query returns the current value of the Use File Setup When Aligning control.

Returned Format `[:MTESt:AUTO] {1 | 0} <NL>`

Example

```
myScope.WriteString ":MTEST:AUTO?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MTESt:AVERage

Command `:MTESt:AVERage {{ON | 1} | {OFF | 0}}`

The :MTESt:AVERage command enables or disables averaging. When ON, the oscilloscope acquires multiple data values for each time bucket, and averages them. When OFF, averaging is disabled. To set the number of averages, use the :MTESt:AVERage:COUNt command described next.

The :ACQuire:AVERage command performs the same function as this command.

Averaging is not available in PDETect mode.

Example This example turns averaging on.

```
myScope.WriteString ":MTEST:AVERAGE ON"
```

Query `:MTESt:AVERage?`

The :MTESt:AVERage? query returns the current setting for averaging.

Returned Format `[:MTESt:AVERage] {1 | 0} <NL>`

Example This example places the current settings for averaging into the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":MTEST:AVERAGE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MTESt:AVERage:COUNt

Command `:MTESt:AVERage:COUNt <count_value>`

The :MTESt:AVERage:COUNt command sets the number of averages for the waveforms. In the AVERage mode, the :MTESt:AVERage:COUNt command specifies the number of data values to be averaged for each time bucket before the acquisition is considered complete for that time bucket.

The :ACQuire:AVERage:COUNt command performs the same function as this command.

<count_value> An integer, 2 to 65534, specifying the number of data values to be averaged.

Example This example specifies that 16 data values must be averaged for each time bucket to be considered complete. The number of time buckets that must be complete for the acquisition to be considered complete is specified by the :MTESt:COMPlete command.

```
myScope.WriteString ":MTESt:AVERage:COUNT 16"
```

Query `:MTESt:AVERage:COUNt?`

The :MTESt:AVERage:COUNt? query returns the currently selected count value.

Returned Format `[:MTESt:AVERage:COUNt] <value><NL>`

<value> An integer, 2 to 65534, specifying the number of data values to be averaged.

Example This example checks the currently selected count value and places that value in the string variable, varResult. The program then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":MTEST:AVERAGE:COUNT?"
varResult = myScope.ReadNumber
Debug.Print FormatNumber(varResult, 0)
```

:MTESt:COUNt:FAILures?

Query `:MTESt:COUNt:FAILures? REGion<number>`

The MTESt:COUNt:FAILures? query returns the number of failures that occurred within a particular mask region.

The value 9.999E37 is returned if mask testing is not enabled or if you specify a region number that is unused.

<number> An integer, 1 through 8, designating the region for which you want to determine the failure count.

Returned Format `[:MTESt:COUNt:FAILures] REGion<number><number_of_failures> <NL>`

<number_of_failures> The number of failures that have occurred for the designated region.

Example This example determines the current failure count for region 3 and prints it on the computer screen.

```
Dim strMask_failures As String
myScope.WriteString ":MTEST:COUNT:FAILURES? REGION3"
strMask_failures = myScope.ReadString
Debug.Print strMask_failures
```

:MTESt:COUNt:FUI?

Query `:MTESt:COUNt:FUI?`

The MTESt:COUNt:FUI? query returns the number of unit interval failures that have occurred.

Returned Format `[:MTESt:COUNt:FUI?] <unit_interval_failures> <NL>`

<unit_interval_failures> The number of unit interval failures.

Example This example determines the current number of unit interval failures and prints it to the computer screen.

```
Dim strFailures As String
myScope.WriteString ":MTEST:COUNT:FUI?"
strFailures = myScope.ReadString
Debug.Print strFailures
```

:MTESt:COUNt:FWAVeforms?

Query `:MTESt:COUNt:FWAVeforms?`

The :MTESt:COUNt:FWAVeforms? query returns the total number of failed waveforms in the current mask test run. This count is for all regions and all waveforms, so if you wish to determine failures by region number, use the COUNt:FAILures? query.

This count may not always be available. It is available only when the following conditions are true:

- Mask testing was turned on before the histogram or color grade persistence, and
- No mask changes have occurred, including scaling changes, editing, or new masks.

The value 9.999E37 is returned if mask testing is not enabled, or if you have modified the mask.

Returned Format `[:MTESt:COUNt:FWAVeforms] <number_of_failed_waveforms><NL>`

<number_of_failed_waveforms> The total number of failed waveforms for the current test run.

Example This example determines the number of failed waveforms and prints the result on the computer screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTEST:COUNT:FWAVEFORMS?"
strMask_fwaveforms = myScope.ReadString
Debug.Print strMask_fwaveforms
```

:MTESt:COUNt:UI?

Query `:MTESt:COUNt:UI?`

The MTESt:COUNt:UI? query returns the number of unit intervals that have been mask tested.

Returned Format `[:MTESt:COUNt:UI?] <unit_intervals_tested> <NL>`

<unit_intervals_tested> The number of unit intervals tested.

Example This example determines the current number of unit intervals tested and prints it to the computer screen.

```
Dim strUnit_intervals As String
myScope.WriteString ":MTEST:COUNT:uUI?"
strUnit_intervals = myScope.ReadString
Debug.Print strUnit_intervals
```

:MTESt:COUNt:WAVeforms?

Query `:MTESt:COUNt:WAVeforms?`

The :MTESt:COUNt:WAVeforms? query returns the total number of waveforms acquired in the current mask test run. The value 9.999E37 is returned if mask testing is not enabled.

Returned Format `[:MTESt:COUNt:WAVeforms] <number_of_waveforms><NL>`

<number_of_waveforms> The total number of waveforms for the current test run.

Example This example determines the number of waveforms acquired in the current test run and prints the result on the computer screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MTEST:COUNT:WAVEFORMS?"
varMask_waveforms = myScope.ReadNumber
Debug.Print FormatNumber(varMask_waveforms, 0)
```

:MTESt:DELete

Command `:MTESt:DELete`

The :MTESt:DELete command clears the currently loaded mask.

Example This example clears the currently loaded mask.

```
myScope.WriteString ":MTEST:DELETE"
```

:MTESt:ENABle

Command `:MTESt:ENABle {{ON | 1} | {OFF | 0}}`

The :MTESt:ENABle command enables or disables the mask test features.

ON Enables the mask test features.

OFF Disables the mask test features.

Example This example enables the mask test features.

```
myScope.WriteString ":MTEST:ENABLE ON"
```

Query `:MTESt:ENABle?`

The :MTESt:ENABle? query returns the current state of mask test features.

Returned Format `[MTESt:ENABle] {1 | 0}<NL>`

Example This example places the current value of the mask test state in the numeric variable varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":MTEST:ENABLE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MTESt:FOLDing

(Clock Recovery software only)

Command

NOTE This command is only available when the E2688A Clock Recovery Software is installed.

```
:MTESt:FOLDing {{ON | 1} | {OFF | 0}}
```

The :MTESt:FOLDing command enables (ON) or disables (OFF) the display of the real time eye. When enabled, an eye diagram of the data will be generated superimposed on the recovered clock. Refer to the :MEASure:CLOCk commands for clock recovery.

Example This example enables the display of the real time eye.

```
myScope.WriteString ":MTEST:FOLDING ON"
```

Query

```
:MTESt:FOLDing?
```

The :MTESt:FOLDing? query returns the current state of clock recovery folding.

Returned Format

```
[:MTESt:FOLDing] {1 | 0} <NL>
```

Example

```
myScope.WriteString ":MTEST:FOLDING?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:CLOCk" on page 501
- ":MEASure:CLOCk:METHod" on page 502
- ":MEASure:CLOCk:METHod:ALIGn" on page 504
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 505
- ":MEASure:CLOCk:METHod:EDGE" on page 506
- ":MEASure:CLOCk:METHod:JTF" on page 508
- ":MEASure:CLOCk:METHod:OJTF" on page 510

:MTESt:FOLDing:BITS

Command

NOTE This command is only available when the E2688A Clock Recovery Software is installed.

```
:MTESt:FOLDing:BITS {BOTH | DEEMphasis | TRANsition
                     | PATTern, "<pattern>", <cursor>}
```

The :MTESt:FOLDing:BITS command determines the type of data bits used to create the eye pattern. The transition bits are greater in amplitude than the deemphasis bits. The PCI Express standard requires that compliance mask testing be done for both bit types.

<pattern> An eight character string 8 of "1", "0", or "X". For example, "101XX010".

<cursor> A value from 0 to 7 representing which bit is bit 0 from the LSB.

Example This example sets bit type to transition bits.

```
myScope.WriteString ":MTESt:FOLDing:BITS TRANsition"
```

Query

```
:MTESt:FOLDing:BITS?
```

The :MTESt:FOLDing:BITS? query returns the current setting of the real time eye bits.

Returned Format

```
[:MTESt:FOLDing:BITS] {BOTH | DEEMphasis | TRANsition
                       | PATT,<pattern>,<cursor>} <NL>
```

Example

```
myScope.WriteString ":MTESt:FOLDing:BITS?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MTESt:FOLDing:FAST

Command `:MTESt:FOLDing:FAST {{ON | 1} | {OFF | 0}}`

The :MTESt:FOLDing:FAST command turns the "Fast, Worst Case Only" real-time eye display option ON or OFF.

When ON, the oscilloscope performs a fast real-time eye display by showing only the worst case bits and other bits that surround them.

This option can be used to speed up the real-time eye display when the oscilloscope is using deep memory.

Worst case bits are evaluated using these 8 points inside the eye:

- Left and right of the eye at the threshold level.
- Left and right of the eye at the threshold + hysteresis level.
- Left and right of the eye at the threshold - hysteresis level.
- Top and bottom of the eye at the center.

For each bit that represents the worst case at one of the evaluated points, the 1,000 bits that surround that bit are also displayed. So, up to 8,000 bits per acquisition can contribute to the real-time eye display. Keep in mind that one bit can represent the worst case at multiple points or that the 1,000 bits surrounding the worst case bits can overlap, so there can be fewer than 8,000 bits displayed per acquisition.

Example This example turns on the "Fast, Worst Case Only" real-time eye display option.

```
myScope.WriteString ":MTESt:FOLDing:FAST ON"
```

Query `:MTESt:FOLDing:FAST?`

The :MTESt:FOLDing:FAST? query returns the current setting.

Returned Format `[:MTESt:FOLDing:FAST] {1 | 0} <NL>`

Example

```
myScope.WriteString ":MTESt:FOLDing:FAST?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MTESt:HAMPlitude

Command `:MTESt:HAMPlitude <upper_limit>`

The :MTESt:HAMPlitude command sets the maximum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the upper limit used during mask testing.

<upper_limit> A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example sets the maximum pulse amplitude to 3.6 volts.

```
myScope.WriteString ":MTEST:HAMPLITUDE 3.6"
```

Query `:MTESt:HAMPlitude?`

The :MTESt:HAMPlitude? query returns the current value of the maximum pulse amplitude.

Returned Format `[MTESt:HAMPlitude] <upper_limit><NL>`

<upper_limit> A real number that represents the maximum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example returns the current upper pulse limit and prints it to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MTEST:HAMPLITUDE?"
varULimit = myScope.ReadNumber
Debug.Print FormatNumber(varULimit, 0)
```

:MTESt:IMPedance

Command `:MTESt:IMPedance {NONE | IMP75 | IMP100 | IMP110 | IMP120}`

The :MTESt:IMPedance command sets the desired probe impedance of the channel being used for mask testing. This impedance value is used when starting a mask test to determine whether or not the correct Infiniium probe is connected and in the case of the E2621A if the switch is set to the correct impedance value.

Infiniium has an AutoProbe interface that detects probes that have Probe ID resistors. If one of these probes is connected to the channel being mask tested and is not the correct probe for the selected impedance, a warning dialog box appears when the mask test is started from the human interface.

This command is meant to be used in the setup section of a mask file.

NONE Disables the probe impedance check.

IMP75 Enables the probe impedance check for the E2622A probe.

IMP100 Enables the probe impedance check for the E2621A probe with the switch set to the 100 ohm position.

IMP110 Enables the probe impedance check for the E2621A probe with the switch set to the 110 ohm position.

IMP120 Enables the probe impedance check for the E2621A probe with the switch set to the 120 ohm position.

Example This example sets the probe impedance of the channel being used for mask testing to 100 ohms.

```
myScope.WriteString ":MTEST:IMPEDANCE IMP100"
```

Query `:MTESt:IMPedance?`

The :MTESt:IMPedance? query returns the current value of the mask test impedance.

Returned Format `[:MTESt:IMPedance] {NONE | IMP75 | IMP100 | IMP110 | IMP120}<NL>`

Example This example returns the current value of the mask test impedance and prints the result to the computer screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MTEST:IMPEDANCE?"
strImpedance = myScope.ReadString
Debug.Print strImpedance
```

:MTESt:INVert

Command `:MTESt:INVert {{ON | 1} | {OFF | 0}}`

The :MTESt:INVert command inverts the mask for testing negative-going pulses. The trigger level and mask offset are also adjusted. Not all masks support negative-going pulse testing, and for these masks, the command is ignored.

Example This example inverts the mask for testing negative-going pulses.

```
myScope.WriteString ":MTEST:INVERT ON"
```

Query `:MTESt:INVert?`

The :MTESt:INVert? query returns the current inversion setting.

Returned Format `[:MTESt:INVert] {1 | 0}<NL>`

:MTESt:LAMPlitude

Command `:MTESt:LAMPlitude <lower_limit>`

The :MTESt:LAMPlitude command sets the minimum pulse amplitude value that passes the pulse standard. For some of the pulse communications standards, a pulse has a range of amplitude values and still passes the standard. This command sets the lower limit used during mask testing.

<lower_limit> A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example sets the minimum pulse amplitude to 2.4 volts.

```
myScope.WriteString ":MTEST:LAMPLITUDE 2.4"
```

Query `:MTESt:LAMPlitude?`

The :MTESt LAMPlitude? query returns the current value of the minimum pulse amplitude.

Returned Format `[:MTESt:LAMPlitude] <lower_limit><NL>`

<lower_limit> A real number that represents the minimum amplitude in volts of a pulse as allowed by the pulse standard.

Example This example returns the current lower pulse limit and prints it to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF  ! Response headers off.
myScope.WriteString ":MTEST:LAMPLITUDE?""
varULimit = myScope.ReadNumber
Debug.Print FormatNumber(varULimit, 0)
```

:MTESt:LOAD

Command `:MTESt:LOAD "<filename>"`

The :MTESt:LOAD command loads the specified mask file. The default path for mask files is C:\Documents and Settings\All Users\Documents\Infiniium\MASKS. To use a different path, specify the complete path and file name.

<filename> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used).

Example This example loads the mask file named "140md_itu_1.msk".

```
myScope.WriteString _
":MTEST:LOAD ""c:\Documents and Settings\All Users\Documents\Infiniium\
masks\140md_itu_1.msk"""
```

:MTESt:NREGions?

Query `:MTESt:NREGions?`

The :MTESt:NREGions? query returns the number of regions that define the mask.

Returned Format `[:MTESt:NREGions] <regions><NL>`

<regions> `An integer from 0 to 8.`

Example This example returns the number of mask regions.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":MTEST:NREGIONS?"
varRegions = myScope.ReadNumber
Debug.Print FormatNumber(varRegions, 0)
```

:MTESt:PROBe:IMPedance?

Query `:MTESt:PROBe:IMPedance?`

The :MTESt:PROBe:IMPedance? query returns the impedance setting for the E2621A and E2622A probes for the current mask test channel.

Returned Format `[:MTESt:PROBe:IMPedance] <impedance><NL>`

<impedance> An unquoted string: 75, 100, 110, 120, or NONE

Example This example returns the impedance setting for the probe.

```
Dim strImpedance As String
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":MTEST:PROBE:IMPEDANCE?"
strImpedance = myScope.ReadString
Debug.Print strImpedance
```

:MTESt:RUMode

Command `:MTESt:RUMode {FORever | TIME, <time> | {WAVeforms, <number_of_waveforms>}}`

The :MTESt:RUMode command determines the termination conditions for the mask test. The choices are FORever, TIME, or WAVeforms.

If WAVeforms is selected, a second parameter is required indicating the number of failures that can occur or the number of samples or waveforms that are to be acquired.

FORever FORever runs the Mask Test until the test is turned off. This is used when you want a measurement to run continually and not to stop after a fixed number of failures. For example, you may want the Mask Test to run overnight and not be limited by a number of failures.

TIME TIME sets the amount of time in minutes that a mask test will run before it terminates.

<time> A real number: 0.1 to 1440.0

WAVeforms WAVeforms sets the maximum number of waveforms that are required before the mask test terminates.

<number_of_waveforms> An integer: 1 to 1,000,000,000.

Example This example sets the mask test subsystem run until mode to continue testing until 500,000 waveforms have been gathered.

```
myScope.WriteString ":MTEST:RUMODE WAVEFORMS,500E3"
```

Query `:MTESt:RUMode?`

The query returns the currently selected termination condition and value.

Returned Format `[:MTESt:RUMode] {FORever | TIME,<time> | {WAVeforms, <number_of_waveforms>}}<NL>`

Example This example gets the current setting of the mask test run until mode from the oscilloscope and prints it on the computer screen.

```
Dim strMTEST_Runmode As String
myScope.WriteString ":MTEST:RUMODE?"
strMTEST_Runmode = myScope.ReadString
Debug.Print strMTEST_Runmode
```

:MTESt:RUMode:SOFailure

Command `:MTESt:RUMode:SOFailure {{ON | 1} | {OFF | 0}}`

The :MTESt:RUMode:SOFailure command enables or disables the Stop On Failure run until criteria. When a mask test is run and a mask violation is detected, the mask test is stopped and the acquisition system is stopped.

Example This example enables the Stop On Failure run until criteria.

```
myScope.WriteString ":MTEST:RUMODE:SOFAILURE ON"
```

Query `:MTESt:SOFailure?`

The :MTESt:SOFailure? query returns the current state of the Stop on Failure control.

Returned Format `[:MTESt:SOFailure] {1 | 0}<NL>`

:MTESt:SCALe:BIND

Command `:MTESt:SCALe:BIND {{ON | 1} | {OFF | 0}}`

The :MTESt:SCALe:BIND command enables or disables Bind 1 & 0 Levels (Bind -1 & 0 Levels for inverted masks) control. If the Bind 1 & 0 Levels control is enabled, the 1 Level and the 0 Level controls track each other. Adjusting either the 1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind 1 & 0 Levels control is disabled, adjusting either the 1 Level or the 0 Level control changes the vertical height of the mask.

If the Bind -1 & 0 Levels control is enabled, the -1 Level and the 0 Level controls track each other. Adjusting either the -1 Level or the 0 Level control shifts the position of the mask up or down without changing its size. If the Bind -1 & 0 Levels control is disabled, adjusting either the -1 Level or the 0 Level control changes the vertical height of the mask.

Example This example enables the Bind 1 & 0 Levels control.

```
myScope.WriteString ":MTEST:SCALE:BIND ON"
```

Query `:MTESt:SCALe:BIND?`

The :MTESt:SCALe:BIND? query returns the value of the Bind 1&0 control (Bind -1&0 for inverted masks).

Returned Format `[:MTESt:SCALe:BIND?] {1 | 0}<NL>`

:MTESt:SCALe:X1

Command `:MTESt:SCALe:X1 <x1_value>`

The :MTESt:SCALe:X1 command defines where X=0 in the base coordinate system used for mask testing. The other X-coordinate is defined by the SCALe:XDELta command. Once the X1 and XDELta coordinates are set, all X values of vertices in the mask regions are defined with respect to this value, according to the equation:

$$X = (X \times \Delta X) + X1$$

Thus, if you set X1 to 100 ms, and XDELta to 100 ms, an X value of 0.100 is a vertex at 110 ms.

The oscilloscope uses this equation to normalize vertices. This simplifies reprogramming to handle different data rates. For example, if you halve the period of the waveform of interest, you need only to adjust the XDELta value to set up the mask for the new waveform.

<x1_value> A time value specifying the location of the X1 coordinate, which will then be treated as X=0 for mask regions coordinates.

Example This example sets the X1 coordinate at 150 ms.

```
myScope.WriteString ":MTEST:SCALE:X1 150E-3"
```

Query `:MTESt:SCALe:X1?`

The :MTESt:SCALe:X1? query returns the current X1 coordinate setting.

Returned Format `[:MTESt:SCALe:X1] <x1_value><NL>`

Example This example gets the current setting of the X1 coordinate from the oscilloscope and prints it on the computer screen.

```
Dim strScale_x1 As String
myScope.WriteString ":MTEST:SCALE:X1?"
strScale_x1 = myScope.ReadString
Debug.Print strScale_x1
```

:MTESt:SCALe:XDELta

Command `:MTESt:SCALe:XDELta <xdelta_value>`

The :MTESt:SCALe:XDELta command defines the position of the X2 marker with respect to the X1 marker. In the mask test coordinate system, the X1 marker defines where X=0; thus, the X2 marker defines where X=1.

Because all X vertices of the regions defined for mask testing are normalized with respect to X1 and ΔX, redefining ΔX also moves those vertices to stay in the same locations with respect to X1 and ΔX. Thus, in many applications, it is best if you define XDELta as a pulse width or bit period. Then a change in data rate without corresponding changes in the waveform can easily be handled by changing ΔX.

The X-coordinate of polygon vertices is normalized using this equation:

$$X = (X \times \Delta X) + X1$$

<xdelta_value> A time value specifying the distance of the X2 marker with respect to the X1 marker.

Example Assume that the period of the waveform you wish to test is 1 ms. Then the following example will set ΔX to 1 ms, ensuring that the waveform's period is between the X1 and X2 markers.

```
myScope.WriteString ":MTEST:SCALE:XDELTA 1E-6:
```

Query `:MTESt:SCALe:XDELta?`

The :MTESt:SCALe:XDELta? query returns the current value of ΔX.

Returned Format `[:MTESt:SCALe:XDELta] <xdelta_value><NL>`

Example This example gets the value of ΔX from the oscilloscope and prints it on the computer screen.

```
Dim strScale_xdelta As String
myScope.WriteString ":MTEST:SCALE:XDELTA?""
strScale_xdelta = myScope.ReadString
Debug.Print strScale_xdelta
```

:MTESt:SCALe:Y1

Command `:MTESt:SCALe:Y1 <y_value>`

The :MTESt:SCALe:Y1 command defines where Y=0 in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries set by SCALe:Y1 and SCALe:Y2 according to the equation:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

<y1_value> A voltage value specifying the point at which Y=0.

Example This example sets the Y1 marker to -150 mV.

```
myScope.WriteString ":MTEST:SCALE:Y1 -150E-3"
```

Query `:MTESt:SCALe:Y1?`

The SCALe:Y1? query returns the current setting of the Y1 marker.

Returned Format `[:MTESt:SCALe:Y1] <y1_value><NL>`

Example This example gets the setting of the Y1 marker from the oscilloscope and prints it on the computer screen.

```
Dim strScale_y1 As String
myScope.WriteString ":MTEST:SCALE:Y1?"
strScale_y1 = myScope.ReadString
Debug.Print strScale_y1
```

:MTESt:SCALe:Y2

Command `:MTESt:SCALe:Y2 <y2_value>`

The :MTESt:SCALe:Y2 command defines the Y2 marker in the coordinate system for mask testing. All Y values of vertices in the coordinate system are defined with respect to the boundaries defined by SCALe:Y1 and SCALe:Y2 according to the following equation:

$$Y = (Y \times (Y2 - Y1)) + Y1$$

Thus, if you set Y1 to 100 mV, and Y2 to 1 V, a Y value of 0.100 in a vertex is at 190 mV.

<y2_value> A voltage value specifying the location of the Y2 marker.

Example This example sets the Y2 marker to 2.5 V.

```
myScope.WriteString ":MTEST:SCALE:Y2 2.5"
```

Query `:MTESt:SCALe:Y2?`

The SCALe:Y2? query returns the current setting of the Y2 marker.

Returned Format `[:MTESt:SCALe:Y2] <y2_value><NL>`

Example This example gets the setting of the Y2 marker from the oscilloscope and prints it on the computer screen.

```
Dim strScale_y2 As String
myScope.WriteString ":MTEST:SCALE:Y2?"
strScale_y2 = myScope.ReadString
Debug.Print strScale_y2
```

:MTESt:SOURce

Command `:MTESt:SOURce {CHANnel<N> | FUNCtion<F> | EQUalized}`

The :MTESt:SOURce command selects the channel which is configured by the commands contained in a mask file when it is loaded.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> An integer, 1-4.

<F> An integer, 1-16.

Example This example selects channel 1 as the mask test source.

```
myScope.WriteString "MTEST:SOURCE CHANNEL1"
```

Query `:MTESt:SOURce?`

The :MTESt:SOURce? query returns the channel which is configured by the commands contained in the current mask file.

Returned Format `[:MTESt:SOURce] {CHANnel<N> | FUNCtion<F> | EQUalized}<NL>`

Example This example gets the mask test source setting and prints the result on the computer display.

```
Dim strAmask_source As String
myScope.WriteString ":MTEST:SOURCE?"
strAmask_source = myScope.ReadString
Debug.Print strAmask_source
```

:MTESt:STARt

Command `:MTESt:STARt`

The :MTESt:STARt command starts the mask test. The :MTESt:STARt command also starts the oscilloscope acquisition system.

Example This example starts the mask test and acquisition system.

```
myScope.WriteString ":MTEST:START"
```

:MTESt:STOP

Command `:MTESt:STOP`

The :MTESt:STOP command stops the mask test. The :MTESt:STOP command does not stop the acquisition system.

Example This example stops the mask test.

```
myScope.WriteString ":MTEST:STOP"
```

:MTESt:STIMe

Command `:MTESt:STIMe <timeout>`

The :MTESt:STIMe command sets the timeout value for the Autoalign feature. If the oscilloscope is unable to align the mask to your waveform within the specified timeout value, it will stop trying to align and will report an alignment failure.

<timeout> An integer from 1 to 120 seconds representing the time between triggers (not the time that it takes to finish the alignment.)

Example This example sets the timeout value for the Autoalign feature to 10 seconds.

```
myScope.WriteString ":MTEST:STIMe 10"
```

Query `:MTESt:STIMe?`

The query returns timeout value for the Autoalign feature.

Returned Format `[:MTESt:STIMe] <timeout><NL>`

Example This example gets the timeout setting and prints the result on the computer display.

```
myScope.WriteString ":MTEST:STIME?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MTESt:TITLe?

Query `:MTESt:TITLe?`

The :MTESt:TITLe? query returns the mask title which is a string of up to 23 characters. The title is displayed in the mask test dialog box and mask test tab when a mask file is loaded.

Returned Format `[:MTESt:TITLe] <mask_title><NL>`

<mask_title> A string of up to 23 ASCII characters which is the mask title.

Example This example places the mask title in the string variable and prints the contents to the computer's screen.

```
Dim strTitle As String
myScope.WriteString ":MTEST:TITLE?"
strTitle = myScope.ReadString
Debug.Print strTitle
```

:MTESt:TRIGger:SOURce

Command `:MTESt:TRIGger:SOURce CHANnel<N>`

The :MTESt:TRIGger:SOURce command sets the channel or function to use as the trigger. Mask testing must be enabled before using this command.

<N> An integer, 1-4.

Example This example sets the mask trigger source to channel 1.

```
myScope.WriteString ":MTEST:TRIGGER:SOURCE CHANNEL1"
```

Query `:MTESt:TRIGger:SOURce?`

The query returns the currently selected mask test trigger source.

Returned Format `[:MTESt:TRIGger] CHANnel<N><NL>`

Example This example gets the trigger source setting and prints the result on the computer display.

```
Dim strAmask_source As String
myScope.WriteString ":MTEST:TRIGGER:SOURCE?"
strAmask_source = myScope.ReadString
Debug.Print strAmask_source
```

25 Measure Commands

:MEASure:AREA 488
:MEASure:BINTerval 489
:MEASure:BPERiod 490
:MEASure:BWIDth 491
:MEASure:CDRRATE 492
:MEASure:CGRade:CROSsing 493
:MEASure:CGRade:DCDistortion 494
:MEASure:CGRade:EHEight 495
:MEASure:CGRade:EWIDth 496
:MEASure:CGRade:EWINdow 497
:MEASure:CGRade:JITTer 498
:MEASure:CGRade:QFACtor 499
:MEASure:CLEar 500
:MEASure:CLOCk 501
:MEASure:CLOCk:METHod 502
:MEASure:CLOCk:METHod:ALIGn 504
:MEASure:CLOCk:METHod:DEEMphasis 505
:MEASure:CLOCk:METHod:EDGE 506
:MEASure:CLOCk:METHod:JTF 508
:MEASure:CLOCk:METHod:OJTF 510
:MEASure:CLOCk:VERTical 512
:MEASure:CLOCk:VERTical:OFFSet 513
:MEASure:CLOCk:VERTical:RANGe 514
:MEASure:CROSsing 515
:MEASure:CTCDutycycle 516
:MEASure:CTCJitter 518
:MEASure:CTCNwidth 520
:MEASure:CTCPwidth 522
:MEASure:DATarate 524
:MEASure:DEEMphasis 526
:MEASure:DELTatime 528
:MEASure:DELTatime:DEFine 530
:MEASure:DUTYcycle 532
:MEASure:EDGE 534

:MEASure:ETOedge 536
:MEASure:FALLtime 537
:MEASure:FFT:DFRequency 539
:MEASure:FFT:DMAGnitude 541
:MEASure:FFT:FREQuency 543
:MEASure:FFT:MAGNitude 544
:MEASure:FFT:PEAK1 545
:MEASure:FFT:PEAK2 546
:MEASure:FFT:THReshold 547
:MEASure:FREQuency 548
:MEASure:HISTogram:HITS 550
:MEASure:HISTogram:M1S 551
:MEASure:HISTogram:M2S 552
:MEASure:HISTogram:M3S 553
:MEASure:HISTogram:MAX 554
:MEASure:HISTogram:MEAN 555
:MEASure:HISTogram:MEDian 556
:MEASure:HISTogram:MIN 557
:MEASure:HISTogram:MODE 558
:MEASure:HISTogram:PEAK 559
:MEASure:HISTogram:PP 560
:MEASure:HISTogram:RESolution 561
:MEASure:HISTogram:STDDev 562
:MEASure:HOLDtime 563
:MEASure:JITTer:HISTogram 565
:MEASure:JITTer:MEASurement 566
:MEASure:JITTer:SPECtrum 567
:MEASure:JITTer:SPECtrum:HORizontal 568
:MEASure:JITTer:SPECtrum:HORizontal:POSition 569
:MEASure:JITTer:SPECtrum:HORizontal:RANGe 570
:MEASure:JITTer:SPECtrum:VERTical 571
:MEASure:JITTer:SPECtrum:VERTical:OFFSet 572
:MEASure:JITTer:SPECtrum:VERTical:RANGe 573
:MEASure:JITTer:SPECtrum:VERTical:TYPE 574
:MEASure:JITTer:SPECtrum:WINDow 575
:MEASure:JITTer:STATistics 576
:MEASure:JITTer:TRENd 577
:MEASure:JITTer:TRENd:SMOoth 578
:MEASure:JITTer:TRENd:SMOoth:POINts 579

:MEASure:JITTer:TRENd:VERTical 580
:MEASure:JITTer:TRENd:VERTical:OFFSet 581
:MEASure:JITTer:TRENd:VERTical:RANGe 582
:MEASure:NAME 583
:MEASure:NCJitter 584
:MEASure:NOISe 586
:MEASure:NOISe:ALL? 587
:MEASure:NOISe:BANDwidth 589
:MEASure:NOISe:LOCation 590
:MEASure:NOISe:METHod 591
:MEASure:NOISe:REPort 592
:MEASure:NOISe:RN 593
:MEASure:NOISe:SCOPe:RN 594
:MEASure:NOISe:STATe 595
:MEASure:NOISe:UNITs 596
:MEASure:NPERiod 597
:MEASure:NPULses 598
:MEASure:NUI 599
:MEASure:NWIDth 600
:MEASure:OVERshoot 602
:MEASure:PAMPlitude 604
:MEASure:PBASe 605
:MEASure:PERiod 606
:MEASure:PHASe 608
:MEASure:PPULses 610
:MEASure:PREShoot 611
:MEASure:PTOP 613
:MEASure:PWIDth 614
:MEASure:QUALifier<M>:CONDition 616
:MEASure:QUALifier<M>:SOURce 617
:MEASure:QUALifier<M>:STATe 618
:MEASure:RESults? 619
:MEASure:RISetime 622
:MEASure:RJDJ:ALL? 624
:MEASure:RJDJ:APLength? 626
:MEASure:RJDJ:BANDwidth 627
:MEASure:RJDJ:BER 628
:MEASure:RJDJ:CLOCk 630
:MEASure:RJDJ:EDGE 631

:MEASure:RJDJ:INTerpolate 632
:MEASure:RJDJ:METHod 633
:MEASure:RJDJ:MODe 634
:MEASure:RJDJ:PLENgth 635
:MEASure:RJDJ:REPort 636
:MEASure:RJDJ:RJ 637
:MEASure:RJDJ:SCOPe:RJ 638
:MEASure:RJDJ:SOURce 639
:MEASure:RJDJ:STATe 640
:MEASure:RJDJ:TJRJDJ? 641
:MEASure:RJDJ:UNITs 642
:MEASure:SCRatch 643
:MEASure:SENDvalid 644
:MEASure:SETuptime 645
:MEASure:SLEWrate 647
:MEASure:SOURce 648
:MEASure:STATistics 649
:MEASure:TEDGe 650
:MEASure:THResholds:ABSolute 652
:MEASure:THResholds:GENeral:ABSolute 654
:MEASure:THResholds:GENeral:HYSTeresis 656
:MEASure:THResholds:GENeral:METHod 658
:MEASure:THResholds:GENeral:PERCent 660
:MEASure:THResholds:GENeral:TOPBase:ABSolute 662
:MEASure:THResholds:GENeral:TOPBase:METHod 664
:MEASure:THResholds:HYSTeresis 666
:MEASure:THResholds:METHod 668
:MEASure:THResholds:PERCent 670
:MEASure:THResholds:RFALl:ABSolute 672
:MEASure:THResholds:RFALl:HYSTeresis 674
:MEASure:THResholds:RFALl:METHod 676
:MEASure:THResholds:RFALl:PERCent 678
:MEASure:THResholds:RFALl:TOPBase:ABSolute 680
:MEASure:THResholds:RFALl:TOPBase:METHod 682
:MEASure:THResholds:SERial:ABSolute 684
:MEASure:THResholds:SERial:HYSTeresis 686
:MEASure:THResholds:SERial:METHod 688
:MEASure:THResholds:SERial:PERCent 690
:MEASure:THResholds:SERial:TOPBase:ABSolute 692

:MEASure:THResholds:SERial:TOPBase:METHod 694
:MEASure:THResholds:TOPBase:ABSolute 696
:MEASure:THResholds:TOPBase:METHod 698
:MEASure:TIEClock2 700
:MEASure:TIEData 702
:MEASure:TIEFilter:SHAPe 704
:MEASure:TIEFilter:STARt 705
:MEASure:TIEFilter:STATe 706
:MEASure:TIEFilter:STOP 707
:MEASure:TIEFilter:TYPE 708
:MEASure:TMAX 709
:MEASure:TMIN 710
:MEASure:TVOLt 711
:MEASure:UITouijitter 713
:MEASure:UNITinterval 714
:MEASure:VAMPlitude 716
:MEASure:VAVerage 717
:MEASure:VBASe 719
:MEASure:VLOWer 720
:MEASure:VMAX 721
:MEASure:VMIDdle 723
:MEASure:VMIN 724
:MEASure:VOVershoot 726
:MEASure:VPP 727
:MEASure:VPReshoot 729
:MEASure:VRMS 730
:MEASure:VTIMe 732
:MEASure:VTOP 734
:MEASure:VUPPer 735
:MEASure:WINDow 737
:MEASurement<N>:NAME 738
:MEASurement<N>:SOURce 739

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

E2688A High Speed Serial Software commands

The following MEASure commands are available when the E2688A High Speed Serial Software is installed.

- ":MEASure:CLOCk" on page 501
- ":MEASure:CLOCk:METHod" on page 502
- ":MEASure:CLOCk:METHod:ALIGn" on page 504
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 505
- ":MEASure:CLOCk:METHod:EDGE" on page 506
- ":MEASure:CLOCk:METHod:JTF" on page 508
- ":MEASure:CLOCk:METHod:OJTF" on page 510
- ":MEASure:CLOCk:VERTical" on page 512
- ":MEASure:CLOCk:VERTical:OFFSet" on page 513
- ":MEASure:CLOCk:VERTical:RANGe" on page 514
- ":MEASure:TIEData" on page 702
- ":MEASure:TIEFilter:SHAPe" on page 704
- ":MEASure:TIEFilter:STARt" on page 705
- ":MEASure:TIEFilter:STATe" on page 706
- ":MEASure:TIEFilter:STOP" on page 707
- ":MEASure:TIEFilter:TYPE" on page 708
- Also see the ":MTESt:FOLDing" on page 456 command in the mask test system.

E2681A EZJIT Jitter Analysis Software commands

The following MEASure commands are available when the E2681A EZJIT Jitter Analysis Software is installed.

- ":MEASure:CTCDutycycle" on page 516
- ":MEASure:CTCJitter" on page 518
- ":MEASure:CTCNwidth" on page 520
- ":MEASure:CTCPwidth" on page 522
- ":MEASure:DATarate" on page 524
- ":MEASure:HOLDtime" on page 563
- ":MEASure:JITTer:HISTogram" on page 565
- ":MEASure:JITTer:MEASurement" on page 566
- ":MEASure:JITTer:SPECtrum" on page 567
- ":MEASure:JITTer:STATistics" on page 576
- ":MEASure:JITTer:TRENd" on page 577
- ":MEASure:NCJitter" on page 584
- ":MEASure:NPERiod" on page 597

- ":MEASure:NUI" on page 599
- ":MEASure:SETuptime" on page 645
- ":MEASure:TIEClock2" on page 700
- ":MEASure:TIEData" on page 702
- ":MEASure:UITouijitter" on page 713
- ":MEASure:UNITinterval" on page 714

N5400A and N5401A Jitter Analysis Software commands

The following MEASure commands are available when the N5400A or N5401A Jitter Analysis Software is installed.

- ":MEASure:CLOCk" on page 501
- ":MEASure:CLOCk:METHod" on page 502
- ":MEASure:CLOCk:METHod:ALIGn" on page 504
- ":MEASure:CLOCk:METHod:DEEMphasis" on page 505
- ":MEASure:CLOCk:METHod:EDGE" on page 506
- ":MEASure:CLOCk:METHod:JTF" on page 508
- ":MEASure:CLOCk:METHod:OJTF" on page 510
- ":MEASure:CLOCk:VERTical" on page 512
- ":MEASure:CLOCk:VERTical:OFFSet" on page 513
- ":MEASure:CLOCk:VERTical:RANGe" on page 514
- ":MEASure:CTCDutycycle" on page 516
- ":MEASure:CTCJitter" on page 518
- ":MEASure:CTCNwidth" on page 520
- ":MEASure:CTCPwidth" on page 522
- ":MEASure:DATarate" on page 524
- ":MEASure:HOLDtime" on page 563
- ":MEASure:JITTer:HISTogram" on page 565
- ":MEASure:JITTer:MEASurement" on page 566
- ":MEASure:JITTer:SPECtrum" on page 567
- ":MEASure:JITTer:STATistics" on page 576
- ":MEASure:JITTer:TRENd" on page 577
- ":MEASure:NCJitter" on page 584
- ":MEASure:RJDJ:ALL?" on page 624
- ":MEASure:RJDJ:APLength?" on page 626
- ":MEASure:RJDJ:BER" on page 628
- ":MEASure:RJDJ:CLOCk" on page 630
- ":MEASure:RJDJ:EDGE" on page 631

- ":MEASure:RJDJ:INTerpolate" on page 632
- ":MEASure:RJDJ:PLENgth" on page 635
- ":MEASure:RJDJ:SOURce" on page 639
- ":MEASure:RJDJ:STATe" on page 640
- ":MEASure:RJDJ:TJRJDJ?" on page 641
- ":MEASure:RJDJ:UNITs" on page 642
- ":MEASure:SETuptime" on page 645
- ":MEASure:TIEClock2" on page 700
- ":MEASure:TIEData" on page 702
- ":MEASure:UNITinterval" on page 714

FFT Commands

The :MEASure:FFT commands control the FFT measurements that are accessible through the Measure subsystem.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope.

- For a period or frequency measurement, at least one and a half complete cycles must be displayed.
- For a pulse width measurement, the entire pulse must be displayed.
- For a rise time measurement, the leading (positive-going) edge of the waveform must be displayed.
- For a fall time measurement, the trailing (negative-going) edge of the waveform must be displayed.

In jitter mode with jitter statistics enabled, measurements are made on all data regardless of what is on screen.

User-Defined Thresholds

If you choose to set user-defined thresholds, they must be set before actually sending the measurement command or query.

Measurement Error

If a measurement cannot be made because of a lack of data, because the source waveform is not displayed, the requested measurement is not possible (for example, a period measurement on an FFT waveform), or for some other reason, the following results are returned:

- 9.99999E+37 is returned as the measurement result.
- If SENDvalid is ON, the error code is also returned as well as the questionable value.

Making Measurements

If more than one period, edge, or pulse is displayed, time measurements are made on the first, left-most portion of the displayed waveform.

When any of the defined measurements are requested, the oscilloscope first determines the top (100%) and base (0%) voltages of the waveform. From this information, the oscilloscope determines the other important voltage values (10%, 90%, and 50% voltage values) for making measurements.

The 10% and 90% voltage values are used in the rise time and fall time measurements when standard thresholds are selected. The 50% voltage value is used for measuring frequency, period, pulse width, and duty cycle with standard thresholds selected.

You can also make measurements using user-defined thresholds instead of the standard thresholds.

When the command form of a measurement is used, the oscilloscope is placed in the continuous measurement mode. The measurement result will be displayed on the front panel. There may be a maximum of 5 measurements running continuously. Use the SCRatch command to turn off the measurements.

When the query form of the measurement is used, the measurement is made one time, and the measurement result is returned.

- If the current acquisition is complete, the current acquisition is measured and the result is returned.
- If the current acquisition is incomplete and the oscilloscope is running, acquisitions will continue to occur until the acquisition is complete. The acquisition will then be measured and the result returned.
- If the current acquisition is incomplete and the oscilloscope is stopped, the measurement result will be 9.99999e+37 and the incomplete result state will be returned if SENDvalid is ON.

All measurements are made using the entire display, except for VAVerage and VRMS which allow measurements on a single cycle. Therefore, if you want to make measurements on a particular cycle, display only that cycle on the screen.

Measurements are made on the displayed waveforms specified by the SOURce command. The SOURce command lets you specify two sources. Most measurements are only made on a single source. Some measurements, such as the DELTatime measurement, require two sources.

If the waveform is clipped, the measurement result may be questionable. In this case, the value returned is the most accurate value that can be made using the current scaling. You might be able to obtain a more accurate measurement by adjusting the vertical scale to prevent the waveform from being clipped.

:MEASure:AREA

Command `:MEASure:AREA {CYCLe | DISPlay}[,<source>]`

The :MEASure:AREA command turns on the area measurement. The area measurement measures between the waveform, or a selected cycle of the waveform, and the waveform ground. When measuring Area, it is sometimes useful to use the Subtract Math Operator to remove any dc offset from a waveform you want to measure. Also see Math/FFT Functions for more details.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example turns on the area measurement which measures between the waveform and ground. Only that portion of the waveform which is in the waveform viewing area is measured.

```
myScope.WriteString ":MEASURE:AREA DISPLAY"
```

Query `:MEASure:AREA?`

The :MEASure:AREA? query returns the area measurement.

Returned Format `[:MEASure:AREA]<value>[,<result_state>]<NL>`

Example This example places the current selection for the area to be measured in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String
myScope.WriteString ":MEASure:AREA?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:MEASure:BINTerval

Command `:MEASure:BINTerval <source>, <idle time>`

The :MEASure:BINTerval command measures the amount of time between the end of a burst and beginning of the next burst. The idle time is the minimum time between bursts.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<idle time> Minimum amount of idle time between bursts.

Example This example measures the burst interval between two bursts on channel 4 (and with an idle time of 5 microseconds)

```
myScope.WriteString ":MEASURE:BINTerval CHAN4, 5e-6"
```

Query `:MEASure:BINTerval? <source>, <idle time>`

The :MEASure:BINTerval? query returns the burst interval time.

:MEASure:BPERiod

Command `:MEASure:BPERiod <source>, <idle time>`

The :MEASure:BPERiod command measures the time between the beginning of a burst and the beginning of the next burst. The idle time is the minimum time between bursts.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<idle time> Minimum amount of idle time between bursts.

Example This example measures the burst period between two bursts on channel 4 (and with an idle time of 5 microseconds)

```
myScope.WriteString ":MEASURE:BPERiod CHAN4, 5e-6"
```

Query `:MEASure:BPERiod? <source>, <idle time>`

The :MEASure:BPERiod? query returns the burst period time.

:MEASure:BWIDth

Command `:MEASure:BWIDth <source>,<idle_time>`

The :MEASure:BWIDth command measures the width of bursts in your waveform. The idle time is the minimum time between bursts.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MSPectrum | MTRend | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> An integer, 1-4.

<F> An integer, 1-16.

<idle_time> Amount of idle time between bursts.

Example This example measures the width of bursts for the waveform on channel one and sets the idle time to 1 microsecond.

```
myScope.WriteString ":MEASURE:BWIDTH CHANNEL1,1E-6"
```

Query `:MEASure:BWIDth? <source>,<idle_time>`

The :MEASure:BWIDth? query returns the width of the burst being measured.

Returned Format `[:MEASure:BWIDth ]<burst_width><NL>`

Example This example returns the width of the burst being measured, in the string variable, strBurstwidth, then prints the contents of the variable to the computer's screen.

```
Dim strBurstwidth As String
myScope.WriteString ":MEASure:BWIDTH? CHANNEL1,1E-6"
strBurstwidth = myScope.ReadString
Debug.Print strBurstwidth
```

:MEASure:CDRRATE

Command `:MEASure:CDRRATE <source>`

The :MEASure:CDRRATE command determines the data rate (clock recovery rate) from the clock recovery method being used. It yields one data point per acquisition so trending cannot be performed on this measurement.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MSPectrum | MTRend | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the clock recovery rate of channel 1.

```
myScope.WriteString ":MEASURE:CDRRATE CHANNEL1"
```

Example This example places the current data rate of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CDRRATE? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:CROSsing

Command `:MEASure:CGRade:CROSsing`

The :MEASure:CGRade:CROSsing command enables the crossing level percent measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example This example measures the crossing level.

```
myScope.WriteString ":MEASURE:CGRADE:CROSSING"
```

Query `:MEASure:CGRade:CROSsing?`

The :MEASure:CGRade:CROSsing? query returns the crossing level percent measurement of the current eye diagram on the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:CROSsing]<value>[,<result_state>]<NL>`

<value> The crossing level.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current crossing level in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:CROSSING?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:DCDistortion

Command `:MEASure:CGRade:DCDistortion <format>`

The :MEASure:CGRade:DCDistortion command enables the duty cycle distortion measurement on the current eye pattern. The parameter specifies the format for reporting the measurement. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<format> {TIME | PERCent}

Example This example measures the duty cycle distortion.

```
myScope.WriteString ":MEASURE:CGRADE:DCDISTORTION TIME"
```

Query `:MEASure:CGRade:DCDistortion? <format>`

The :MEASure:CGRade:DCDistortion query returns the duty cycle distortion measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:DCDistortion]<value>[,<result_state>]<NL>`

<value> The duty cycle distortion.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current duty cycle distortion in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"
myScope.WriteString ":MEASURE:CGRADE:DCDISTORTION? PERCENT"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:EHEight

Command `:MEASure:CGRade:EHEight <algorithm>`

The :MEASure:CGRade:EHEight command enables the eye height measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<algorithm> {MEASured | EXTRapolated} EXTRapolated is optional because it is the default if you do not specify an algorithm.

MEASured will measure the eye height within the window (see CGRade:EWINdow) of the current data. The smallest eye height is reported. Extrapolated will estimate the eye height based upon the mean and standard deviation of the eye top and base.

Example This example enables the eye height measurement.

```
myScope.WriteString ":MEASURE:CGRADE:EHEIGHT"
```

Query `:MEASure:CGRade:EHEight?`

The :MEASure:CGRade:EHEight? query returns the eye height measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:EHEight]<value>[,<result_state>]<NL>`

<value> The eye height.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current eye height in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:EHEIGHT?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:EWIDth

Command `:MEASure:CGRade:EWIDth <algorithm>`

The :MEASure:CGRade:EWIDth command enables the eye width measurement on the current eye pattern. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

<algorithm> {MEASured | EXTRapolated} EXTRapolated is optional because it is the default if you do not specify an algorithm.

MEASured will measure the eye width within the window (see CGRade:EWINdow) of the current data. The smallest eye width is reported. Extrapolated will estimate the eye width based upon the mean and standard deviation of the crossings.

Example This example measures the eye width.

```
myScope.WriteString ":MEASURE:CGRADE:EWIDTH"
```

Query `:MEASure:CGRade:EWIDth?`

The :MEASure:CGRade:EWIDth? query returns the eye width measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:EWIDth]<value>[,<result_state>]<NL>`

<value> The eye width.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current eye width in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:EWIDTH?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:EWINdow

Command `:MEASure:CGRade:EWINdow <start>,<stop>[,<start_after>]`

The :MEASure:CGRade:EWINdow command is used to change the starting point and the stopping point of the window used to make the eye pattern measurements of eye height, eye crossing %, and eye q-factor. In addition, the number of waveform hits can be set to ensure that enough data has been collected to make accurate measurements.

<start> An integer from 1 to 100 for horizontal starting point. (Default value is 40%.)

<stop> An integer from 1 to 100 for horizontal stopping point. (Default value is 60%.)

<start_after> An integer from 1 to 63,488 for number of hits to acquire before making measurements. (Default value is 1.)

Example This example sets the eye window starting point to 2%, the stopping point to 75% and the start after to 5,000 hits.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:EWINDOW 2,75,5000"
```

Query `:MEASure:CGRade:EWINdow?`

The :MEASure:CGRade:EWINdow query returns the starting point, the ending point, and the start after setting for the eye pattern measurements.

Returned Format `[:MEASure:CGRade:EWIDdow] <start>,<stop>,<start_after> <NL>`

The following example returns the values for the eye window.

Example

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:EWINDOW?"
varStart,Stop,Startafter = myScope.ReadNumber
Debug.Print FormatNumber(varStart,Stop,Startafter, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:CGRade:JITTer

Command `:MEASure:CGRade:JITTer <format>`

The :MEASure:CGRade:JITTer measures the jitter at the eye diagram crossing point. The parameter specifies the format, peak-to-peak or RMS, of the returned results. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

<format> {PP | RMS}

Example This example measures the jitter.

```
myScope.WriteString ":MEASURE:CGRADE:JITTER RMS"
```

Query `:MEASure:CGRade:JITTer? <format>`

The :MEASure:CGRade:JITTer? query returns the jitter measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:JITTer]<value>[,<result_state>]<NL>`

<value> The jitter.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the current jitter in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CGRADE:JITTER? RMS"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CGRade:QFACtor

Command `:MEASure:CGRade:QFACtor`

The :MEASure:CGRade:QFACtor command measures the Q factor. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature. Also, there must be a full eye diagram on screen before a valid measurement can be made.

Example This example measures the Q factor.

```
myScope.WriteString ":MEASURE:CGRade:QFACTOR"
```

Query `:MEASure:CGRade:QFACtor?`

The :MEASure:CGRade:QFACtor? query returns the Q factor measurement of the color grade display. Before using this command or query, you must use the :DISPlay:CGRade command to enable the color grade persistence feature.

Returned Format `[:MEASure:CGRade:QFACtor]<value>[,<result_state>]<NL>`

<value> The Q factor.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the Q factor in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CGRade:QFACTOR"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CLEar

Command `:MEASure:{CLEar | SCRatch}`

The :MEASure:CLEar command clears the measurement results from the screen and disables all previously enabled measurements.

Example This example clears the current measurement results from the screen.

```
myScope.WriteString ":MEASURE:CLEAR"
```

:MEASure:CLOCk

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk {{{ON|1},CHANnel<N>} | {OFF|0}}
```

The :MEASure:CLOCk command turns the recovered clock display on or off and sets the clock recovery channel source.

<N> An integer, 1-4.

Example This example turns the recovered clock display on for channel 1.

```
myScope.WriteString ":MEASURE:CLOCK ON,CHANNEL1"
```

Query

```
:MEASure:CLOCk?
```

The :MEASure :CLOCk? query returns the state of the recovered clock display.

Returned Format

```
[:MEASure:CLOCk] {1 | 0}<NL>
```

Example This example places the current setting of the recovered clock display in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:CLOCK?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:MEASure:CLOCk:METHod

Command

NOTE

:MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod
    {PCIE,{DEEMphasis | TRANsition | BOTH}}
  | {FC,{FC1063 | FC2125 | FC425}}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH}[,<multiplier>]}
  | {FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
```

The :MEASure:CLOCk:METHod command sets the clock recovery method to:

- PCIE (PCI Express).
- FC (Fibre Channel).
- EXPlicit (Explicit Clock).
- FIXed (Constant Frequency).
- FLEXR (FlexRay Receiver).
- FLEXT (FlexRay Transmitter).

For setting phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":MEASure:CLOCk:METHod:OJTF" on page 510.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":MEASure:CLOCk:METHod:JTF" on page 508.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number for the base data rate in Hertz.

<multiplier> An integer used as the multiplication factor.

<baud_rate> A real number used for the baud rate.

Example This example sets the explicit clock recovery method on channel 1, rising edge, with a multiplier of 2.

```
myScope.WriteString ":MEASure:CLOCk:METHod EXPlicit,CHANnel1,RISing,2"
```

Query `:MEASure:CLOCk:METHod?`

The :MEASure:CLOCk:METHod? query returns the state of the clock recovery method.

NOTE You can use the :MEASure:CLOCk:METHod? query when phase-locked loop (PLL) clock recovery methods are set up. The format returned will be that of the :MEASure:CLOCk:METHod:OJTF? query. See ":MEASure:CLOCk:METHod:OJTF" on page 510.

Returned Format

```
[:MEASure:CLOCk:METHod]
    {PCIE,{DEEMphasis | TRANsition | BOTH}}
  | {FC,{FC1063 | FC2125 | FC425}}
  | {EXPlicit,<source>,{RISing | FALLing | BOTH},<multiplier>}
  | {FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
  | {FLEXR,<baud_rate>}
  | {FLEXT,<baud_rate>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:OJTF" on page 510
- ":MEASure:CLOCk:METHod:JTF" on page 508

:MEASure:CLOCk:METHod:ALIGn

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCK:METHod:ALIGn {CENTer | EDGE}
```

When using an explicit method of clock recovery, the :MEASure:CLOCk:METHod:ALIGn command specifies how the clock is aligned with data:

- CENTer — Clock edges are aligned with the center of data.
- EDGE — Clock edges are aligned with data edges. In this case, Time Interval Error (TIE) is measured directly from the data edge to the clock edge.

Example When using an explicit method of clock recovery, this example specifies that clock edges are aligned with the center of data.

```
myScope.WriteString ":MEASURE:CLOCK:METHOD:ALIGn CENTer"
```

Query

```
:MEASure:CLOCK:METHod:ALIGn?
```

The :MEASure:CLOCk:METHod:ALIGn? query returns the clock recovery method's edge alignment setting.

Returned Format

```
[:MEASure:CLOCk:METHod:ALIGn] {CENT | EDGE}
```

Example This example places the current edge alignment setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:CLOCK:METHOD:ALIGn?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:CLOCk:METHod:DEEMphasis

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod:DEEMphasis {OFF | ON}
```

The :MEASure:CLOCk:METHod:DEEMphasis command turns de-emphasis on or off. See the help system for more information on de-emphasis.

Example This example enables de-emphasis.

```
myScope.WriteString ":MEASURE:CLOCk:METHod:DEEMphasis ON"
```

Query

```
:MEASure:CLOCk:METHod:DEEMphasis?
```

The :MEASure:CLOCk:METHod:DEEMphasis? query returns whether or not de-emphasis is turned on.

Returned Format

```
[:MEASure:CLOCk:METHod:DEEMphasis] {OFF | ON}
```

Example This example places the current setting of the de-emphasis mode in the string variable strDeemph, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:CLOCK:METHod:DEEMphasis?"
strDeemph = myScope.ReadString
Debug.Print strDeemph
```

:MEASure:CLOCk:METHod:EDGE

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod:EDGE {RISing | FALLing | BOTH}
```

The :MEASure:CLOCk:METHod:EDGE command specifies which edge(s) of the data are used to recover a clock. (In the front panel GUI, this control appears in the Advanced Clock Recovery dialog box.) Normally, both edges are used. However, if you are performing clock recovery on a low duty cycle clock signal, for example, you may want to use just the rising or falling edge.

This command applies to the following clock recovery methods:

- FIXed (Constant Frequency).
- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EXPlicit (Explicit Clock).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).

To measure jitter on only rising (or falling) edges of a clock, you must also set :MEASure:RJDJ:EDGE to the same RISing or FALLing option, and you must set :MEASure:RJDJ:CLOCk ON to force the pattern to be a clock and set the jitter for edges not examined to zero (0).

Example This example specifies that both rising and falling edges of the data are used to recover a clock.

```
myScope.WriteString ":MEASure:CLOCk:METHod:EDGE BOTH"
```

Query

```
:MEASure:CLOCK:METHod:EDGE?
```

The :MEASure:CLOCk:METHod:EDGE? query returns the clock recovery method's edge setting.

Returned Format

```
[:MEASure:CLOCk:METHod:EDGE] {RIS | FALL | BOTH}
```

Example This example places the current edge setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:EDGE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod" on page 502
- ":MEASure:CLOCk:METHod:OJTF" on page 510
- ":MEASure:CLOCk:METHod:JTF" on page 508
- ":MEASure:CLOCk:METHod:ALIGn" on page 504
- ":MEASure:RJDJ:EDGE" on page 631
- ":MEASure:RJDJ:CLOCk" on page 630

:MEASure:CLOCk:METHod:JTF

Command

NOTE

:MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod:JTF
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>, <peaking>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
```

The :MEASure:CLOCk:METHod:JTF command specifies the clock recovery PLL's response in terms of the Jitter Transfer Function's (JTF) 3 dB bandwidth.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).

The EQUalized clock recovery methods are only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Observed Jitter Transfer Function (OJTF), see ":MEASure:CLOCk:METHod:OJTF" on page 510.

For setting other clock recovery methods, see ":MEASure:CLOCk:METHod" on page 502.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number for the base data rate in bits per second.

<peaking> The peaking value in dB.

<jtf_loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a peaking value of 1.25 dB.

```
myScope.WriteString ":MEASure:CLOCk:METHod:JTF SOPLL,4E9,3.822E6,1.25"
```

Query

```
:MEASure:CLOCk:METHod:JTF?
```

The :MEASure:CLOCk:METHod:JTF? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod:JTF]
    {FOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<jtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
  | {EQSOPLL,<data_rate>,<jtf_loop_bandwidth>,<peaking>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<jtf_loop_bandwidth>,<peaking>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:JTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:OJTF" on page 510
- ":MEASure:CLOCk:METHod" on page 502

:MEASure:CLOCk:METHod:OJTF

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod:OJTF
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>, <damping_factor>}
  | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_factor>}
```

The :MEASure:CLOCk:METHod:OJTF command specifies the clock recovery PLL's response in terms of the Observed Jitter Transfer Function's (OJTF) 3 dB bandwidth.

You can set these types of PLL clock recovery methods:

- FOPLL (First Order PLL).
- SOPLL (Second Order PLL).
- EQFOPLL (Equalized First Order PLL).
- EQSOPLL (Equalized Second Order PLL).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).

The EQUalized clock recovery methods are only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled.

For setting phase-locked loop (PLL) clock recovery methods in terms of the Jitter Transfer Function (JTF), see ":MEASure:CLOCk:METHod:JTF" on page 508.

For setting other clock recovery methods, see ":MEASure:CLOCk:METHod" on page 502.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number for the base data rate in bits per second.

<damping_factor> A real number for the damping factor of the PLL.

<ojtf_loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

Example This example sets the clock recovery method to Second Order PLL, a nominal data rate of 4 Gb/s, and a damping factor of 1.0.

```
myScope.WriteString ":MEASure:CLOCk:METHod:OJTF SOPLL,4E9,2.4E6,1.0"
```

Query

```
:MEASure:CLOCk:METHod:OJTF?
```

The :MEASure:CLOCk:METHod:OJTF? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod:OJTF]
    {FOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {EQFOPLL,<data_rate>,<ojtf_loop_bandwidth>}
  | {SOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EQSOPLL,<data_rate>,<ojtf_loop_bandwidth>,<damping_factor>}
  | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>}
  | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<ojtf_loop_bandwidth>,<damping_fact>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:CLOCk:METHod:OJTF?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:JTF" on page 508
- ":MEASure:CLOCk:METHod" on page 502

:MEASure:CLOCk:VERTical

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:VERTical {AUTO | MANual}
```

The :MEASure:CLOCk:VERTical command sets the recovered clock vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the recovered clock vertical scale mode to automatic.

```
myScope.WriteString ":MEASURE:CLOCk:VERTical AUTO"
```

Query

```
:MEASure:CLOCk:VERTical?
```

The :MEASure:CLOCk:VERTical? query returns the current recovered clock vertical scale mode setting.

Returned Format

```
[:MEASure:CLOCk:VERTical] {AUTO | MANual}
```

Example This example places the current setting of the recovered clock vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:CLOCK:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:CLOCk:VERTical:OFFSet

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:VERTical:OFFSet <offset>
```

The :MEASure:CLOCk:VERTical:OFFSet command sets the recovered clock vertical offset.

<offset> A real number for the recovered clock vertical offset.

Example This example sets the clock recovery vertical offset to 1 volt.

```
myScope.WriteString ":MEASURE:CLOCK:VERTICAL:OFFSET 1"
```

Query

```
:MEASure:CLOCk:VERTical:OFFSet?
```

The :MEASure:CLOCk:VERTical:OFFSet? query returns the clock recovery vertical offset setting.

Returned Format

```
[:MEASure:CLOCk:VERTical:OFFSet] <value><NL>
```

<value> The clock recovery vertical offset setting.

Example This example places the current value of recovered clock vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CLOCK:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CLOCk:VERTical:RANGe

Command

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:VERTical:RANGe <range>
```

The :MEASure:CLOCk:VERTical:RANGe command sets the recovered clock vertical range.

<range> A real number for the full-scale recovered clock vertical range.

Example This example sets the recovered clock vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":MEASURE:CLOCK:VERTICAL:RANGE 16"
```

Query

```
:MEASure:CLOCk:VERTical:RANGe?
```

The :MEASure:CLOCk:VERTical:RANGe? query returns the recovered clock vertical range setting.

Returned Format

```
[:MEASure:CLOCk:VERTical:RANGe] <value><NL>
```

<value> The recovered clock vertical range setting.

Example This example places the current value of recovered clock vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CLOCK:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CROSsing

Command `:MEASure:CROSsing <source1>, <source2>`

The :MEASure:CROSsing command adds the crossing measurement to the screen. The crossing measurement is the voltage where two signals cross (uses edges closest to the center of the screen)

<source1> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<source2> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<hysteresis> a real number

Example This example measures the voltage where channel 1 and 2 cross.

```
myScope.WriteString ":MEASure:CROSsing CHANnel1, CHANnel2"
```

Query `:MEASure:CROSsing? [<source1>, <source2>]`

The :MEASure:CROSsing? query returns the crossing measurement value.

If the <source> parameters are not specified, the two sources specified by the :MEASure:SOURce command are used.

Returned Format `[:MEASure:CROSsing] <value><NL>`

<value> The voltage value where the signals cross.

Example This example places the crossing voltage value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:CROSsing? CHANnel1, CHANnel2"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also

- ":MEASure:SOURce" on page 648

:MEASure:CTCDutycycle

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:CTCDutycycle <source>,<direction>
```

The :MEASure:CYCDutycycle command measures the cycle-to-cycle duty cycle jitter (%) of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example This example measures the cycle-to-cycle duty cycle on the rising edge of channel 1.

```
myScope.WriteString ":MEASURE:CTCDUTYCYCLE CHANNEL1,RISING"
```

Query

```
:MEASure:CTCDutycycle? <source>,<direction>
```

The :MEASure:CTCDutycycle? query returns the cycle-to-cycle duty cycle jitter (%) measurement.

Returned Format

```
[:MEASure:CTCDutycycle <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle duty cycle jitter (%) of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle duty cycle of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CTCDUTYCYCLE CHANNEL1,RISING"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CTCJitter

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:CTCJitter <source>,<direction>
```

The :MEASure:CYCJitter command measures the cycle-to-cycle jitter of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> {RISing | FALLing}

Specifies direction of waveform edge to make measurement.

Example This example measures the cycle-to-cycle jitter on the rising edge of channel 1.

```
myScope.WriteString ":MEASURE:CTCJITTER CHANNEL1,RISING"
```

Query

```
:MEASure:CTCJitter? <source>,<direction>
```

The :MEASure:CTCJitter? query returns the cycle-to-cycle jitter time measurement.

Returned Format

```
[:MEASure:CTCJitter <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle jitter time of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle jitter of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CTCJITTER CHANNEL1,RISING"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CTCNwidth

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:CTCNwidth [<source>]
```

The :MEASure:CTCNwidth command measures the cycle-to-cycle -width jitter of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the cycle-to-cycle -width of channel 1.

```
myScope.WriteString ":MEASURE:CTCNWIDTH CHANNEL1"
```

Query

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

The :MEASure:CTCNwidth? query returns the cycle-to-cycle -width jitter measurement.

Returned Format

```
[:MEASure:CTCNwidth <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle - width jitter of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle - width of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CTCNWIDTH CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:CTCPwidth

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:CTCPwidth [<source>]
```

The :MEASure:CYCPwidth command measures the cycle-to-cycle + width jitter of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the cycle-to-cycle - width of channel 1.

```
myScope.WriteString ":MEASURE:CTCPWIDTH CHANNEL1"
```

Query

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

The :MEASure:CTCPwidth? query returns the cycle-to-cycle + width jitter measurement.

Returned Format

```
[:MEASure:CTCPwidth <value>[,<result_state>]<NL>
```

<value> The cycle-to-cycle + width jitter of the waveform.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example places the cycle-to-cycle + width of channel 1 in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:CTCPWIDTH CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:DATarate

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:DATarate <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:DATarate command measures the data rate in bits per second for the selected source. Use the :MEASure:UNITinterval command/query to measure the unit interval of the source

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number specifying the data rate.

Example This example measures the data rate of channel 1.

```
myScope.WriteString ":MEASURE:DATARATE CHANNEL1"
```

Query

```
:MEASure:DATarate? <source>[,{Auto | (SEMI,<data_rate>)}]
```

The :MEASure:DATarate? query returns the measured data rate.

Returned Format

```
[:MEASure:DATarate] <value>[,<result_state>]<NL>
```

<value> Data rate frequency in bits per second for the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current data rate of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:DATARATE? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:DEEMphasis

Command `:MEASure:DEEMphasis [<source>]`

When the EZJIT Complete application is licensed, the Deemphasis serial data measurement becomes available.

The :MEASure:DEEMphasis command adds the deemphasis measurement.

The de-emphasis measurement relies on the clock recovery to recover a clock for each bit in the data waveform. You need to configure clock recovery appropriately for your signal.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DEEMphasis command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example adds the deemphasis measurement on the channel 1 source.

```
myScope.WriteString ":MEASure:DEEMphasis CHANnel1"
```

Query `:MEASure:DEEMphasis? [<source>]`

The :MEASure:DEEMphasis? query returns the measured deemphasis value of the specified source.

Due to random noise, many bits need to be averaged together to average out the noise. Therefore, the current value has little importance and the mean should be used. See ":MEASure:STATistics" on page 649.

Returned Format `[:MEASure:DEEMphasis] <value>[,<result_state>]<NL>`

<value> For every de-emphasis bit in the waveform, a value is computed using:

$$20 * \log_{10}(\text{de-emphasis voltage} / \text{transition voltage})$$

Where:

- Transition voltage is the voltage at the clock location of the preceding transition bit.
- De-emphasis voltage is the voltage at the clock location of de-emphasis bits following a transition bit.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value for deemphasis in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:DEEMphasis? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:DELTatime

Command `:MEASure:DELTatime [<source>[,<source>]]`

The :MEASure:DELTatime command measures the delta time between two edges. If one source is specified, the delta time from the leading edge of the specified source to the trailing edge of the specified source is measured. If two sources are specified, the delta time from the leading edge on the first source to the trailing edge on the second source is measured.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:DELTatime command. The rest of the parameters for this command are specified with the :MEASure:DEFine command.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example measures the delta time between channel 1 and channel 2.

```
myScope.WriteString ":MEASure:DELTatime CHANnel1,CHANnel2"
```

Query `:MEASure:DELTatime? [<source>[,<source>]]`

The :MEASure:DELTatime? query returns the measured delta time value.

Returned Format `[:MEASure:DELTatime] <value>[,<result_state>]<NL>`

<value> Delta time from the first specified edge on one source to the next specified edge on another source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of delta time in the numeric variable, varValue, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:DELTatime?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:DELTatime:DEFine

Command

```
:MEASure:DELTatime:DEFine <start_edge_direction>,<start_edge_number>,
    <start_edge_position>,<stop_edge_direction>,<stop_edge_number>,
    <stop_edge_position>
```

The :MEASure:DELTatime:DEFine command sets the type of direction, the number of the edge, and the edge position for the delta time measurement.

<start_edge_direction> {RISing | FALLing | EITHer} for start directions.

<start_edge_number> An integer from 1 to 65534 for start edge numbers.

<start_edge_position> {UPPer | MIDDle | LOWer} for start edge positions.

<stop_edge_direction> {RISing | FALLing | EITHer} for stop directions.

<stop_edge_number> An integer from 1 to 65534 for stop edge numbers.

<stop_edge_position> {UPPer | MIDDle | LOWer} for stop edge positions.

Example This example sets the delta time starting edge to a rising edge on the 5th edge at the middle position and the stopping edge to falling on the 50th edge at the lower position.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString _
    ":MEASURE:DELTATIME:DEFINE RISING,5,MIDDLE,FALLING,50,LOWER"
```

Query

```
:MEASure:DELTatime:DEFine?
```

The :MEASure:DELTatime:DEFine? query returns the measured delta time value.

Returned Format

```
[:MEASure:DELTatime:DEFine] <start_edge_direction>,<start_edge_number>,
    <start_edge_position>,<stop_edge_direction>,<stop_edge_number>,
    <stop_edge_position><NL>
```

Example This example places the current value of delta time definition in the string variable, strValue, then prints the contents of the variable to the computer's screen. This example assumes the source was set using :MEASure:SOURce.

```
Dim strValue As String    ' Dimension variable.
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:DELTATIME:DEFINE?"
strValue = myScope.ReadString
Debug.Print strValue
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:DUTYcycle

Command `:MEASure:DUTYcycle [<source>[,<direction>]]`

The :MEASure:DUTYcycle command measures the ratio (%) of the positive pulse width to the period.

Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter following the :MEASure:DUTYcycle command.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<direction> {RISing | FALLing}

Specifies direction of edge to start measurement. When <direction> is specified, the <source> parameter is required.

Using the <direction> parameter will set the "Measure All Edges" mode if it is not currently set. You can use the :MEASure:JITTer:STATistics command to turn the "Measure All Edges" mode off (or on again). See ":MEASure:JITTer:STATistics" on page 576.

Example This example measures the duty cycle of the channel 1 waveform.

```
myScope.WriteString ":MEASure:DUTYcycle CHANnel1"
```

Query `:MEASure:DUTYcycle? [<source>],<direction>`

The :MEASure:DUTYcycle? query returns the measured duty cycle (%) of the specified source.

Returned Format `[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>`

<value> The ratio (%) of the positive pulse width to the period.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current duty cycle of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:DUTYcycle? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:EDGE

Command `:MEASure:EDGE [<source>[,<direction>]]`

The :MEASure:EDGE command measures the time of the edge, relative to the reference location.

Sources are specified with the :MEASure:SOURce command or with the optional <source> parameter following the :MEASure:DUTYcycle command.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<direction> {RISing | FALLing | BOTH}

Specifies direction of edge to start measurement. When <direction> is specified, the <source> parameter is required.

Using the <direction> parameter will set the "Measure All Edges" mode if it is not currently set. You can use the :MEASure:JITTer:STATistics command to turn the "Measure All Edges" mode off (or on again). See ":MEASure:JITTer:STATistics" on page 576.

Example This example measures the duty cycle of the channel 1 waveform.

```
myScope.WriteString ":MEASure:EDGE CHANnel1"
```

Query `:MEASure:EDGE? [<source>[,<direction>]]`

The :MEASure:EDGE? query returns the measured edge time of the specified source.

Returned Format `[:MEASure:DUTYcycle] <value>[,<result_state>]<NL>`

<value> The measured edge time.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current duty cycle of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:EDGE? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:ETOedge

Command `:MEASure:ETOedge <source>, <direction 1>, <position 1>, <next/prev>, <relative edge number>, <source 2>, <direction 2>, <position 2>`

The :MEASure:ETOedge command measures the delta time between two edges. It is similar to the delta time measurement, but can be applied to the measurement trend. It also enables you to set whether the measurement is between an edge before or after a specific edge and the number of edges to move forward or backwards.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> May be RISing, FALLing, or BOTH

<position> May be UPPer, MIDDle, or LOWer

<next/prev> May be NEXT or PREVious

:MEASure:FALLtime

Command `:MEASure:FALLtime [<source>]`

The :MEASure:FALLtime command measures the time at the upper threshold of the falling edge, measures the time at the lower threshold of the falling edge, then calculates the fall time. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FALLtime command.

The first displayed falling edge is used for the fall-time measurement. To make this measurement requires 4 or more sample points on the falling edge of the waveform.

Fall time = time at lower threshold point - time at upper threshold point.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the fall time of the channel 1 waveform.

```
myScope.WriteString ":MEASURE:FALLTIME CHANNEL1"
```

Query `:MEASure:FALLtime? [<source>]`

The :MEASure:FALLtime? query returns the fall time of the specified source.

Returned Format `[:MEASure:FALLtime] <value>[,<result_state>]<NL>`

<value> Time at lower threshold - time at upper threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value for fall time in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:FALLTIME? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:FFT:DFRequency

Command `:MEASure:FFT:DFRequency [<source>]`

The :MEASure:FFT:DFRequency command enables the delta frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT:DFR command.

The source must be a function that is set to FFTMagnitude, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:DFRequency? [<source>]`

The :MEASure:FFT:DFRequency? query returns the FFT delta frequency of the specified peaks.

Returned Format `[:MEASure:FFT:DFRequency] <delta_frequency>[,<result_state>]<NL>`

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Related Commands :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

Example This example measures the frequency difference between the peaks specified by the :meas:fft:peak1 and :meas:fft:peak2 for channel 4.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":func4:fftm chan4"     ' Perform FFT on channel 4.
myScope.WriteString ":func4:disp on"        ' Display the FFT.
myScope.WriteString ":meas:FFT:thr-47"   ' Set peak threshold at-47 dBm.
myScope.WriteString ":meas:FFT:Peak1 2"     ' Meas between peak 2 and 3.
myScope.WriteString ":meas:FFT:Peak2 3"
```

```
myScope.WriteString ":meas:FFT:dfr func4"  ' Perform dfrequency meas.
myScope.WriteString ":meas:FFT:dfr? func4" ' Query for measurement.
varFrequency = myScope.ReadNumber
Debug.Print FormatNumber(varFrequency, "Scientific")
```

:MEASure:FFT:DMAGnitude

Command `:MEASure:FFT:DMAGnitude [<source>]`

The :MEASure:FFT:DMAGnitude command enables the delta magnitude measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:DMAGnitude? [<source>]`

The :MEASure:FFT:DMAGnitude? query returns the delta magnitude of the specified peaks.

Returned Format `[:MEASure:FFT:DMAGnitude] <delta_magnitude>[,<result_state>]<NL>`

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Related Commands :MEASure:FFT:PEAK1, :MEASure:FFT:PEAK2, :MEASure:FFT:THReshold

Example This example measures the magnitude difference between the peaks specified by the :meas:fft:peak1 and :meas:fft:peak2 for channel 4.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":func4:fftm chan4"     ' Perform FFT on channel 4.
myScope.WriteString ":func4:disp on"        ' Display the FFT.
myScope.WriteString ":meas:FFT:thr-47" ' Set peak threshold at-47 dBm.
myScope.WriteString ":meas:FFT:Peak1 2"     ' Meas between peak 2 and 3.
myScope.WriteString ":meas:FFT:Peak2 3"
```

```
myScope.WriteString ":meas:FFT:dmag func4"  ' Perform magnitude meas.
myScope.WriteString ":meas:FFT:dmag? func4" ' Query for measurement.
varMagnitude = myScope.ReadNumber
Debug.Print FormatNumber(varMagnitude, "Scientific")
```

:MEASure:FFT:FREQuency

Command `:MEASure:FFT:FREQuency [<source>]`

The :MEASure:FFT:FREQuency command enables the frequency measurement. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:FREQuency? [<source>]`

The :MEASure:FFT:FREQuency? query returns the frequency measurement.

Returned Format `[:MEASure:FFT:FREQuency] <frequency>[,<result_state>]<NL>`

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example measures the frequency the peak specified by the :meas:fft:peak1 for channel 4.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":func4:fftm chan4"     ' Perform FFT on channel 4.
myScope.WriteString ":func4:disp on"        ' Display the FFT.
myScope.WriteString ":meas:FFT:thr-47"   ' Set peak threshold at-47 dBm.
myScope.WriteString ":meas:FFT:Peak1 2"     ' Meas amplitude of peak 2.
myScope.WriteString ":meas:FFT:freq func4"   ' Perform frequency meas.
myScope.WriteString ":meas:FFT:freq? func4" ' Query for measurement.
varFrequency = myScope.ReadNumber
Debug.Print FormatNumber(varFrequency, "Scientific")
```

:MEASure:FFT:MAGNitude

Command `:MEASure:FFT:MAGNitude [<source>]`

The :MEASure:FFT:MAGNitude command measures the magnitude of the FFT. The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FFT command.

The source must be a function that is set to FFT, or a waveform memory that contains an FFT for this command and query to work.

<source> {FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:MAGNitude?`

The :MEASure:FFT:MAGNitude? query returns the magnitude value of the FFT.

Returned Format `[:MEASure:FFT:FMAGNitude] <magnitude>[,<result_state>]<NL>`

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example measures the magnitude of the peak specified by the :meas:fft:peak for channel 4.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":func4:fftm chan4"     ' Perform FFT on channel 4.
myScope.WriteString ":func4:disp on"        ' Display the FFT.
myScope.WriteString ":meas:FFT:thr-47" ' Set peak threshold at-47 dBm.
myScope.WriteString ":meas:FFT:Peak1 2"     ' Meas magnitude of peak 2.
myScope.WriteString ":meas:FFT:magn func4"  ' Perform magnitude meas.
myScope.WriteString ":meas:FFT:magn? func4" ' Query for measurement.
varMagnitude = myScope.ReadNumber
Debug.Print FormatNumber(varMagnitude, "Scientific")
```

:MEASure:FFT:PEAK1

Command `:MEASure:FFT:PEAK1 <1st_peak_number>`

The :MEASure:FFT:PEAK1command sets the peak number of the first peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<F> or WMEMory<N>.

<1st_peak_number> An integer, 1 to 100 specifying the number of the first peak.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:PEAK1?`

The :MEASure:FFT:PEAK1? query returns the peak number currently set as the first peak.

Returned Format `[:MEASure:FFT:PEAK1] <1st_peak_number><NL>`

See Also :MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

:MEASure:FFT:PEAK2

Command `:MEASure:FFT:PEAK2 <2nd_peak_number>`

The :MEASure:FFT:PEAK2 command sets the peak number of the second peak for FFT measurements. The source is specified with the :MEASure:SOURce command as FUNCtion<F> or WMEMory<N>.

<2nd_peak_number> An integer, 1 to 100 specifying the number of the second peak.

<N> WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:FFT:PEAK2?`

The :MEASure:FFT:PEAK2? query returns the peak number currently set as the second peak.

Returned Format `[:MEASure:FFT:PEAK1] <2nd_peak_number><NL>`

See Also :MEASure:FFT:THReshold

Also see the example for :MEASure:FFT:DFRequency in this chapter.

:MEASure:FFT:THReshold

Command `:MEASure:FFT:THReshold <threshold_value>`

The :MEASure:FFT:THReshold command sets the peak search threshold value in dB. The dB after the threshold value is optional.

<threshold_value> A real number specifying the threshold for peaks.

Query `:MEASure:FFT:THReshold?`

The :MEASure:FFT:THReshold? query returns the peak search threshold value.

Returned Format `[:MEASure:FFT:THReshold] <threshold_value><NL>`

These :MEASure commands also operate on FFT functions:

Measure Command	Measurement Performed
:TMAX	The frequency of the maximum value in the spectrum.
:TMIN	The frequency of the minimum value in the spectrum.
:VMAX	The maximum value in the spectrum.
:VMIN	The minimum value in the spectrum.
:VPP	The range of values in the spectrum.
:VTIM	The value at a specified frequency.

See Also Also see the example for :MEASure:FFT:DFRequency in this chapter.

:MEASure:FREQuency

Command `:MEASure:FREQuency [<source>[,<direction>]]`

The :MEASure:FREQuency command measures the frequency of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels if standard thresholds are selected).

The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:FREQuency command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   frequency = 1/(second rising edge time - first rising edge time)
else
   frequency = 1/(second falling edge time - first falling edge time)
```

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<direction> {RISing | FALLing}

Specifies direction of edge for measurement. When <direction> is specified, the <source> parameter is required.

Using the <direction> parameter will set the "Measure All Edges" mode if it is not currently set. You can use the :MEASure:JITTer:STATistics command to turn the "Measure All Edges" mode off (or on again). See ":MEASure:JITTer:STATistics" on page 576.

Example This example measures the frequency of the channel 1 waveform.

```
myScope.WriteString ":MEASure:FREQuency CHANnel1"
```

Query `:MEASure:FREQuency? [<source>[,<direction>]]`

The :MEASure:FREQuency? query returns the measured frequency.

Returned Format `[:MEASure:FREQuency] <value>[,<result_state>]<NL>`

<value> The frequency value in Hertz of the first complete cycle on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current frequency of the waveform in the numeric variable, varFreq, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:FREQuency? CHANnel1"
varFreq = myScope.ReadNumber
Debug.Print FormatNumber(varFreq, 0)
```

:MEASure:HISTogram:HITS

Command `:MEASure:HISTogram:HITS [<source>]`

The :MEASure:HISTogram:HITS command places the histogram hits measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the number of hits within the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:HITS WMEMory1"
```

Query `:MEASure:HISTogram:HITS? [<source>]`

The :MEASure:HISTogram:HITS? query returns the number of hits within the histogram.

Returned Format `[:MEASure:HISTogram:HITS]<value>[,<result_state>]<NL>`

<value> The number of hits in the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits within the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:HITS? WMEMory1"
varHisthits = myScope.ReadNumber
Debug.Print FormatNumber(varHisthits, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:M1S

Command `:MEASure:HISTogram:M1S [<source>]`

The :MEASure:HISTogram:M1S command places the histogram percentage of points within one standard deviation of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example measures the percentage of points that are within one standard deviation of the mean of the histogram of the data stored in waveform memory 3.

```
myScope.WriteString ":MEASure:HISTogram:M1S WMEMory3"
```

Query `:MEASure:HISTogram:M1S? [<source>]`

The :MEASure:HISTogram:M1S? query returns the measurement of the percentage of points within one standard deviation of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M1S]<value>[,<result_state>]<NL>`

<value> The percentage of points within one standard deviation of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within one standard deviation of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M1S? WMEMory1"
varHistm1s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm1s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:M2S

Command `:MEASure:HISTogram:M2S [<source>]`

The :MEASure:HISTogram:M2S command places the histogram percentage of points within two standard deviations of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example measures the percentage of points that are within two standard deviations of the mean of the histogram whose source is specified using the MEASure:SOURce command.

```
myScope.WriteString ":MEASure:HISTogram:M2S WMEMory1"
```

Query `:MEASure:HISTogram:M2S? [<source>]`

The :MEASure:HISTogram:M2S? query returns the measurement of the percentage of points within two standard deviations of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M2S]<value>[,<result_state>]<NL>`

<value> The percentage of points within two standard deviations of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within two standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M2S? WMEMory1"
varHistm2s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm2s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:M3S

Command `:MEASure:HISTogram:M3S [<source>]`

The :MEASure:HISTogram:M2S command places the histogram percentage of points within two standard deviations of the mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example measures the percentage of points that are within three standard deviations of the mean of the histogram.

```
myScope.WriteString ":MEASure:HISTogram:M3S HISTogram"
```

Query `:MEASure:HISTogram:M3S? [<source>]`

The :MEASure:HISTogram:M3S? query returns the measurement of the percentage of points within three standard deviations of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:M3S]<value>[,<result_state>]<NL>`

<value> The percentage of points within three standard deviations of the mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the percentage of points within three standard deviations of the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:M3S? WMEMory1"
varHistm3s = myScope.ReadNumber
Debug.Print FormatNumber(varHistm3s, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:MAX

Command `:MEASure:HISTogram:MAX [<source>]`

The :MEASure:HISTogram:MAX command places the histogram maximum value measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the maximum value of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MAX WMEMory1"
```

Query `:MEASure:HISTogram:MAX? [<source>]`

The :MEASure:HISTogram:MAX? query returns the measurement of the maximum value of the histogram.

Returned Format `[:MEASure:HISTogram:MAX]<value>[,<result_state>]<NL>`

<value> The maximum value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the maximum value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MAX?"
varHistmax = myScope.ReadNumber
Debug.Print FormatNumber(varHistmax, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:MEAN

Command `:MEASure:HISTogram:MEAN [<source>]`

The :MEASure:HISTogram:MEAN command places the histogram mean measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the mean of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MEAN WMEMory1"
```

Query `:MEASure:HISTogram:MEAN? [<source>]`

The :MEASure:HISTogram:MEAN? query returns the measurement of the mean of the histogram.

Returned Format `[:MEASure:HISTogram:MEAN]<value>[,<result_state>]<NL>`

<value> The mean of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the mean of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MEAN? WMEMory1"
varHistmean = myScope.ReadNumber
Debug.Print FormatNumber(varHistmean, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:MEDian

Command `:MEASure:HISTogram:MEDian [<source>]`

The :MEASure:HISTogram:MEDian command places the histogram median measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the median of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MEDian WMEMory1"
```

Query `:MEASure:HISTogram:MEDian? [<source>]`

The :MEASure:HISTogram:MEDian? query returns the measurement of the median of the histogram.

Returned Format `[:MEASure:HISTogram:MEDian]<value>[,<result_state>]<NL>`

<value> The median of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the median of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MEDian? WMEMory1"
varHistmed = myScope.ReadNumber
Debug.Print FormatNumber(varHistmed, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:MIN

Command `:MEASure:HISTogram:MIN [<source>]`

The :MEASure:HISTogram:MIN command places the histogram minimum measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the minimum the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MIN WMEMory1"
```

Query `:MEASure:HISTogram:MIN? [<source>]`

The :MEASure:HISTogram:MIN? query returns the measurement of the minimum value of the histogram.

Returned Format `[:MEASure:HISTogram:MIN]<value>[,<result_state>]<NL>`

<value> The minimum value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the minimum value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MIN?"
varHistmin = myScope.ReadNumber
Debug.Print FormatNumber(varHistmin, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:MODE

Command `:MEASure:HISTogram:MODE [<source>]`

The :MEASure:HISTogram:MODE command places the histogram mode measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the mode of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:MODE WMEMory1"
```

Query `:MEASure:HISTogram:MODE? [<source>]`

The :MEASure:HISTogram:MODE? query returns the measurement histogram's Mode value.

Returned Format `[:MEASure:HISTogram:MODE]<value>[,<result_state>]<NL>`

<value> The Mode value of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the Mode value of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:MODE? WMEMory1"
varHistMode = myScope.ReadNumber
Debug.Print FormatNumber(varHistMode, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:PEAK

Command `:MEASure:HISTogram:PEAK [<source>]`

The :MEASure:HISTogram:PEAK command places the histogram number of hits in the greatest peak measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the number of hits in the greatest peak of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:PEAK WMEMory1"
```

Query `:MEASure:HISTogram:PEAK? [<source>]`

The :MEASure:HISTogram:PEAK? query returns the number of hits in the greatest peak of the histogram measurement.

Returned Format `[:MEASure:HISTogram:PEAK]<value>[,<result_state>]<NL>`

<value> The number of hits in the histogram peak.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the number of hits in the greatest peak of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:PEAK? WMEMory1"
varHistpeak = myScope.ReadNumber
Debug.Print FormatNumber(varHistpeak, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:PP

Command `:MEASure:HISTogram:PP [<source>]`

The :MEASure:HISTogram:PP command places the histogram width measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the width of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:PP WMEMory1"
```

Query `:MEASure:HISTogram:PP? [<source>]`

The :MEASure:HISTogram:PP? query returns the measurement of the width of the histogram.

Returned Format `[:MEASure:HISTogram:PP]<value>[,<result_state>]<NL>`

<value> The width of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the width of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:PP? WMEMory1"
varHistpp = myScope.ReadNumber
Debug.Print FormatNumber(varHistpp, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:RESolution

Command `:MEASure:HISTogram:RESolution [<source>]`

The :MEASure:HISTogram:RESolution command places the histogram bin width measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the bin width of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:RESolution WMEMory1"
```

Query `:MEASure:HISTogram:RES? [<source>]`

The :MEASure:HISTogram:RES? query returns the measurement of the bin width of the histogram.

Returned Format `[:MEASure:HISTogram:RES]<value>[,<result_state>]<NL>`

<value> The width of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the width of the current histogram and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:RESolution? WMEMory1"
varHistpp = myScope.ReadNumber
Debug.Print FormatNumber(varHistpp, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HISTogram:STDDev

Command `:MEASure:HISTogram:STDDev [<source>]`

The :MEASure:HISTogram:STDDev command places the histogram standard deviation measurement into the Measurements tab of the oscilloscope's user interface.

The source is specified with the :MEASure:SOURce command or with the optional parameter following the command.

The :MEASure:HISTogram commands only apply to Meas Histogram math functions, the histogram waveform, or memories containing histograms.

<source> { FUNCtion<F> | WMEMory<N> | HISTogram}

<F> An integer, 1-16.

<N> An integer, 1-4.

Example This example places into the Measurements tab the standard deviation of the histogram stored in WMEMory1.

```
myScope.WriteString ":MEASure:HISTogram:STDDev WMEMory1"
```

Query `:MEASure:HISTogram:STDDev? [<source>]`

The :MEASure:HISTogram:STDDev? query returns the measurement of standard deviation of the histogram.

Returned Format `[:MEASure:HISTogram:STDDev]<value>[,<result_state>]<NL>`

<value> The standard deviation of the histogram.

<result_state> If SENDVALID is ON, the result state is returned with the measurement result. Refer to the MEASure:RESults command, for a list of the result states.

Example This example returns the standard deviation of the histogram whose source is specified using the MEASure:SOURce command and prints the result to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:HISTogram:STDDEV? WMEMory1"
varHiststtd = myScope.ReadNumber
Debug.Print FormatNumber(varHiststtd, 0)
```

See Also

- ":FUNCtion<F>:MHIStogram" on page 356
- ":HISTogram:MODE" on page 380

:MEASure:HOLDtime

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:HOLDtime [<data_source>,<data_source_dir>, <clock_source>,<cloc
k_ source_dir>]
```

The :MEASure:HOLDtime command measures the hold time between the specified clock and data sources.

<data_source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<clock_source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_source_dir> {RISing | FALLing | BOTH}

Selects the direction of the data source edge.

<clock_source_dir> {RISing | FALLing}

Selects the direction of the clock source edge.

Example This example measures the hold time from the rising edge of channel 1 to the rising edge of channel 2.

```
myScope.WriteString ":MEASURE:HOLDTIME CHAN1,RIS,CHAN2,RIS"
```

Query `:MEASure:HOLDtime? [<data_source>,<data_source_dir>,<clock_source>,<clock_ source_dir>]`

The :MEASure:HOLDtime? query returns the measured hold time between the specified clock and data source.

Returned Format `{:MEASure:SETuptime] <value><NL>`

<value> Hold time in seconds.

Example This example places the current value of hold time in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:HOLDTIME? CHAN1,RIS,CHAN2,RIS"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

See Also Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

:MEASure:JITTer:HISTogram

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:HISTogram {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:HISTogram command turns the measurement histogram display on or off when a jitter measurement is displayed.

Example This example turns the jitter measurement histogram display on.

```
myScope.WriteString ":MEASURE:JITTER:HISTOGRAM ON"
```

Query

```
:MEASure:JITTer:HISTogram?
```

The :MEASure :JITTer:HISTogram? query returns the state of measurement histogram display.

Returned Format

```
[:MEASure:JITTer:HISTogram] {1 | 0}
```

Example This example places the current setting of the jitter spectrum mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:HISTOGRAM?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:MEASure:JITTer:MEASurement

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:MEASurement {MEASurement<N>}
```

The :MEASure :JITTer:MEASurement command selects which measurement displayed on the oscilloscope you are performing the jitter analysis on. MEASurement1 is the left-most measurement on the display.

<N> {1 | 2 | 3 | 4 | 5}

Example This example assigns measurement 2 to the jitter measurement analysis.

```
myScope.WriteString ":MEASURE:JITTER:MEASUREMENT MEASUREMENT2"
```

Query

```
:MEASure:JITTer:MEASurement?
```

The :MEASure :JITTer:MEASurement? query returns the measurement number you are performing the jitter analysis on. If no measurements are being displayed on the oscilloscope, the query will return a null string.

Returned Format

```
[:MEASure:JITTer:MEASurement MEASurement<N>]
```

Example This example places the current measurement number that you are performing jitter analysis on in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:MEASUREMENT?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:SPECtrum

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:SPECtrum command turns the jitter spectrum display on or off when a jitter measurement is displayed.

Example This example turns the jitter measurement spectrum display on.

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM ON"
```

Query

```
:MEASure:JITTer:SPECtrum?
```

The :MEASure :JITTer:SPECtrum? query returns the state of jitter spectrum display.

Returned Format

```
[:MEASure:JITTer:SPECtrum] {1 | 0}
```

Example This example places the current setting of the jitter spectrum mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:SPECTRUM?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:MEASure:JITTer:SPECtrum:HORizontal

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal {AUTO | MANual}
```

The :MEASure:JITTer:SPECtrum:HORizontal command sets the jitter spectrum horizontal mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the horizontal scaling and center frequency. In manual mode, you can set your own horizontal scaling and center frequency values.

Example This example sets the jitter spectrum horizontal mode to automatic.

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL AUTO"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal?
```

The :MEASure:JITTer:SPECtrum:HORizontal? query returns the current jitter spectrum horizontal mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal] {AUTO | MANual}
```

Example This example places the current setting of the jitter trend horizontal mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:SPECtrum:HORizontal:POSition

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition <position>
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition command sets the jitter spectrum horizontal center frequency position.

<position> A real number for the center frequency position in Hertz.

Example This example sets the jitter spectrum horizontal center frequency position to 250 kHz.

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL:POSITION 250E3"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:POSition?
```

The :MEASure:JITTer:SPECtrum:HORizontal:POSition? query returns the current jitter spectrum horizontal center frequency position setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal:POSition] <value><NL>
```

<value> The jitter spectrum horizontal center frequency setting.

Example This example places the current setting of the jitter trend horizontal center frequency position in the variable varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL:POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:SPECtrum:HORizontal:RANGe

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe command sets the jitter spectrum horizontal range.

<range> A real number for the horizontal frequency range in Hertz.

Example This example sets the jitter spectrum horizontal range to 10 GHz (1 GHz/div).

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL:RANGE 10E9"
```

Query

```
:MEASure:JITTer:SPECtrum:HORizontal:RANGe?
```

The :MEASure:JITTer:SPECtrum:HORizontal:RANGe? query returns the current jitter spectrum horizontal range setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:HORizontal:RANGe] <value><NL>
```

<value> The jitter spectrum horizontal range setting.

Example This example places the current setting of the jitter trend horizontal range in the variable varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:HORIZONTAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:SPECtrum:VERTical

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:VERTical {AUTO | MANual}
```

The :MEASure:JITTer:SPECtrum:VERTical command sets the jitter spectrum vertical mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own vertical scaling and offset values.

Example This example sets the jitter spectrum vertical mode to automatic.

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL AUTO"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical?
```

The :MEASure:JITTer:SPECtrum:VERTical? query returns the current jitter spectrum vertical mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical] {AUTO | MANual}
```

Example This example places the current setting of the jitter spectrum vertical mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:SPECtrum:VERTical:OFFSet

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:SPECtrum:VERTical:OFFSet command sets the jitter spectrum vertical offset.

<offset> A real number for the vertical offset of the jitter measurement spectrum.

Example This example sets the jitter spectrum vertical offset to 2 ns.

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL:OFFSET 10E-9"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:OFFSet?
```

The :MEASure:JITTer:SPECtrum:VERTical:OFFSet? query returns the jitter spectrum vertical offset time.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:OFFSet] <value> [,<result_state>]<NL>
```

<value> The jitter vertical spectrum offset time setting.

Example This example places the current value of jitter spectrum vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:SPECtrum:VERTical:RANGe

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe <range>
```

The :MEASure:JITTer:SPECtrum:VERTical:RANGe command sets the jitter spectrum vertical range.

<range> A real number for the full-scale vertical range for the jitter measurement spectrum.

Example This example sets the jitter spectrum vertical range to 4 ns (500 ps/div X 8 div).

```
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL:RANGE 4E-9"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:RANGe?
```

The :MEASure:JITTer:SPECtrum:VERTical:RANGe? query returns the jitter spectrum range time setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:RANGe] <value> [,<result_state>]<NL>
```

<value> The jitter spectrum vertical range setting.

Example This example places the current value of jitter spectrum vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:JITTER:SPECTRUM:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:SPECtrum:VERTical:TYPE

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:VERTical:TYPE {LINear | LOGarithmic}
```

The :MEASure:JITTer:SPECtrum:VERTical:TYPE command lets you select either a LINear or a LOGarithmic vertical scale for the jitter spectrum plot.

Example This example sets a linear vertical scale for the jitter spectrum plot.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:TYPE LINear"
```

Query

```
:MEASure:JITTer:SPECtrum:VERTical:TYPE?
```

The :MEASure:JITTer:SPECtrum:VERTical:TYPE? query returns the current jitter spectrum plot vertical scale setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:VERTical:TYPE] {LINear | LOGarithmic}
```

Example This example places the current jitter spectrum plot vertical scale setting in the string variable strType, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:VERTical:TYPE?"
strType = myScope.ReadString
Debug.Print strType
```

:MEASure:JITTer:SPECtrum:WINDow

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:SPECtrum:WINDow {RECTangular | HANNing | FLATtop
                                | BHARris | HAMMing}
```

The :MEASure:JITTer:SPECtrum:WINDow command sets the jitter spectrum window mode. For a description of the window modes, see ":FUNCtion<F>:FFT:WINDow" on page 342.

Example This example sets the jitter spectrum window mode to Hanning.

```
myScope.WriteString ":MEASure:JITTer:SPECtrum:WINDow HANNing"
```

Query

```
:MEASure:JITTer:SPECtrum:WINDow?
```

The :MEASure:JITTer:SPECtrum:WINDow? query returns the current jitter spectrum window mode setting.

Returned Format

```
[:MEASure:JITTer:SPECtrum:WINDow] {RECTangular | HANNing | FLATtop
                                  | BHARris | HAMMing}<NL>
```

Example This example places the current setting of the jitter spectrum window mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:SPECtrum:WINDow?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:STATistics

Command `:MEASure:JITTer:STATistics {{ON|1} | {OFF|0}}`

The :MEASure:JITTer:STATistics command enables or disables jitter mode and allows you to view: measurement histogram (:MEASure:JITTer:HISTogram), measurement trend (:MEASure:JITTer:TRENd), and jitter spectrum (:MEASure:JITTer:SPsECtrum) if they are enabled.

The :MEASure:JITTer:STATistics command also turns on or off the ability to measure all edges in the waveform; not just the first edge on screen.

Example This example turns the jitter measurement statistics and the "Measure All Edges" mode on.

```
myScope.WriteString ":MEASure:JITTer:STATistics ON"
```

Query `:MEASure:JITTer:STATistics?`

The :MEASure:JITTer:STATistics? query returns the state of jitter statistics.

Returned Format `[:MEASure:JITTer:STATistics] {1 | 0}`

Example This example places the current setting of the jitter statistics mode in the variable varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASure:JITTer:STATistics?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:MEASure:JITTer:TRENd

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd command turns the jitter measurement trend display on or off. When on, trend plots measurement results time correlated to the waveform being measured.

Example This example turns the jitter measurement trend display on.

```
myScope.WriteString ":MEASURE:JITTER:TREND ON"
```

Query

```
:MEASure:JITTer:TRENd?
```

The :MEASure :JITTer:TRENd? query returns the state of jitter trend display.

Returned Format

```
[:MEASure:JITTer:TRENd] {1 | 0}
```

Example This example places the current setting of the jitter trend mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:TREND?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:TRENd:SMOoth

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd:SMOoth {{ON|1} | {OFF|0}}
```

The :MEASure:JITTer:TRENd:SMOoth command sets jitter trend smoothing to on or off. When on, smoothing creates a running average smoothed by the number of points set by the :JITTer:TRENd:SMOoth:POINts command.

Example This example sets the jitter trend smoothing mode to on.

```
myScope.WriteString ":MEASURE:JITTer:TREND:SMOOTH ON"
```

Query

```
:MEASure:JITTer:TRENd:SMOoth?
```

The :MEASure:JITTer:TRENd:SMOoth? query returns the current jitter trend smoothing mode setting.

Returned Format

```
[:MEASure:JITTer:TRENd:SMOoth] {1 | 0}
```

Example This example places the current setting of the jitter trend smoothing mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:TREND:SMOOTH?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:TRENd:SMOoth:POINts

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd:SMOoth:POINts <points>
```

The :MEASure:JITTer:TRENd:SMOoth:POINts command sets the number of points as a set size for the data smoothing feature.

<points> odd integers, 3 to 1001. If out of range, the number will be rounded to nearest lower odd integer.

Example This example sets the jitter trend smoothing points to 7.

```
myScope.WriteString ":MEASURE:JITTER:TREND:SMOOTH:POINTS 7"
```

Query

```
:MEASure:JITTer:TRENd:SMOoth:POINts?
```

The :MEASure:JITTer:TRENd:SMOoth:POINts? query returns the current setting for jitter trend smoothing points.

Returned Format

```
[:MEASure:JITTer:TRENd:SMOoth:POINts] <value><NL>
```

<value> The jitter offset smoothing points setting.

Example This example places the current value of jitter trend smoothing points in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:JITTER:TREND:SMOOTH:POINTS?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:TRENd:VERTical

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd:VERTical {AUTO | MANual}
```

The :MEASure:JITTer:TRENd:VERTical command sets the jitter trend vertical mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the jitter trend vertical mode to automatic.

```
myScope.WriteString ":MEASURE:JITTer:TRENd:VERTical AUTO"
```

Query

```
:MEASure:JITTer:TRENd:VERTical?
```

The :MEASure:JITTer:TRENd:VERTical? query returns the current jitter trend vertical mode setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical] {AUTO | MANual}
```

Example This example places the current setting of the jitter trend vertical mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:JITTER:TREND:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:JITTer:TRENd:VERTical:OFFSet

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd:VERTical:OFFSet <offset>
```

The :MEASure:JITTer:TRENd:VERTical:OFFSet command sets the jitter trend vertical offset.

<offset> A real number for the vertical offset for the jitter measurement trend.

Example This example sets the jitter trend vertical offset to 100 ps.

```
myScope.WriteString ":MEASURE:JITTER:TREND:VERTICAL:OFFSET 100E-12"
```

Query

```
:MEASure:JITTer:TRENd:VERTical:OFFSet?
```

The :MEASure:JITTer:TRENd:VERTical:OFFSet? query returns the jitter trend vertical offset setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical:OFFSet] <value><NL>
```

<value> The jitter vertical trend offset setting.

Example This example places the current value of jitter trend vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:JITTER:TREND:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:JITTer:TRENd:VERTical:RANGe

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:JITTer:TRENd:VERTical:RANGe <range>
```

The :MEASure:JITTer:TRENd:VERTical:RANGe command sets the jitter trend vertical range.

<range> A real number for the full-scale vertical range for the jitter measurement trend.

Example This example sets the jitter trend vertical range to 4 ns (500 ps/div X 8 div).

```
myScope.WriteString ":MEASURE:JITTER:TREND:VERTICAL:RANGE 4E-9"
```

Query

```
:MEASure:JITTer:TRENd:VERTical:RANGe?
```

The :MEASure:JITTer:TRENd:VERTical:RANGe? query returns the jitter trend vertical range setting.

Returned Format

```
[:MEASure:JITTer:TRENd:VERTical:RANGe] <value><NL>
```

<value> The jitter trend vertical range setting.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of jitter trend vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:JITTER:TREND:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:NAME

Command `:MEASure:NAME {MEAS1 | MEAS2 | MEAS3 | MEAS4}, <name>`

The :MEASure:NAME commands sets the name of the specified measurement to whatever string is given to <name>. This enables you to give specific names to measurements displayed on the oscilloscope's screen.

<name> a quoted string

Query `:MEASure:NAME? {MEAS1 | MEAS2 | MEAS3 | MEAS4}`

The :MEASure:NAME? query returns the name of the corresponding measurement.

:MEASure:NCJitter

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:NCJitter <source>,<direction>,<n>,<start>
```

The :MEASure:NCJitter command measures the N cycle jitter of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> {RISing | FALLing}, specifies direction of waveform edge to make measurement.

<n> An integer, 1 to 99, the number of cycles in a group.

<start> An integer, 1 to <n> - 1, typically 1, the cycle to start measuring.

Example This example measures the N cycle jitter on channel 1, rising edge, 5 cycles in a group, starting on the first cycle of the waveform.

```
myScope.WriteString ":MEASURE:NCJITTER CHANNEL1,RISING,5,1"
```

Query

```
:MEASure:NCJitter? <source>,<direction>,<n>,<start>
```

The :MEASure:NCJitter? query returns the measured N cycle jitter time of the waveform.

Returned Format

```
[:MEASure:NCJitter] <value>[,<result_state>]<NL>
```

<value> The N cycle jitter time of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of N cycle jitter in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NCJITTER? CHANNEL1,RIS,5,1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:NOISe

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe <source>, {VOLT | UNITamp}, {ZERO | ONE | BOTH}
```

The :MEASure:NOISe command adds a Noise measurement to the oscilloscope display.

The parameters specify the input source to be measured, the units (in volts or unit amplitude), and whether "zeros", "ones", or both "zeros" and "ones" should be measured.

This command is the equivalent of adding a noise measurement via **Measure > Data > Noise** in the front panel user interface.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | EQUalized}

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> In CHANnel<N> and WMEMory<N>, N is an integer, 1-4, representing the selected channel or waveform memory.

<F> FUNCtion<F> is an integer, 1-16.

Example This example adds a "ones" Noise measurement on channel 1, in volt units, to the oscilloscope display. The measurement results appear in the Measurements tab.

```
myScope.WriteString ":MEASure:NOISe CHANnel1,VOLT,ONE"
```

Query

```
:MEASure:NOISe? <source>, {VOLT | UNIT}, {ZERO | ONE | BOTH}
```

The :MEASure:NOISe? query returns the measured noise value.

Returned Format

```
[:MEAS:NOIS] <measured_value><NL>
```

<measured_value> The measured "zeros", "ones", or both noise value in volts or unit amplitude.

Example This example places the measurement result in the varMeasuredNoise variable.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe? CHANnel1,VOLT,ONE"
varMeasuredNoise = myScope.ReadNumber
Debug.Print FormatNumber(varMeasuredNoise, 0)
```

:MEASure:NOISe:ALL?

Query

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:ALL? {ZERO | ONE}
```

The :MEASure:NOISe:ALL? query returns the NOISe measurement results for the specified level. These values are returned as comma separated values using the following format:

Returned Format

```
[:MEASure:NOISe:ALL<space>]
TI(<format>),<result>,<state>,
RN(<format>),<result>,<state>,
DI(<format>),<result>,<state>,
PI(<format>),<result>,<state>,
ABUI(<format>),<result>,<state>,
BUI(<format>),<result>,<state>,
ISI(<format>),<result>,<state>,
Count,<number_of_bits>,<state>,
Level,<nominal_level>,<state>,
Eye Height(<format>),<result>,<state>,<NL>
```

NOTE Whether some of these values are included or not depends on the setting of :MEASure:NOISe:METHod and :MEASure:NOISe:REPort.

For example, when :MEASure:NOISe:REPort or :MEASure:NOISe:METHod is SPECtral, the BUI and ABUI values are not returned, and there are two PI values (one "rms" and one "dd").

<space> White space (ASCII 32) character.

<format> The format value tells you something about how the measurement is made. For instance, TI(1E-12) means that the TI measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (dd) means the measurement uses a dual-Dirac delta model to derive the measurement. A format of (pp) means the measurement is a peak-to-peak measurement.

<result> The measured results for the NOISe measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<state> The measurement result state. See Table 24 for a list of values and descriptions of the result state value.

<number_of_bits> The number of waveform bits that have been measured.

<nominal_level> The Level line returns the nominal one or zero level. The unit amplitude = the nominal one level – nominal zero level.

Example This example places the noise measurement result for "ones" in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NOISe:ALL? ONE"
strResults = myScope.ReadString
Debug.Print strResults
```

See Also

- ":MEASure:NOISe:METHod" on page 591
- ":MEASure:NOISe:REPort" on page 592

:MEASure:NOISe:BANDwidth

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:BANDwidth {NARRow | WIDE}
```

The :MEASure:NOISe:BANDwidth command sets the type of filtering used to separate the data dependent noise from the random noise and the periodic noise.

Example This example sets the RN bandwidth to WIDE.

```
myScope.WriteString ":MEASURE:NOISe:BANDWIDTH WIDE"
```

Query

```
:MEASure:NOISe:BANDwidth?
```

The :MEASure:NOISe:BANDwidth? query returns the RN bandwidth filter setting.

Returned Format

```
[:MEASure:NOISe:BANDwidth] {NARRow | WIDE}<NL>
```

Example This example places the RN filter setting the strFilter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NOISe:BANDWIDTH?"
strFilter = myScope.ReadString
Debug.Print strFilter
```

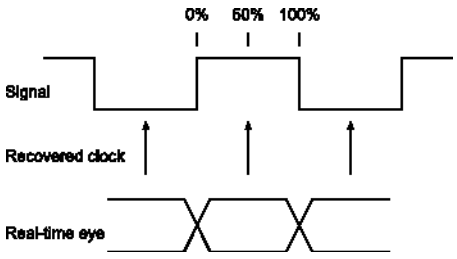
:MEASure:NOISe:LOCation

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:LOCation <location>
```

The :MEASure:NOISe:LOCation command specifies the measurement location within the bit where 0% is the beginning of the bit, 50% is the middle of the bit, and 100% is the end of the bit.



You can specify a location value from 5% to 95%.

Example This example sets the measurement location to 60%.

```
myScope.WriteString ":MEASURE:NOISe:LOCation 60"
```

Query

```
:MEASure:NOISe:LOCation?
```

The :MEASure:NOISe:LOCation? query returns the measurement location setting.

Returned Format

```
[:MEASure:NOISe:LOCation] <location><NL>
```

Example This example places the measurement location setting the varLocation variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe:LOCation?"
varLocation = myScope.ReadNumber
Debug.Print FormatNumber(varLocation, 0)
```

:MEASure:NOISe:METHod

Command

NOTE This command is only available when the EZJIT Complete jitter analysis application is licensed.

```
:MEASure:NOISe:METHod {SPECtral | BOTH}
```

The :MEASure:NOISe:METHod command lets you select the method for random noise (RN) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.

When analyzing noise with crosstalk or ground bounce effects present in your signal, select BOTH. When this option is selected, the deterministic interference (DI) that is uncorrelated to the data pattern, also known as bounded uncorrelated interference (BUI), is separated into periodic interference (PI) and aperiodic bounded uncorrelated interference (ABUi). ABUi is caused by crosstalk and ground bounce effects.

When there are no crosstalk or ground bounce effects present in your signal, you can select the SPECtral method in order to run faster. When this option is selected, the deterministic interference (DI) that is uncorrelated to the data pattern is all reported as periodic interference (PI).

Example This example sets NOISe method to BOTH the spectral and tail fit analysis.

```
myScope.WriteString ":MEASURE:NOISe:METHod BOTH"
```

Query

```
:MEASure:NOISe:METHod?
```

The :MEASure:NOISe:METHod? query returns the selected NOISe method.

Returned Format

```
[:MEASure:NOISe:METHod] {SPEC | BOTH}<NL>
```

Example This example places the NOISe method setting the strNoiseMethod variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NOISe:METHod?"
strNoiseMethod = myScope.ReadString
Debug.Print strNoiseMethod
```

See Also

- ":MEASure:NOISe:REPort" on page 592

:MEASure:NOISe:REPort

Command

NOTE This command is only available when the EZJIT Complete jitter analysis application is licensed.

```
:MEASure:NOISe:REPort {SPECtral | TAILfit}
```

When the :MEASure:NOISe:METHod BOTH command selects both the spectral and tail fit methods for random noise analysis, the :MEASure:NOISe:REPort command specifies which method is used for the reports in the noise graphs / histograms and Noise tab measurements.

Example This example specifies that the NOISe report include measurements from both the spectral and tail fit analysis (including aperiodic bounded uncorrelated interference ABUI measurements).

```
myScope.WriteString ":MEASURE:NOISe:REPort TAILfit"
```

Query

```
:MEASure:NOISe:REPort?
```

The :MEASure:NOISe:REPort? query returns the report setting.

Returned Format

```
[:MEASure:NOISe:REPort] {SPEC | TAIL}<NL>
```

Example This example places the report setting in the strReportSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe:REPort?"
strReportSetting = myScope.ReadString
Debug.Print strReportSetting
```

See Also

- ":MEASure:NOISe:METHod" on page 591

:MEASure:NOISe:RN

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:RN {ON, <RNrms Zero>, <RNrms One> | OFF}
```

The :MEASure:NOISe:RN command can specify a known amount of random noise. When used, the remaining amount of the total noise measured is reported as periodic interference (PI).

This command is used in situations when crosstalk aggressors influence the random noise measured on a signal. If the random noise on a signal is measured without the aggressor signal crosstalk, this known amount of random noise can be specified when measuring the noise again with the crosstalk aggressors.

- ON — Enables a specified amount of random noise.
- <RNrms Zero> — The known amount of "zeros" random noise.
- <RNrms One> — The known amount of "ones" random noise.
- OFF — Disables the specification of random noise amounts.

Specified amounts of "ones" and "zeros" random noise is shown in the noise measurement results (see page 587) as "RN(rms specified)".

Example This example specifies 100 µV of known "zeros" random noise and 200 µV of known "ones" random noise.

```
myScope.WriteString ":MEAS:NOISE:RN ON, 100e-6, 200e-6"
```

Query

```
:MEASure:NOISe:RN?
```

The :MEASure:NOISe:RN? query returns the specified RN settings.

Returned Format

```
[:MEASure:NOISe:RN] {ON, <RNrms Zero>, <RNrms One> | OFF}<NL>
```

Example This example places the specified RN settings in the strKnownRandomNoise variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe:RN?"
strKnownRandomNoise = myScope.ReadString
Debug.Print strKnownRandomNoise
```

:MEASure:NOISe:SCOPe:RN

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:SCOPe:RN {ON, <RNrms Zero>, <RNrms One> | OFF}
```

The :MEASure:NOISe:SCOPe:RN command can specify the removal of the oscilloscope's calibrated random noise from the reported RN.

- ON – Enables the removal of the oscilloscope's calibrated random noise from the reported RN.
- <RNrms Zero> – The oscilloscope's "zeros" random noise to remove from the reported RN.
- <RNrms One> – The oscilloscope's "ones" random noise to remove from the reported RN.
- OFF – Disables the removal of the oscilloscope's calibrated random noise from the reported RN.

Running the **Calibrate scope jitter / noise** from the front panel user interface will set <RNrms Zero> and <RNrms One> to the measured values; however, the measures values can be changed by this command.

Example This example specifies 100 µV of oscilloscope "zeros" random noise and 200 µV of oscilloscope "ones" random noise.

```
myScope.WriteString ":MEAS:NOISE:SCOPE:RN ON, 100e-6, 200e-6"
```

Query

```
:MEASure:NOISe:SCOPe:RN?
```

The :MEASure:NOISe:SCOPe:RN? query returns the oscilloscope RN settings.

Returned Format

```
[:MEASure:NOISe:SCOPe:RN] {ON, <RNrms Zero>, <RNrms One> | OFF}<NL>
```

Example This example places the oscilloscope RN settings in the strScopeRandomNoise variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NOISe:SCOPe:RN?"
strScopeRandomNoise = myScope.ReadString
Debug.Print strScopeRandomNoise
```

:MEASure:NOISe:STATe

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:STATe {ON | OFF}
```

The :MEASure:NOISe:STATe command enables or disables the NOISe measurements.

Example This example sets the NOISe state to on.

```
myScope.WriteString ":MEASURE:NOISe:STATe ON"
```

Query

```
:MEASure:NOISe:STATe?
```

The :MEASure:NOISe:STATe? query returns the state of the NOISe measurements.

Returned Format

```
[:MEASure:NOISe:STATe] {1 | 0}<NL>
```

Example This example places the current state of the NOISe measurements in the varState variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NOISe:STATE?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

:MEASure:NOISe:UNITs

Command

NOTE This command is only available when the EZJIT Complete software is installed.

```
:MEASure:NOISe:UNITs {VOLT | UNITamplitude}
```

The :MEASure:NOISe:UNITs command sets the unit of measure for NOISe measurements to volts or unit amplitude.

Example This example sets the NOISe units to unit amplitude.

```
myScope.WriteString ":MEASURE:NOISe:UNITs UNITamplitude"
```

Query

```
:MEASure:NOISe:UNITs?
```

The :MEASure:NOISe:UNITs? query returns the units of measure being used for the NOISe measurements.

Returned Format

```
[:MEASure:NOISe:UNITs] {VOLT | UNITamplitude}<NL>
```

Example This example places the current units of measure for the NOISe measurements in the strUnits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:NOISe:UNITs?"
strUnits = myScope.ReadString
Debug.Print strUnits
```

:MEASure:NPERiod

Command `:MEASure:NPERiod <source>, <slope>, <N>`

The :MEASure:NPERiod command measures the span of time of N consecutive periods. The measurement then moves over one period and measures the span of time of the next N consecutive periods.

<source> the source on which the measurement is made

<slope> rising or falling

<N> An integer greater than or equal to 1.

Example This example measures the time span of 3 consecutive periods on channel 1 (rising edge).

```
myScope.WriteString ":MEASURE:NPERiod CHAN1, rising, 3"
```

Query `:MEASure:NPERiod?`

:MEASure:NPULses

Command `:MEASure:NPULses <source>`

The :MEASure:NPULses measures the number of negative pulses on the screen.

<source> the source on which the measurement is made

Example This example measures the number of negative pulses on channel 1.

```
myScope.WriteString ":MEASURE:NPULses CHAN1"
```

Query :MEASure:NPULses?

This query returns the result for the NPULses measurement.

:MEASure:NUI

Command `:MEASure:NPERiod <source>, <N>`

The :MEASure:NPERiod command measures N consecutive unit intervals. The measurement then moves over one unit interval and measures the span of time of the next N consecutive unit intervals.

<source> the source on which the measurement is made

<N> An integer greater than or equal to 1.

Example This example measures the time span of 3 consecutive unit intervals on channel 1.

```
myScope.WriteString ":MEASURE:NUI CHAN1, 3"
```

Query `:MEASure:NUI?`

:MEASure:NWIDth

Command `:MEASure:NWIDth [<source>]`

The :MEASure:NWIDth command measures the width of the first negative pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard threshold selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:NWIDth command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   nwidth = time at the second rising edge - time at the first
   falling edge
else
   nwidth = time at the first rising edge - time at the first
   falling edge
```

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example measures the width of the first negative pulse on the screen.

```
myScope.WriteString ":MEASure:NWIDth CHANnel1"
```

Query `:MEASure:NWIDth? [<source>]`

The :MEASure:NWIDth? query returns the measured width of the first negative pulse of the specified source.

Returned Format `[:MEASure:NWIDth] <value>[,<result_state>]<NL>`

<value> The width of the first negative pulse on the screen using the mid-threshold levels of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current width of the first negative pulse on the screen in the numeric variable, varWidth, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:NWIDth? CHANnel1"
varWidth = myScope.ReadNumber
Debug.Print FormatNumber(varWidth, 0)
```

:MEASure:OVERshoot

Command `:MEASure:OVERshoot [<source>][,<direction>]`

The :MEASure:OVERshoot command measures the overshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:OVERshoot command.

The algorithm is:

If the first edge on the screen is rising,

then

overshoot = (Local Vmax - Vtop) / Vamplitude

else

overshoot = (Vbase - Local Vmin) / Vamplitude.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> RISing or FALLing

Example This example measures the overshoot of the first edge on the screen.

```
myScope.WriteString ":MEASURE:OVERSHOOT CHANNEL1"
```

Query `:MEASure:OVERshoot? [<source>][,<direction>]`

The :MEASure:OVERshoot? query returns the measured overshoot of the specified source.

Returned Format `[:MEASure:OVERshoot] <value>[,<result_state>]<NL>`

<value> Ratio of overshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of overshoot in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:OVERSHOOT? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:PAMPlitude

Command `:MEASure:PAMPlitude [<source>, <width>, <direction>]`

The :MEASure:PAMPlitude command measures the pulse amplitude around the specified edge. There is only a single width applied to the top and base for the amplitude measurement.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<width> width to measure at the top and base of the pulse (in percent, 0-100)

<direction> the edge direction to measure (RISing or FALLing). The pulse measured is to the left and right of the specified edge.

Example This example measures the pulse amplitude around a rising edge (width set to 50%)

```
myScope.WriteString ":MEASURE:PAMPlitude CHAN1, 50, RISing"
```

Query `:MEASure:PAMPlitude? <source>, <width>, <direction>`

The :MEASure:PAMPlitude? query returns the pulse amplitude around the specified edge.

:MEASure:PBASe

Command `:MEASure:PBASe <source>, <pulse width percent>`

The :MEASure:PBASe command measures the average of the data of a negative pulse within the pulse window. The pulse window is a range of data centered within the pulse width using the specified percentage of the data as measured as the middle threshold level. For example, a 50% window would not include in the average the first or last 25% of the pulse width as measured at the middle threshold level. A 100% window would measure the average of the entire positive or negative pulse. In measure all edges mode and EZJIT, these measurements can be trended, histogrammed, etc.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<pulse width percent> pulse width percent to use in average (in percent, 0-100)

Example This example measures the average of the data of a negative pulse within the pulse window (width set to 50%)

```
myScope.WriteString ":MEASURE:PBASe CHAN1, 50"
```

Query `:MEASure:PBASe? <source>, <pulse width percentage>`

The :MEASure:PBASe? query returns the average pulse base of the data of a negative pulse within the specified pulse window.

:MEASure:PERiod

Command `:MEASure:PERiod [<source>[,<direction>]]`

The :MEASure:PERiod command measures the period of the first complete cycle on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected).

The source is specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PERiod command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   period = second rising edge time - first rising edge time
else
   period = second falling edge time - first falling edge time
```

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<direction> {RISing | FALLing}

Specifies direction of edge to start measurement. When <direction> is specified, the <source> parameter is required.

Using the <direction> parameter will set the "Measure All Edges" mode if it is not currently set. You can use the :MEASure:JITTer:STATistics command to turn the "Measure All Edges" mode off (or on again). See ":MEASure:JITTer:STATistics" on page 576.

Example This example measures the period of the waveform.

```
myScope.WriteString ":MEASure:PERiod CHANnel1"
```

Query `:MEASure:PERiod? [<source>],<direction>`

The :MEASure:PERiod? query returns the measured period of the specified source.

Returned Format `[:MEASure:PERiod] <value>[,<result_state>]<NL>`

<value> Period of the first complete cycle on the screen.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current period of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:PERiod? CHANnel1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:PHASe

Command `:MEASure:PHASe [<source>[,<source>[,<direction>]]]`

The :MEASure:PHASe command measures the phase in degrees between two edges. If two sources are specified, the phase from the specified edge of the first source to the specified edge of the second source is measured. If one source is specified, the phase is always 0.0E0.00°.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> {RISing | FALLing}

Specifies direction of edge to measure. When <direction> is specified, the <source> parameter is required.

Using the <direction> parameter will set the "Measure All Edges" mode if it is not currently set. You can use the :MEASure:JITTer:STATistics command to turn the "Measure All Edges" mode off (or on again). See ":MEASure:JITTer:STATistics" on page 576.

Example This example measures the phase between channel 1 and channel 2.

```
myScope.WriteString ":MEASure:PHASe CHANnel1,CHANnel2"
```

Query `:MEASure:PHASe? [<source>[,<source>[,<direction>]]]`

The :MEASure:PHASe? query returns the measured phase angle value.

The necessary waveform edges must be present on the display. The query will return 9.99999E+37 if the necessary edges are not displayed.

Returned Format `[:MEASure:PHASe] <value>[,result_state]<NL>`

<value> Phase angle from the first edge on the first source to the first edge on the second source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current phase angle value between channel 1 and channel 2 in the variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:PHASe? CHANnel1,CHANnel2"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:PPULses

Command `:MEASure:PPULses <source>`

The :MEASure:PPULses measures the number of positive pulses on the screen.

<source> the source on which the measurement is made

Example This example measures the number of positive pulses on channel 1.

```
myScope.WriteString ":MEASURE:PPULses CHAN1"
```

Query :MEASure:PPULses?

This query returns the result for the PPULses measurement.

:MEASure:PREShoot

Command `:MEASure:PREShoot [<source>]`

The :MEASure:PREShoot command measures the preshoot of the first edge on the screen. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PREShoot command.

The algorithm is:

If the first edge on the screen is rising,

then

preshoot = (Vbase - Local Vmin) / Vamplitude

else

preshoot = (Local Vmax - Vtop) / Vamplitude.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the preshoot of the waveform on the screen.

```
myScope.WriteString ":MEASURE:PRESHOOT CHANNEL1"
```

Query `:MEASure:PREShoot?[<source>]`

The :MEASure:PREShoot? query returns the measured preshoot of the specified source.

Returned Format `[:MEASure:PREShoot] <value>[,<result state>]<NL>`

<value> Ratio of preshoot to amplitude, in percent.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of preshoot in the numeric variable, varPreshoot, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:PRESHOOT? CHANNEL1"
varPreshoot = myScope.ReadNumber
Debug.Print FormatNumber(varPreshoot, 0)
```

:MEASure:PTOP

Command `:MEASure:PTOP <source>, <pulse width percent>`

The :MEASure:PTOP command measures the average of the data of a positive pulse within the pulse window. The pulse window is a range of data centered within the pulse width using the specified percentage of the data as measured as the middle threshold level. For example, a 50% window would not include in the average the first or last 25% of the pulse width as measured at the middle threshold level. A 100% window would measure the average of the entire positive or negative pulse. In measure all edges mode and EZJIT, these measurements can be trended, histogrammed, etc.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<pulse width percent> pulse width percent to use in average (in percent, 0-100)

Example This example measures the average of the data of a positive pulse within the pulse window (width set to 50%)

```
myScope.WriteString ":MEASURE:PTOP CHAN1, 50"
```

Query `:MEASure:PTOP? <source>, <pulse width percentage>`

The :MEASure:PTOP? query returns the average of the data of a positive pulse within the specified pulse window.

:MEASure:PWIDth

Command `:MEASure:PWIDth [<source>]`

The :MEASure:PWIDth command measures the width of the first positive pulse on the screen using the mid-threshold levels of the waveform (50% levels with standard measurements selected). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:PWIDth command.

The algorithm is:

```
If the first edge on the screen is rising,
then
   pwidth = time at the first falling edge - time at the
   first rising edge
else
   pwidth = time at the second falling edge - time at the
   first rising edge
```

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example measures the width of the first positive pulse on the screen.

```
myScope.WriteString ":MEASure:PWIDth CHANnel1"
```

Query `:MEASure:PWIDth?[<source>]`

The :MEASure:PWIDth? query returns the measured width of the first positive pulse of the specified source.

Returned Format `[:MEASure:PWIDth] <value>[,<result_state>]<NL>`

<value> Width of the first positive pulse on the screen in seconds.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the width of the first positive pulse on the screen in the numeric variable, varWidth, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:PWIDth? CHANnel1"
varWidth = myScope.ReadNumber
Debug.Print FormatNumber(varWidth, 0)
```

:MEASure:QUALifier<M>:CONDition

Command `:MEASure:QUALifier<M>:CONDition {HIGH | LOW | INSide | OUTSide}`

The :MEASure:QUALifier<M>:CONDition command sets the condition when valid timing measurements are made

- Above Middle Threshold (HIGH)
- Below Middle Threshold (LOW)
- Between Upper, Lower Thresholds (INSide)
- Not Between Thresholds (OUTSide)

<M> An integer, 1-3.

Example This example sets the level qualifier 2 condition to HIGH.

```
myScope.WriteString ":MEASURE:QUALIFIER2:CONDITION HIGH"
```

Query `:MEASure:QUALifier<M>:CONDition?`

The :MEASure:QUALifier<M>:CONDition? query returns the condition being used of the level qualifier.

Returned Format `[:MEASure:QUALifier<M>:CONDition] <source><NL>`

Example This example places the current condition of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:QUALIFIER2:CONDition?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

:MEASure:QUALifier<M>:SOURce

Command

NOTE The channel being selected must not be used to make a timing measurement and must be turned on.

```
:MEASure:QUALifier<M>:SOURce <source>
```

The :MEASure:QUALifier<M>:SOURce command sets the source of the level qualify for timing measurements.

<source> CHANnel<N>

<N> An integer, 1- 4.

<M> An integer, 1-3.

Example This example sets the level qualifier 2 source to the channel 1 waveform.

```
myScope.WriteString ":MEASURE:QUALIFIER2:SOURce CHANNEL1"
```

Query

```
:MEASure:QUALifier<M>:SOURce?
```

The :MEASure:QUALifier<M>:SOURce? query returns the source being used of the level qualifier for timing measurements.

Returned Format

```
[:MEASure:QUALifier<M>:SOURce] <source><NL>
```

Example This example places the current source of level qualifier for timing measurements in the source variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:QUALIFIER2:SOURce?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

:MEASure:QUALifier<M>:STATe

Command `:MEASure:QUALifier<M>:STATe {{ON | 1} | {OFF | 0}}`

The :MEASure:QUALifier<M>:STATe command enables or disables level qualifying for timing measurements.

<M> An integer, 1-3.

Example This example sets the level qualifier 2 state to ON.

```
myScope.WriteString ":MEASURE:QUALIFIER2:STATE ON"
```

Query `:MEASure:QUALifier<M>:STATe?`

The :MEASure:QUALifier<M>:STATe? query returns the state of the level qualifier for timing measurements.

Returned Format `[:MEASure:QUALifier<M>:SOURce] {1 | 0}<NL>`

Example This example places the current state of the level qualifier for timing measurements in the state variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:QUALIFIER2:STATE?"
varstate = myScope.ReadNumber
Debug.Print FormatNumber(varstate, 0)
```

:MEASure:RESults?

Query `:MEASure:RESults?`

The :MEASure:RESults? query returns the results of the continuously displayed measurements. The response to the MEASure:RESults? query is a list of comma-separated values. If SENDvalid is ON, the results state is returned.

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

Returned Format `[:MEASure:RESults] <result_list><NL>`

<result_list> A list of the measurement results separated with commas. The following shows the order of values received for a single measurement if :MEASure:STATistics is set to ON.

Measurement label	current	result state	min	max	mean	std dev	# of meas

Min, max, mean, std dev, and # of meas are only returned if the :MEASure:STATistics is ON. The result state is only returned if :MEASure:SENDvalid is ON. See Table 24 for the meaning of the result state codes.

If the :MEASure:STATistics is set to CURRENT, MAX, MEAN, MIN, or STDDEV only that particular statistic value is returned for each measurement that is on.

Example This example places the current results of the measurements in the string variable, strResult, then prints the contents of the variable to the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":MEASURE:RESULTS?"
strResult = myScope.ReadString
Debug.Print strResult
```

Table 24 Result States

Code	Description
0	Result correct. No problem found.
1	Result questionable but could be measured.
2	Result less than or equal to value returned.
3	Result greater than or equal to value returned.
4	Result returned is invalid.
5	Result invalid. Required edge not found.
6	Result invalid. Max not found.
7	Result invalid. Min not found.
8	Result invalid. Requested time not found.
9	Result invalid. Requested voltage not found.
10	Result invalid. Top and base are equal.
11	Result invalid. Measurement zone too small.
12	Result invalid. Lower threshold not on waveform.
13	Result invalid. Upper threshold not on waveform.
14	Result invalid. Upper and lower thresholds are too close.
15	Result invalid. Top not on waveform.
16	Result invalid. Base not on waveform.
17	Result invalid. Completion criteria not reached.
18	Result invalid. Measurement invalid for this type of waveform.
19	Result invalid. waveform is not displayed.
20	Result invalid. Waveform is clipped high.
21	Result invalid. Waveform is clipped low.
22	Result invalid. Waveform is clipped high and low.
23	Result invalid. Data contains all holes.
24	Result invalid. No data on screen.
29	Result invalid. FFT peak not found.
30	Result invalid. Eye pattern not found.
31	Result invalid. No NRZ eye pattern found.
33	Result invalid. There is more than one source on creating the database.
35	Signal may be too small to evaluate.

Table 24 Result States (continued)

Code	Description
36	Result invalid. Awaiting completion of averaging.
39	Result invalid. Need jitter package to make this measurement or must be in jitter mode to make this measurement.
40	Current measurement is not on screen.
41	Not enough points available to recover the clock.
42	The loop bandwidth of the PLL is too high to recover the clock.
43	RJDJ pattern not found in data.
45	Clock recovery mode is not permitted.
46	Too much jitter to make a RJDJ separation.

:MEASure:RISetime

Command `:MEASure:RISetime [<source>]`

The :MEASure:RISetime command measures the rise time of the first displayed edge by measuring the time at the lower threshold of the rising edge, measuring the time at the upper threshold of the rising edge, then calculating the rise time with the following algorithm:

Rise time = time at upper threshold point - time at lower threshold point.

To make this measurement requires 4 or more sample points on the rising edge of the waveform.

Sources are specified with the :MEASure:SOURce command or with the optional parameter following the RISetime command. With standard thresholds selected, the lower threshold is at the 10% point and the upper threshold is at the 90% point on the rising edge.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the rise time of the channel 1 waveform.

```
myScope.WriteString ":MEASURE:RISETIME CHANNEL1"
```

Query `:MEASure:RISetime?[<source>]`

The :MEASure:RISetime? query returns the rise time of the specified source.

Returned Format `[:MEASure:RISetime] <value>[,<result_state>]<NL>`

<value> Rise time in seconds.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of rise time in the numeric variable, varRise, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RISETIME? CHANNEL1"
varRise = myScope.ReadNumber
Debug.Print FormatNumber(varRise, 0)
```

:MEASure:RJDJ:ALL?

Query

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:ALL?
```

The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements. These values are returned as comma separated values using the following format:

Returned Format

```
[:MEASure:RJDJ:ALL<space>]
TJ(<format>),<result>,<state>,
RJ(<format>),<result>,<state>,
DJ(<format>),<result>,<state>,
PJ(<format>),<result>,<state>,
BUJ(<format>),<result>,<state>,
DDJ(<format>),<result>,<state>,
DCD,<result>,<state>,
ISI(<format>),<result>,<state>,
Transitions,<number_of_transitions>,<transitions_state>,
Scope RJ(<format>),<result>,<state>,
DDPWS,<result>,<state>,
ABUJ(<format>),<result>,<state><NL>
```

NOTE Whether some of these values are included or not depends on the setting of :MEASure:RJDJ:METHod and :MEASure:RJDJ:REPort.

For example, when :MEASure:RJDJ:REPort or :MEASure:RJDJ:METHod is SPECtral, the BUJ and ABUJ values are not returned, and there are two PJ values (one "rms" and one "dd").

<space> White space (ASCII 32) character.

<format> The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (dd) means the measurement uses a dual-Dirac delta model to derive the measurement. A format of (pp) means the measurement is a peak-to-peak measurement.

<result> The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<state> The measurement result state. See Table 24 for a list of values and descriptions of the result state value.

<number_of_transitions> The number of waveform transitions that have been measured.

Example This example places the jitter measures in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:ALL?"
strResults = myScope.ReadString
Debug.Print strResults
```

See Also

- ":MEASure:RJDJ:METHod" on page 633
- ":MEASure:RJDJ:REPort" on page 636

:MEASure:RJDJ:APLength?

Query

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:APLength?
```

The :MEASure:RJDJ:APLength? query returns the determined RjDj pattern length.

Returned Format `[:MEASure:RJDJ:APLength] <value><NL>`

<value> The determined RjDj pattern length as a numeric data value. Invalid (9.99999E+37) is returned if there is no data.

Example This example places the calculated pattern length in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTEM:HEADER OFF"   ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:APLength?"
strResults = myScope.ReadString
Debug.Print strResults
```

:MEASure:RJDJ:BANDwidth

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:BANDwidth {NARRow | WIDE}
```

The :MEASure:RJDJ:BANDwidth command sets the type of filtering used to separate the data dependent jitter form the random jitter and the periodic jitter.

Example This example sets the RJ bandwidth to WIDE.

```
myScope.WriteString ":MEASURE:RJDJ:BANDWIDTH WIDE"
```

Query

```
:MEASure:RJDJ:BANDwidth?
```

The :MEASure:RJDJ:BANDwidth? query returns the RJ bandwidth filter setting.

Returned Format

```
[:MEASure:RJDJ:BANDwidth] {NARRow | WIDE}<NL>
```

Example This example places the RJ filter setting the varFilter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:BANDWIDTH?"
varFilter = myScope.ReadNumber
Debug.Print FormatNumber(varFilter, 0)
```

:MEASure:RJDJ:BER

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:BER {E6 | E7 | E8 | E9 | E10 | E11 | E12 | E13 | E14
                   | E15 | E16 | E17 | E18 | J2 | J9}
```

The :MEASure:RJDJ:BER command sets the bit error rate for the Total Jitter (TJ) measurement. The E and J parameters have the following bit error rate meanings:

- E6 = 1E-6
- E7 = 1E-7
- E8 = 1E-8
- E9 = 1E-9
- E10 = 1E-10
- E11 = 1E-11
- E12 = 1E-12
- E13 = 1E-13
- E14 = 1E-14
- E15 = 1E-15
- E16 = 1E-16
- E17 = 1E-17
- E18 = 1E-18
- J2 = 2.5E-3
- J9 = 2.5E-10

Example This example sets the bit error rate to E16.

```
myScope.WriteString ":MEASURE:RJDJ:BER E16"
```

Query

```
:MEASure:RJDJ:BER?
```

The :MEASure:RJDJ:BER? query returns the bit error rate setting.

Returned Format

```
[:MEASure:RJDJ:BER] {E6 | E7 | E8 | E9 | E10 | E11 | E12 | E13 | E14
                     | E15 | E16 | E17 | E18 | J2 | J9}<NL>
```

Example This example places the bit error rate in the varRate variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:BER?"
varRate = myScope.ReadNumber
Debug.Print FormatNumber(varRate, 0)
```

:MEASure:RJDJ:CLOCk

Command

NOTE This command is available when the EZJIT+ or EZJIT Complete jitter analysis software is installed.

```
:MEASure:RJDJ:CLOCk {ON | OFF}
```

When the :MEASure:RJDJ:CLOCk command is set to ON, it forces the pattern to be a clock and sets the jitter for edges not examined to zero (0).

To measure jitter on only rising (or falling) edges of a clock, you must also set :MEASure:CLOCk:METHod:EDGE to RISing or FALLing, and you must set :MEASure:RJDJ:EDGE to the same RISing or FALLing option.

Example This example turns on the RJDJ clock option.

```
myScope.WriteString ":MEASURE:RJDJ:CLOCk ON"
```

Query

```
:MEASure:RJDJ:CLOCk?
```

The :MEASure:RJDJ:CLOCk? query returns the setting.

Returned Format

```
[:MEASure:RJDJ:CLOCk] {ON | OFF}<NL>
```

Example This example places the current RJDJ clock setting in the strSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:CLOCk?"
strSetting = myScope.ReadNumber
Debug.Print strSetting
```

See Also

- ":MEASure:CLOCk:METHod:EDGE" on page 506
- ":MEASure:RJDJ:EDGE" on page 631

:MEASure:RJDJ:EDGE

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:EDGE {RISING | FALLING | BOTH}
```

The :MEASure:RJDJ:EDGE command sets the edge used for the RJDJ measurements.

Example This example sets the RJDJ edge to use both edges.

```
myScope.WriteString ":MEASURE:RJDJ:EDGE BOTH"
```

Query

```
:MEASure:RJDJ:EDGE?
```

The :MEASure:RJDJ:EDGE? query returns the edge being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:EDGE] {RISING | FALLING | BOTH}<NL>
```

Example This example places the current edge being used for RJDJ measurements in the varEdge variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:EDGE?"
varEdge = myScope.ReadNumber
Debug.Print FormatNumber(varEdge, 0)
```

:MEASure:RJDJ:INTerpolate

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:INTerpolate {LINear | NONE}
```

The :MEASure:RJDJ:INTerpolate command sets the interpolation mode used for the RJDJ measurements.

Example This example sets the RJDJ interpolation to use both linear.

```
myScope.WriteString ":MEASURE:RJDJ:INTERPOLATE LINEAR"
```

Query

```
:MEASure:RJDJ:INTerpolate?
```

The :MEASure:RJDJ:INTerpolate? query returns the edge being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:INTerpolate] {LINear | NONE}<NL>
```

Example This example places the current interpolation mode being used for RJDJ measurements in the interpolate variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:INTERPOLATE?"
varinterpolate = myScope.ReadNumber
Debug.Print FormatNumber(varinterpolate, 0)
```

:MEASure:RJDJ:METHod

Command

NOTE This command is only available when the EZJIT+ jitter analysis application is licensed.

```
:MEASure:RJDJ:METHod {SPECtral | BOTH}
```

The :MEASure:RJDJ:METHod command lets you select the method for random jitter (RJ) analysis, either the SPECtral method or BOTH the spectral and tail fit methods.

When analyzing jitter with crosstalk or ground bounce effects present in your signal, select BOTH. When this option is selected, the deterministic jitter (DJ) that is uncorrelated to the data pattern, also known as bounded uncorrelated jitter (BUJ), is separated into periodic jitter (PJ) and aperiodic bounded uncorrelated jitter (ABUJ). ABUJ is caused by crosstalk and ground bounce effects.

When there are no crosstalk or ground bounce effects present in your signal, you can select the SPECtral method in order to run faster. When this option is selected, the deterministic jitter (DJ) that is uncorrelated to the data pattern is all reported as periodic jitter (PJ).

Example This example sets the RJDJ method to BOTH the spectral and tail fit analysis.

```
myScope.WriteString ":MEASURE:RJDJ:METHod BOTH"
```

Query

```
:MEASure:RJDJ:METHod?
```

The :MEASure:RJDJ:METHod? query returns the selected RJDJ method.

Returned Format

```
[:MEASure:RJDJ:METHod] {SPEC | BOTH}<NL>
```

Example This example places the RJDJ method setting the strJitterMethod variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:METHod?"
strJitterMethod = myScope.ReadString
Debug.Print strJitterMethod
```

See Also

- ":MEASure:RJDJ:REPort" on page 636

:MEASure:RJDJ:MODe

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:MODe {TIE | PERiod | NUI[,<ui>]}
```

The :MEASure:RJDJ:MODe command sets the RJDJ measurement mode. If NUI is selected then <ui> selects the number of unit intervals (for example: :MEASure:RJDJ:MODe NUI,5.

Example This example sets the RJDJ mode to TIE.

```
myScope.WriteString ":MEASURE:RJDJ:MODe TIE"
```

Query

```
:MEASure:RJDJ:MODe?
```

The :MEASure:RJDJ:MODe? query returns the mode of the RJDJ measurements.

:MEASure:RJDJ:PLENgth

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:PLENgth {AUTO
    | {ARBitrary,<isi_filter_lead>,<isi_filter_lag>}
    | <number_of_bits>}
```

The :MEASure:RJDJ:PLENgth command sets the number of bits used pattern length for the RJDJ measurements.

<isi_filter_ lead> An integer number that is less than or equal to 0 that is the number of leading bits that are used to calculate the ISI filter.

<isi_filter_ lag} An integer number that is greater than or equal to 0 that is the number of trailing bits used to calculate the ISI filter.

<number_of_ bits> An integer number that is the length of pattern from 2 to 1024.

Example This example sets the RJDJ bits to 5.

```
myScope.WriteString ":MEASURE:RJDJ:PLENgth 5"
```

Query

```
:MEASure:RJDJ:PLENgth?
```

The :MEASure:RJDJ:PLENgth? query returns the number of bits being used for the RJDJ measurements when Periodic pattern length is set. For Arbitrary pattern length, the ISI filter lead and filter lag numbers are returned.

Returned Format

```
[MEASure:RJDJ:PLENgth] {AUTO
    | ARBitrary,<isi_filter_lead>,<isi_filter_lag>
    | <number_of_bits>}<NL>
```

Example This example places the current number of bits for RJDJ measurements in the varBits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:PLENgth?"
varBits = myScope.ReadNumber
Debug.Print FormatNumber(varBits, 0)
```

:MEASure:RJDJ:REPort

Command

NOTE This command is only available when the EZJIT+ jitter analysis application is licensed.

```
:MEASure:RJDJ:REPort {SPECtral | TAILfit}
```

When the :MEASure:RJDJ:METHod BOTH command selects both the spectral and tail fit methods for random jitter analysis, the :MEASure:RJDJ:REPort command specifies which method is used for the reports in the jitter graphs/histograms and Jitter tab measurements.

Example This example specifies that the RJDJ report include measurements from both the spectral and tail fit analysis (including aperiodic bounded uncorrelated jitter ABUJ measurements).

```
myScope.WriteString ":MEASURE:RJDJ:REPort TAILfit"
```

Query

```
:MEASure:RJDJ:REPort?
```

The :MEASure:RJDJ:REPort? query returns the report setting.

Returned Format

```
[:MEASure:RJDJ:REPort] {SPEC | TAIL}<NL>
```

Example This example places the report setting in the strReportSetting variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:REPort?"
strReportSetting = myScope.ReadString
Debug.Print strReportSetting
```

See Also

- ":MEASure:RJDJ:METHod" on page 633

:MEASure:RJDJ:RJ

Command

NOTE This command is available when the EZJIT+ software is installed.

```
:MEASure:RJDJ:RJ {ON, <RJrms> | OFF}
```

The :MEASure:RJDJ:RJ command can specify a known amount of random jitter. When used, the remaining amount of the total jitter measured is reported as periodic jitter (PJ).

This command is used in situations when crosstalk aggressors influence the random jitter measured on a signal. If the random jitter on a signal is measured without the aggressor signal crosstalk, this known amount of random jitter can be specified when measuring the jitter again with the crosstalk aggressors.

- ON – Enables a specified amount of random jitter.
- <RJrms> – The known amount of random jitter.
- OFF – Disables the specification of known random jitter.

The amount of random jitter is shown in the jitter measurement results (see page 624) as "RJ(rms specified)".

Example This example specifies 500 fs of random jitter.

```
myScope.WriteString ":MEAS:RJDJ:RJ ON, 500e-15"
```

Query

```
:MEASure:RJDJ:RJ?
```

The :MEASure:RJDJ:RJ? query returns the specified RJ settings.

Returned Format

```
[:MEASure:RJDJ:RJ] {ON, <RJrms> | OFF}<NL>
```

Example This example places the specified RJ settings in the strKnownRandomJitter variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:RJ?"
strKnownRandomJitter = myScope.ReadString
Debug.Print strKnownRandomJitter
```

:MEASure:RJDJ:SCOPe:RJ

Command

NOTE This command is available when the EZJIT+ software is installed.

```
:MEASure:RJDJ:SCOPe:RJ {ON, <RJrms> | AUTO | OFF}
```

The :MEASure:RJDJ:SCOPe:RJ command can specify the removal of the oscilloscope's calibrated random jitter from the reported RJ.

- ON — Enables the "manual" removal of a known oscilloscope random jitter from the reported RJ.
- <RJrms> — The known oscilloscope random jitter to remove from the reported RJ.
- AUTO — This option cannot be selected until the scope jitter calibration has been run (use the **Calibrate scope jitter** button in the front panel user interface). When selected, the calculated oscilloscope random jitter is removed from the reported RJ.

  The calculated oscilloscope random jitter is shown in the jitter measurement results (see page 624) as "Scope RJ(rms)".
- OFF — Disables the removal of the oscilloscope's calibrated random jitter from the reported RJ.

Example This example specifies 300 fs of known oscilloscope random jitter.

```
myScope.WriteString ":MEASure:RJDJ:SCOPe:RJ ON, 300e-15"
```

Query

```
:MEASure:RJDJ:SCOPe:RJ?
```

The :MEASure:RJDJ:SCOPe:RJ? query returns the oscilloscope RJ settings.

Returned Format

```
[:MEASure:RJDJ:SCOPe:RJ] {ON, <RJrms> | OFF}<NL>
```

Example This example places the oscilloscope RJ settings in the strScopeRJSettings variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:RJDJ:SCOPe:RJ?"
strScopeRJSettings = myScope.ReadString
Debug.Print strScopeRJSettings
```

:MEASure:RJDJ:SOURce

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:SOURce <source>
```

The :MEASure:RJDJ:SOURce command sets the source for the RJDJ measurements.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example sets the RJDJ source to the channel 1 waveform.

```
myScope.WriteString ":MEASURE:RJDJ:SOURce CHANNEL1"
```

Query

```
:MEASure:RJDJ:SOURce?
```

The :MEASure:RJDJ:SOURce? query returns the source being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:SOURce] <source><NL>
```

Example This example places the current source for RJDJ measurements in the varSource variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:SOURce?"
varSource = myScope.ReadNumber
Debug.Print FormatNumber(varSource, 0)
```

:MEASure:RJDJ:STATe

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:STATe {ON | OFF}
```

The :MEASure:RJDJ:STATe command enables or disables the RJDJ measurements.

Example This example sets the RJDJ state to on.

```
myScope.WriteString ":MEASURE:RJDJ:STATE ON"
```

Query

```
:MEASure:RJDJ:STATe?
```

The :MEASure:RJDJ:STATe? query returns the state of the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:STATe] {1 | 0}<NL>
```

Example This example places the current state of the RJDJ measurements in the varState variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:STATE?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

:MEASure:RJDJ:TJRJDJ?

Query

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:TJRJDJ?
```

The :MEASure:RJDJ:TJRJDJ? query returns the Total Jitter (TJ), Random Jitter (RJ), and the Deterministic Jitter (DJ) measurements. These values are returned as comma separated values using the following format:

Returned Format

```
[:MEASure:RJDJ:TJRJDJ] TJ(<tj_format>),<tj_result>,<tj_state>,
RJ(<rj_format>),<rj_result>,rj_state,
DJ(<dj_format>),<dj_result>,<dj_state><NL>
```

<tj_format>
<rj_format>
<dj_format>

The format value tells you something about how the measurement is made. For instance, TJ(1E-12) means that the TJ measurement was derived using a bit error rate of 1E-12. A format of (rms) means the measurement is a root-mean-square measurement. A format of (d-d) means the measurement uses from a dual-Dirac delta model used to derive the measurement. A format of (p-p) means the measurement is a peak-to-peak measurement.

<tj_result>
<rj_result>
<dj_result>

The measured results for the RJDJ measurements. A value of 9.99999E+37 means that the oscilloscope was unable to make the measurement.

<tj_state>
<rj_state>
<dj_state>

The measurement result state. See Table 24 for a list of values and descriptions of the result state value.

Example

This example places the Total Jitter (TJ), Random Jitter (RJ), and the Deterministic Jitter (DJ) measurements in the strResults variable and displays it on the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTEM:HEADER OFF"   ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:TJRJDJ?"
strResult = myScope.ReadString
Debug.Print strResult
```

:MEASure:RJDJ:UNITs

Command

NOTE This command is only available when the N5400A/N5401A Software is installed.

```
:MEASure:RJDJ:UNITs {SECond | UNITinterval}
```

The :MEASure:RJDJ:UNITs command sets the unit of measure for RJDJ measurements to seconds or unit intervals.

Example This example sets the RJDJ units to unit interval.

```
myScope.WriteString ":MEASURE:RJDJ:UNITS UNITINTERVAL"
```

Query

```
:MEASure:RJDJ:UNITs?
```

The :MEASure:RJDJ:UNITs? query returns the units of measure being used for the RJDJ measurements.

Returned Format

```
[:MEASure:RJDJ:UNITs] {SECond | UNITinterval}<NL>
```

Example This example places the current units of measure for the RJDJ measurements in the varUnits variable and displays it on the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:RJDJ:UNITS?"
varUnits = myScope.ReadNumber
Debug.Print FormatNumber(varUnits, 0)
```

:MEASure:SCRatch

Command `:MEASure:{SCRatch | CLEar}`

The :MEASure:SCRatch command clears the measurement results from the screen. This command performs the same function as :MEASure:CLEar.

Example This example clears the current measurement results from the screen.

```
myScope.WriteString ":MEASURE:SCRATCH"
```

:MEASure:SENDvalid

Command `:MEASure:SENDvalid {{OFF|0} | {ON|1}}`

The :MEASure:SENDvalid command enables the result state code to be returned with the :MEASure:RESults? query and all other measurement queries.

Example This example turns the send valid function on.

```
myScope.WriteString ":MEASURE:SENDVALID ON"
```

Query `:MEASure:SENDvalid?`

The :MEASure:SENDvalid? query returns the state of the send valid control.

Returned Format `{:MEASure:SENDvalid] {0 | 1}<NL>`

Example This example places the current mode for SENDvalid in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASURE:SENDVALID?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also Refer to the :MEASure:RESults? query for information on the results returned and how they are affected by the SENDvalid command. Refer to the individual measurements for information on how the result state is returned.

:MEASure:SETuptime

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

```
:MEASure:SETuptime [<data_source>,<data_source_dir>,<clock_source>,
<clock_ source_dir>]
```

The :MEASure:SETuptime command measures the setup time between the specified clock and data source.

<data_source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<clock_source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<data_source_dir> {RISing | FALLing | BOTH}

Selects the direction of the data source edge. BOTH selects both edges to be measured.

<clock_source_dir> {RISing | FALLing}

Selects the direction of the clock source edge.

Example This example measures the setup time from the rising edge of channel 1 to the rising edge of channel 2.

```
myScope.WriteString ":MEASure:SETuptime CHAN1,RIS,CHAN2,RIS"
```

Query

```
:MEASure:SETuptime? [<data_source>,<data_source_dir>,<clock_source>,
<clock_ source_dir>]
```

The :MEASure:SETuptime query returns the measured setup time between the specified clock and data source.

Returned Format

```
{:MEASure:SETuptime] <value><NL>
```

<value> Setup time in seconds.

Example This example places the current value of setup time in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:SETuptime? CHAN1,RIS,CHAN2,RIS"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MEASure:SLEWrate

Command

NOTE This command is only available when the E2681A Jitter Analysis Software is installed.

```
:MEASure:SLEWrate [<data_source>, <edge_dir>]
```

The :MEASure:SLEWrate command measures the slew rate of the specified data source.

<data_source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> An integer, 1-4.

<F> An integer, 1-16.

<edge_dir> {RISing | FALLing | BOTH}

Example This example measures the slew rate of channel 1.

```
myScope.WriteString ":MEASure:SLEWrate CHANnel1,RISing"
```

Query

```
:MEASure:SLEWrate? [<data_source>, <edge_dir>]
```

The :MEASure:SLEWrate? query returns the measured slew rate for the specified source.

Returned Format

```
{:MEASure:SLEWrate] <value><NL>
```

<value> Slew rate in volts per second.

Example This example places the channel 1 value of slew rate in the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:SLEWrate? CHANnel1,RISing"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MEASure:SOURce

Command `:MEASure:SOURce {<source>[,<source>]}`

The :MEASure:SOURce command selects the source for measurements. You can specify one or two sources with this command. All measurements except :MEASure:HOLDtime, :MEASure:SETUPtime, and :MEASure:DELTatime are made on the first specified source. The delta time measurement uses two sources if two are specified.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 1 as the source for measurements.

```
myScope.WriteString ":MEASure:SOURce CHANnel1"
```

Query `:MEASure:SOURce?`

The :MEASure:SOURce? query returns the current source selection.

Returned Format `[:MEASure:SOURce] <source>[,<source>]<NL>`

Example This example places the currently specified sources in the string variable, strSource, then prints the contents of the variable to the computer's screen.

```
Dim strSource As String   ' Dimension variable.
myScope.WriteString ":MEASure:SOURce?"
strSource = myScope.ReadString
Debug.Print strSource
```

:MEASure:STATistics

Command `:MEASure:STATistics {{ON | 1} | CURRent | MAXimum | MEAN | MINimum | STDDev}`

The :MEASure:STATistics command determines the type of information returned by the :MEASure:RESults? query. ON means all the statistics are on.

Example This example turns all the statistics function on.

```
myScope.WriteString ":MEASURE:STATISTICS ON"
```

Query `:MEASure:STATistics?`

The :MEASure:STATistics? query returns the current statistics mode.

Returned Format `[:MEASure:STATistics] {ON | CURRent | MAXimum | MEAN | MINimum | STDDev} <NL>`

Example This example places the current mode for statistics in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASURE:STATISTICS?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also Refer to the :MEASure:RESults? query for information on the result returned and how it is affected by the STATistics command.

:MEASure:TEDGe

Command `:MEASure:TEDGe <meas_thres_txt>,[<slope>]<occurrence>[,<source>]`

The :MEASure:TEDGe command measures the time interval between the trigger event and the specified edge (threshold level, slope, and transition). Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TEDGe command.

<meas_thres_txt> UPPer, MIDDle, or LOWer to identify the threshold.

<slope> { - (minus) for falling | + (plus) for rising | <none> (the slope is optional; if no slope is specified, + (plus) is assumed) }

<occurrence> An integer value representing the edge of the occurrence. The desired edge must be present on the display. Edges are counted with 1 being the first edge from the left on the display, and a maximum value of 65534.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:MEASure:TEDGe? <meas_thres_txt>,<slope><occurrence> [,<source>]`

The :MEASure:TEDGe? query returns the time interval between the trigger event and the specified edge (threshold level, slope, and transition).

Returned Format `[:MEASure:TEDGe] <time>[,<result_state>]<NL>`

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time interval between the trigger event and the 90% threshold on the second rising edge of the source waveform to the numeric variable, varTime. The contents of the variable are then printed to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:TEDGe? UPPER,+2,CHANnel1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:THResholds:ABSolute

Command

```
:MEASure:THResholds:ABSolute <source>,
<upper_volts>,<middle_volts>,<lower_volts>
```

The :MEASure:THResholds:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

NOTE

This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:ABSolute" on page 654
- ":MEASure:THResholds:RFALl:ABSolute" on page 672
- ":MEASure:THResholds:SERial:ABSolute" on page 684

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_volts>
<middle_volts>
<lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:ABSolute CHANnel2,0.9,0.5,0.1"
```

Query

```
:MEASure:THResholds:ABSolute? <source>
```

The :MEASure:THResholds:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:ABSolute] <upper_volts>,<middle_volts>,<lower_volts
><NL>
```

Example

This example returns the upper level, middle level, and lower level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:THResholds:GENeral:ABSolute

Command `:MEASure:THResholds:GENeral:ABSolute <source>, <upper_volts>,<middle_volts>,<lower_volts>`

The :MEASure:THResholds:GENeral:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_volts>
<middle_volts>
<lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:ABSolute CHANnel2,0.9,0
.5,0.1"
```

Query `:MEASure:THResholds:GENeral:ABSolute? <source>`

The :MEASure:THResholds:GENeral:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:GENeral:ABSolute] <upper_volts>,<middle_volts>,<lower_volts><NL>
```

Example

This example returns the upper level, middle level, and lower level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 652
- ":MEASure:THResholds:RFALl:ABSolute" on page 672
- ":MEASure:THResholds:SERial:ABSolute" on page 684

:MEASure:THResholds:GENeral:HYSTeresis

Command `:MEASure:THResholds:GENeral:HYSTeresis <source>,<range>,<level>`

The :MEASure:THResholds:GENeral:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:HYSTeresis CHANNEL2,0.9
,0.1"
```

Query `:MEASure:THResholds:GENeral:HYSTeresis? <source>`

The :MEASure:THResholds:GENeral:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:GENeral:HYSTeresis]<range>,<level><NL>
```

Example

This example returns the range and level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:HYSTeresis? CHANNEL1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:HYSTeresis" on page 666
- ":MEASure:THResholds:RFALl:HYSTeresis" on page 674
- ":MEASure:THResholds:SERial:HYSTeresis" on page 686

:MEASure:THResholds:GENeral:METHod

Command `:MEASure:THResholds:GENeral:METHod <source>,{ABSolute | PERCent | HYSTeresis}`

The :MEASure:THResholds:GENeral:METHod command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:METHod CHANnel1,HYSTeresis"
```

Query `:MEASure:THResholds:GENeral:METHod? <source>`

The :MEASure:THResholds:GENeral:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:GENeral:METHod <source>,] {ABSolute | PERCent | HYSTeresis}`

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 668
- ":MEASure:THResholds:RFALl:METHod" on page 676
- ":MEASure:THResholds:SERial:METHod" on page 688

:MEASure:THResholds:GENeral:PERCent

Command `:MEASure:THResholds:GENeral:PERCent <source>,<upper_pct>,<middle_pct>,<lower_pct>`

The :MEASure:THResholds:GENeral:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_pct>
<middle_pct>
<lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:PERCent CHANnel2,100,50
,0"
```

Query `:MEASure:THResholds:GENeral:PERCent? <source>`

The :MEASure:THResholds:GENeral:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:GENeral:PERCent] <upper_pct>,<middle_pcts>,<lower_pct><NL>
```

Example

This example returns the upper level, middle level, and lower level percentages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 670
- ":MEASure:THResholds:RFALl:PERCent" on page 678
- ":MEASure:THResholds:SERial:PERCent" on page 690

:MEASure:THResholds:GENeral:TOPBase:ABSolute

Command `:MEASure:TOPBase:THResholds:GENeral:ABSolute <source>,<top_volts>,<base_volts>`

The :MEASure:TOPBase:THResholds:GENeral:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<top_volts>
<base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:ABSolute CHANnel2,0.9,0.1"
```

Query `:MEASure:THResholds:GENeral:TOPBase:ABSolute? <source>`

The :MEASure:THResholds:GENeral:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format `[:MEASure:THResholds:GENeral:TOPBase:ABSolute] <top_volts>,<base_volts><NL>`

Example This example returns the top level and base level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:ABSolute? CHANn
el1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 696
- ":MEASure:THResholds:RFALl:TOPBase:ABSolute" on page 680
- ":MEASure:THResholds:SERial:TOPBase:ABSolute" on page 692

:MEASure:THResholds:GENeral:TOPBase:METHod

Command `:MEASure:THResholds:GENeral:TOPBase:METHod <source>,{ABSolute | HISTONLY | MINmax | STANdard}`

The :MEASure:THResholds:GENeral:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

NOTE These general-purpose threshold settings are used for everything except rise/fall measurements and protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:METHod CHANnel1
,HISTONLY"
```

Query `:MEASure:THResholds:GENeral:TOPBase:METHod?`

The :MEASure:THResholds:GENeral:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:GENeral:TOPBase:METHod] {ABSolute | HISTONLY | MINmax | STANdard}`

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:GENeral:TOPBase:METHod CHANnel1
"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 698
- ":MEASure:THResholds:RFALl:TOPBase:METHod" on page 682
- ":MEASure:THResholds:SERial:TOPBase:METHod" on page 694

:MEASure:THResholds:HYSTeresis

Command `:MEASure:THResholds:HYSTeresis <source>,<range>,<level>`

The :MEASure:THResholds:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE

This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:HYSTeresis" on page 656
- ":MEASure:THResholds:RFALl:HYSTeresis" on page 674
- ":MEASure:THResholds:SERial:HYSTeresis" on page 686

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:HYSTeresis CHANNEL2,0.9,0.1"
```

Query `:MEASure:THResholds:HYSTeresis? <source>`

The :MEASure:THResholds:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format `[:MEASure:THResholds:HYSTeresis]<range>,<level><NL>`

Example This example returns the range and level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:HYSTeresis? CHANNEL1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:THResholds:METHod

Command `:MEASure:THResholds:METHod <source>,{ABSolute | PERCent | HYSTeresis}`

The :MEASure:THResholds:METHod command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

NOTE This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:METHod" on page 658
- ":MEASure:THResholds:RFALl:METHod" on page 676
- ":MEASure:THResholds:SERial:METHod" on page 688

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:METHod CHANnel1,HYSTeresis"
```

Query `:MEASure:THResholds:METHod? <source>`

The :MEASure:THResholds:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:METHod <source>,] {ABSolute | PERCent | HYSTeresis}`

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

:MEASure:THResholds:PERCent

Command `:MEASure:THResholds:PERCent <source>,<upper_pct>,<middle_pct>,<lower_pct>`

The :MEASure:THResholds:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

NOTE This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:PERCent" on page 660
- ":MEASure:THResholds:RFALl:PERCent" on page 678
- ":MEASure:THResholds:SERial:PERCent" on page 690

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_pct>
<middle_pct>
<lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:PERCent CHANnel2,100,50,0"
```

Query `:MEASure:THResholds:PERCent? <source>`

The :MEASure:THResholds:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:PERCent] <upper_pct>,<middle_pcts>,<lower_pct><NL>
```

Example

This example returns the upper level, middle level, and lower level percentages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:THResholds:RFALl:ABSolute

Command `:MEASure:THResholds:RFALl:ABSolute <source>,<upper_volts>,<middle_volts>,<lower_volts>`

The :MEASure:THResholds:RFALl:ABSolute command sets the upper level, middle level, and lower level voltages that are used to calculate the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_volts>
<middle_volts>
<lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:ABSolute CHANnel2,0.9,0.5
,0.1"
```

Query `:MEASure:THResholds:RFALl:ABSolute? <source>`

The :MEASure:THResholds:RFALl:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:RFALl:ABSolute] <upper_volts>,<middle_volts>,<lower_volts><NL>
```

Example

This example returns the upper level, middle level, and lower level voltages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 652
- ":MEASure:THResholds:GENeral:ABSolute" on page 654
- ":MEASure:THResholds:SERial:ABSolute" on page 684

:MEASure:THResholds:RFALl:HYSTeresis

Command `:MEASure:THResholds:RFALl:HYSTeresis <source>,<range>,<level>`

The :MEASure:THResholds:RFALl:HYSTeresis command sets the range and level voltages that are used to calculate the measurements that use them. The range is added to the level to determine the upper level voltage for measurements that use it. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:HYSTeresis CHANNEL2,0.9,0
.1"
```

Query `:MEASure:THResholds:RFALl:HYSTeresis? <source>`

The :MEASure:THResholds:RFALl:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:RFALl:HYSTeresis]<range>,<level><NL>
```

Example

This example returns the range and level voltages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:HYSTeresis? CHANNEL1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:HYSTeresis" on page 666
- ":MEASure:THResholds:GENeral:HYSTeresis" on page 656
- ":MEASure:THResholds:SERial:HYSTeresis" on page 686

:MEASure:THResholds:RFALl:METHod

Command `:MEASure:THResholds:RFALl:METHod <source>,{ABSolute | PERCent | HYSTeresis}`

The :MEASure:THResholds:RFALl:METHod command determines the way that the top and base of a waveform are calculated for all of the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:METHod CHANnel1,HYSTeresis"
```

Query `:MEASure:THResholds:RFALl:METHod? <source>`

The :MEASure:THResholds:RFALl:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:RFALl:METHod <source>,] {ABSolute | PERCent | HYSTeresis}`

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 668
- ":MEASure:THResholds:GENeral:METHod" on page 658
- ":MEASure:THResholds:SERial:METHod" on page 688

:MEASure:THResholds:RFALl:PERCent

Command `:MEASure:THResholds:RFALl:PERCent <source>,<upper_pct>,<middle_pct>,<lower_pct>`

The :MEASure:THResholds:RFALl:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used to calculate the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_pct>
<middle_pct>
<lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:PERCent CHANnel2,100,50,0
"
```

Query `:MEASure:THResholds:RFALl:PERCent? <source>`

The :MEASure:THResholds:RFALl:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:RFALl:PERCent] <upper_pct>,<middle_pcts>,<lower_pct
><NL>
```

Example

This example returns the upper level, middle level, and lower level percentages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 670
- ":MEASure:THResholds:GENeral:PERCent" on page 660
- ":MEASure:THResholds:SERial:PERCent" on page 690

:MEASure:THResholds:RFALl:TOPBase:ABSolute

Command `:MEASure:TOPBase:THResholds:RFALl:ABSolute <source>,<top_volts>,<base_volts>`

The :MEASure:TOPBase:THResholds:RFALl:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<top_volts>
<base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:ABSolute CHANnel2
,0.9,0.1"
```

Query `:MEASure:THResholds:RFALl:TOPBase:ABSolute? <source>`

The :MEASure:THResholds:RFALl:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format `[:MEASure:THResholds:RFALl:TOPBase:ABSolute] <top_volts>,<base_volts><NL>`

Example This example returns the top level and base level voltages used to calculate the rise/fall measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:ABSolute? CHANnel
1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 696
- ":MEASure:THResholds:GENeral:TOPBase:ABSolute" on page 662
- ":MEASure:THResholds:SERial:TOPBase:ABSolute" on page 692

:MEASure:THResholds:RFALl:TOPBase:METHod

Command `:MEASure:THResholds:RFALl:TOPBase:METHod <source>,{ABSolute | HISTONLY | MINmax | STANdard}`

The :MEASure:THResholds:RFALl:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

NOTE These threshold settings are used for rise/fall measurements.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:METHod CHANnel1,HISTONLY"
```

Query `:MEASure:THResholds:RFALl:TOPBase:METHod?`

The :MEASure:THResholds:RFALl:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:RFALl:TOPBase:METHod] {ABSolute | HISTONLY | MINmax | STANdard}`

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:RFALl:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 698
- ":MEASure:THResholds:GENeral:TOPBase:METHod" on page 664
- ":MEASure:THResholds:SERial:TOPBase:METHod" on page 694

:MEASure:THResholds:SERial:ABSolute

Command `:MEASure:THResholds:SERial:ABSolute <source>,<upper_volts>,<middle_volts>,<lower_volts>`

The :MEASure:THResholds:SERial:ABSolute command sets the upper level, middle level, and lower level voltages that are used for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_volts>
<middle_volts>
<lower_volts> A real number specifying voltage thresholds.

Example This example sets the custom voltage thresholds to 0.9 volts for the upper level, 0.5 volts for the middle level and 0.1 volts for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:ABSolute CHANnel2,0.9,0.5,0.1"
```

Query `:MEASure:THResholds:SERial:ABSolute? <source>`

The :MEASure:THResholds:SERial:ABSolute? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:SERial:ABSolute] <upper_volts>,<middle_volts>,<lowe
r_volts><NL>
```

Example

This example returns the upper level, middle level, and lower level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:ABSolute? CHANnel1"
strThresholds = myScope.ReadString
Debug.Print strThresholds
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:ABSolute" on page 652
- ":MEASure:THResholds:GENeral:ABSolute" on page 654
- ":MEASure:THResholds:RFALl:ABSolute" on page 672

:MEASure:THResholds:SERial:HYSTeresis

Command `:MEASure:THResholds:SERial:HYSTeresis <source>,<range>,<level>`

The :MEASure:THResholds:SERial:HYSTeresis command sets the range and level voltages that are used for protocol decode. The range is added to the level to determine the upper level voltage. The range is subtracted from the level to determine the lower level voltage. The level is the middle level voltage.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<range> A real number specifying voltage range for the hysteresis around the level value.

<level> A real number specifying voltage level.

Example This example sets the hysteresis range to 0.9 volts and 0.1 volts for the level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:HYSTeresis CHANNEL2,0.9,
0.1"
```

Query `:MEASure:THResholds:SERial:HYSTeresis? <source>`

The :MEASure:THResholds:SERial:HYSTeresis? query returns the current settings for upper level, middle level, and lower level voltages for the custom thresholds.

Returned Format

```
[:MEASure:THResholds:SERial:HYSTeresis]<range>,<level><NL>
```

Example

This example returns the range and level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:HYSTeresis? CHANNEL1"
strRangeLevel = myScope.ReadString
Debug.Print strRangeLevel
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:HYSTeresis" on page 666
- ":MEASure:THResholds:GENeral:HYSTeresis" on page 656
- ":MEASure:THResholds:RFALl:HYSTeresis" on page 674

:MEASure:THResholds:SERial:METHod

Command `:MEASure:THResholds:SERial:METHod <source>,{ABSolute | PERCent | HYSTeresis}`

The :MEASure:THResholds:SERial:METHod command determines the way that the top and base of a waveform are calculated for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:METHod CHANnel1,HYSTeresis"
```

Query `:MEASure:THResholds:SERial:METHod? <source>`

The :MEASure:THResholds:SERial:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:SERial:METHod <source>,] {ABSolute | PERCent | HYSTeresis}`

Example This example returns the method used to calculate the top and base of a waveform to hysteresis.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:METHod?"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:METHod" on page 668
- ":MEASure:THResholds:GENeral:METHod" on page 658
- ":MEASure:THResholds:RFALl:METHod" on page 676

:MEASure:THResholds:SERial:PERCent

Command `:MEASure:THResholds:SERial:PERCent <source>,<upper_pct>,<middle_pct>,<lower_pct>`

The :MEASure:THResholds:SERial:PERCent command sets the upper level, middle level, and lower level voltages as a percentage of the top and base voltages which are used for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<upper_pct>
<middle_pct>
<lower_pct> A real number specifying upper percentage from -24.8 to 125.0 A real number specifying the middle percentage from -24.9 to 124.9. A real number specifying the lower percentage from -25.0 to 125.8

Example This example sets the percentage to 100% for the upper level, 50% for the middle level and 0% for the lower level on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:PERCent CHANnel2,100,50,
0"
```

Query `:MEASure:THResholds:SERial:PERCent? <source>`

The :MEASure:THResholds:SERial:PERCent? query returns the current settings for upper level, middle level, and lower level percentages.

Returned Format

```
[:MEASure:THResholds:SERial:PERCent] <upper_pct>,<middle_pcts>,<lower_pct><NL>
```

Example

This example returns the upper level, middle level, and lower level percentages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:PERCent? CHANnel1"
strThresholdsPct = myScope.ReadString
Debug.Print strThresholdsPct
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:PERCent" on page 670
- ":MEASure:THResholds:GENeral:PERCent" on page 660
- ":MEASure:THResholds:RFALl:PERCent" on page 678

:MEASure:THResholds:SERial:TOPBase:ABSolute

Command `:MEASure:TOPBase:THResholds:SERial:ABSolute <source>,<top_volts>,<base_volts>`

The :MEASure:TOPBase:THResholds:SERial:ABSolute command sets the top level and base level voltages that are used for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<top_volts>
<base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:ABSolute CHANnel
2,0.9,0.1"
```

Query `:MEASure:THResholds:SERial:TOPBase:ABSolute? <source>`

The :MEASure:THResholds:SERial:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format `[:MEASure:THResholds:SERial:TOPBase:ABSolute] <top_volts>,<base_volts><NL>`

Example This example returns the top level and base level voltages used for protocol decode on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:ABSolute? CHANne
l1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

See Also

- ":MEASure:THResholds:TOPBase:ABSolute" on page 696
- ":MEASure:THResholds:GENeral:TOPBase:ABSolute" on page 662
- ":MEASure:THResholds:RFALl:TOPBase:ABSolute" on page 680

:MEASure:THResholds:SERial:TOPBase:METHod

Command `:MEASure:THResholds:SERial:TOPBase:METHod <source>,{ABSolute | HISTONLY | MINmax | STANdard}`

The :MEASure:THResholds:SERial:TOPBase:METHod command determines the way that the top and base of a waveform are derived for protocol decode.

NOTE These serial threshold settings are used for protocol decode.

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:METHod CHANnel1,
HISTONLY"
```

Query `:MEASure:THResholds:SERial:TOPBase:METHod?`

The :MEASure:THResholds:SERial:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format `[:MEASure:THResholds:SERial:TOPBase:METHod] {ABSolute | HISTONLY | MINmax | STANdard}`

Example This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:SERial:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

See Also

- ":MEASure:THResholds:TOPBase:METHod" on page 698
- ":MEASure:THResholds:GENeral:TOPBase:METHod" on page 664
- ":MEASure:THResholds:RFALl:TOPBase:METHod" on page 682

:MEASure:THResholds:TOPBase:ABSolute

Command `:MEASure:TOPBase:THResholds:ABSolute <source>,<top_volts>,<base_volts>`

The :MEASure:TOPBase:THResholds:ABSolute command sets the top level and base level voltages that are used to calculate the measurements that use them.

NOTE This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:TOPBase:ABSolute" on page 662
- ":MEASure:THResholds:RFALl:TOPBase:ABSolute" on page 680
- ":MEASure:THResholds:SERial:TOPBase:ABSolute" on page 692

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

<top_volts>
<base_volts> A real number specifying voltage levels. The top voltage level must be greater than the base voltage level.

Example This example sets the voltage level for the top to 0.9 volts and the voltage level for the base to 0.1 volts on channel 2.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:ABSolute CHANnel2,0.9,0
.1"
```

Query `:MEASure:THResholds:TOPBase:ABSolute? <source>`

The :MEASure:THResholds:TOPBase:ABSolute? query returns the current settings for top level and base level voltages.

Returned Format

```
[:MEASure:THResholds:TOPBase:ABSolute] <top_volts>,<base_volts><NL>
```

Example

This example returns the top level and base level voltages used to calculate the measurements on channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:ABSolute? CHANnel1"
strTopBase = myScope.ReadString
Debug.Print strTopBase
```

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

:MEASure:THResholds:TOPBase:METHod

Command `:MEASure:THResholds:TOPBase:METHod <source>,{ABSolute | HISTONLY | MINmax | STANdard}`

The :MEASure:THResholds:TOPBase:METHod command determines the way that the top and base of a waveform are derived for all of the measurements that use them.

NOTE This command changes the threshold settings used for rise/fall time measurements, protocol decode, and all other general-purpose measurements that use thresholds. To change the settings used for these types of measurements individually, see:

- ":MEASure:THResholds:GENeral:TOPBase:METHod" on page 664
- ":MEASure:THResholds:RFALl:TOPBase:METHod" on page 682
- ":MEASure:THResholds:SERial:TOPBase:METHod" on page 694

<source> {ALL | CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source. Setting the source to ALL does not affect the individual channel settings which is the behavior as the user interface.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example sets the method used to derive the top and base of a waveform to the histogram method.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:METHod CHANnel1,HISTONLY"
```

Query `:MEASure:THResholds:TOPBase:METHod?`

The :MEASure:THResholds:TOPBase:METHod? query returns the current method being used to calculate the top and base of a waveform.

Returned Format

```
[:MEASure:THResholds:TOPBase:METHod] {ABSolute | HISTONLY | MINmax | STANdard}
```

Example

This example returns the method used to derive the top and base of a waveform for channel 1.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:THResholds:TOPBase:METHod CHANnel1"
varMethod = myScope.ReadNumber
Debug.Print FormatNumber(varMethod, 0)
```

:MEASure:TIEClock2

Command

NOTE

Turn Off Headers

When receiving numeric data into numeric variables, turn off the headers. Otherwise, the headers may cause misinterpretation of returned data.

NOTE

This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

```
:MEASure:TIEClock2 <source>,{SECond | UNITinterval},
<direction>,{AUTO | CUSTOM,<frequency>} | {VARiable,<frequency>,<bandwid
th>} | CLOCk}
```

The :MEASure:TIEClock2 command measures time interval error on a clock. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval. If AUTO is selected, the oscilloscope selects the ideal constant clock frequency. If CUSTom is selected, you can enter your own ideal clock frequency. If VARiable is selected, a first order PLL clock recovery is used at the give clock frequency and loop bandwidth. If CLOCk is given, clock recovery is specified with the :MEASure:CLOCk:METHod command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is:

An integer, 1-2, for two channel Infiniium Oscilloscope. An integer, 1-4, for all other Infiniium Oscilloscope models.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> {RISing | FALLing | BOTH}

Specifies direction of clock edge. BOTH selects the first edge from the left-hand side of the waveform viewing area.

<frequency> A real number for the ideal clock frequency for clock recovery.

<bandwidth> A real number for the loop bandwidth of the PLL clock recovery method.

Example This example measures the clock time interval error on the rising edge of channel 1, ideal clock frequency set to automatic, units set to seconds.

```
myScope.WriteString ":MEASURE:TIECLOCK2 CHANNEL1,SECOND,RISING,AUTO"
```

Query

```
:MEASure:TIEClock2? <source>,{SECond | UNITinterval},<direction>,{AUTO |
 CUSTOM,<frequency> | {VARiable,<frequency>,<bandwidth>} | CLOCk}
```

The :MEASure:TIEClock2? query returns the current value of the clock time interval error.

Returned Format

```
[:MEASure:TIEClock2] <value>[,<result_state>]<NL>
```

<value> The clock time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of the clock time interval error in the variable strValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:TIECLOCK2? CHANNEL1,SECOND,FALLING,CUSTOM,
2.5E9"
strValue = myScope.ReadString
Debug.Print strValue
```

:MEASure:TIEData

Command

NOTE This command is only available when the E2681A Jitter Analysis Software, Serial Data Analysis, or the N5400A/5401A Software is installed.

```
:MEASure:TIEData <source>,{SECond | UNITinterval}, {AUTO | CUSTOM,<data_
rate> | VARiable,<data_rate>,<bandwidth> | CLOCk}
```

The :MEASure:TIEData command measures data time interval error. You can set the units of the measurement by selecting SECond (seconds) or UNITinterval. If AUTO is selected, the oscilloscope selects the ideal data rate. If CUSTom is selected, you can enter your own ideal constant data rate. If VARiable is selected, a first order PLL clock recovery is used at a given data rate and loop bandwidth. If CLOCk is given, clock recovery as specified with the :MEASure:CLOCk:METHod is used.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is:

An integer, 1-2, for two channel Infiniium Oscilloscope. An integer, 1-4, for all other Infiniium Oscilloscope models.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number for the ideal data rate for clock recovery.

<bandwidth> A real number for the loop bandwidth of the PLL clock recovery method.

Example This example measures the data time interval error on channel 1, ideal data rate set to automatic, units set to seconds.

```
myScope.WriteString ":MEASURE:TIEDATA CHANNEL1,SECOND,AUTO"
```

Query

```
:MEASure:TIEData? <source>,(SECond | UNITinterval},
{AUTO | CUSTom,<frequency> | VARiable,<frequency>,<bandwidth> | CLOCk}
```

The :MEASure:TIEData? query returns the current value of the data time interval error.

Returned Format `[:MEASure:TIEData] <value>[,<result_state>]<NL>`

<value> The data time interval error value.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current value of the data time interval error in the variable strValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:TIEDATA? CHANNEL1,SECOND,CUSTOM,1E9"
strValue = myScope.ReadString
Debug.Print strValue
```

:MEASure:TIEFilter:SHAPe

Command `:MEASure:TIEFilter:SHAPe {RECTangular | DB20 | DB40}`

The :MEASure:TIEFilter:SHAPe command specifies the shape of the TIE filter edge(s):

- RECTangular — The TIE filter is a brickwall filter.
- DB20 — The TIE filter edge(s) roll off at 20 dB per decade.
- DB40 — The TIE filter edge(s) roll off at 40 dB per decade.

Example This example specifies that the TIE filter edge(s) roll off at 40 dB per decade.

```
myScope.WriteString ":MEASURE:TIEFilter:SHAPe DB40"
```

Query `:MEASure:TIEFilter:SHAPe?`

The :MEASure:TIEFilter:SHAPe? query returns the specified shape of the TIE filter edge(s).

Returned Format `[:MEASure:TIEFilter:SHAPe] {RECTangular | DB20 | DB40}<NL>`

Example This example places the specified shape of the TIE filter edge(s) in the string variable, strShape, then prints the contents of the variable to the computer's screen.

```
Dim strShape As String   ' Dimension variable.
myScope.WriteString ":MEASure:TIEFilter:SHAPe?"
strShape = myScope.ReadString
Debug.Print strShape
```

See Also

- ":MEASure:TIEFilter:STATe" on page 706
- ":MEASure:TIEFilter:TYPE" on page 708
- ":MEASure:TIEFilter:STARt" on page 705
- ":MEASure:TIEFilter:STOP" on page 707

:MEASure:TIEFilter:STARt

Command `:MEASure:TIEFilter:STARt <start_frequency>`

The :MEASure:TIEFilter:STARt command sets the starting frequency for the TIE filter.

<start_frequency> A real number.

Query `:MEASure:TIEFilter:STARt?`

The :MEASure:TIEFilter:STARt? query returns the current value of the starting frequency of the TIE filter.

Returned Format `[:MEASure:TIEFilter:STARt] <value><NL>`

<value> The start frequency for the TIE filter.

Example This example returns the current value of the starting frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STARt?"
varStart = myScope.ReadNumber
Debug.Print FormatNumber(varStart, 0)
```

See Also

- ":MEASure:TIEFilter:STATe" on page 706
- ":MEASure:TIEFilter:TYPE" on page 708
- ":MEASure:TIEFilter:SHAPe" on page 704
- ":MEASure:TIEFilter:STOP" on page 707

:MEASure:TIEFilter:STATe

Command `:MEASure:TIEFilter:STATe {{ON | 1} | {OFF | 0}}`

The :MEASure:TIEFilter:STATe command enables the TIE filter for TIE data measurements.

Query `:MEASure:TIEFilter:STATe?`

The :MEASure:TIEFilter:STATe? query returns the current state of the TIE data filter.

Returned Format `[:MEASure:TIEFilter:STATe] {0 | 1}<NL>`

Example This example returns the current state of the TIE data filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STATe?"
varState = myScope.ReadNumber
Debug.Print FormatNumber(varState, 0)
```

See Also

- ":MEASure:TIEFilter:TYPE" on page 708
- ":MEASure:TIEFilter:SHAPe" on page 704
- ":MEASure:TIEFilter:STARt" on page 705
- ":MEASure:TIEFilter:STOP" on page 707

:MEASure:TIEFilter:STOP

Command `:MEASure:TIEFilter:STOP <stop_frequency>`

The :MEASure:TIEFilter:STOP command sets the stopping frequency for the TIE filter.

<stop_ frequency> A real number.

Query `:MEASure:TIEFilter:STOP?`

The :MEASure:TIEFilter:STOP? query returns the current value of the stopping frequency of the TIE filter.

Returned Format [:MEASure:TIEFilter:STOP] <value><NL>

<value> The stop frequency for the TIE filter.

Example This example returns the current value of the stopping frequency for the TIE filter then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":MEASure:TIEFilter:STOP?"
varStop = myScope.ReadNumber
Debug.Print FormatNumber(varStop, 0)
```

See Also

- ":MEASure:TIEFilter:STATe" on page 706
- ":MEASure:TIEFilter:TYPE" on page 708
- ":MEASure:TIEFilter:SHAPe" on page 704
- ":MEASure:TIEFilter:STARt" on page 705

:MEASure:TIEFilter:TYPE

Command `:MEASure:TIEFilter:TYPE {BANDpass | LOWPass | HIGHpass}`

The :MEASure:TIEFilter:TYPE command sets the type of TIE filter to be used.

Example This example sets the TIE filter to highpass.

```
myScope.WriteString ":MEASURE:TIEFilter:TYPE HIGHpass"
```

Query `:MEASure:TIEFilter:TYPE?`

The :MEASure:TIEFilter:TYPE? query returns the current type of TIE filter being used.

Returned Format `[:MEASure:TIEFilter:TYPE] {BANDpass | LOWPass | HIGHpass}<NL>`

Example This example places the current mode for TIEFilter:TYPE in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":MEASURE:TIEFilter:TYPE?"
strMode = myScope.ReadString
Debug.Print strMode
```

See Also

- ":MEASure:TIEFilter:STATe" on page 706
- ":MEASure:TIEFilter:SHAPe" on page 704
- ":MEASure:TIEFilter:STARt" on page 705
- ":MEASure:TIEFilter:STOP" on page 707

:MEASure:TMAX

Command `:MEASure:TMAX [<source>]`

The :MEASure:TMAX command measures the first time at which the maximum voltage of the source waveform occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMAX command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:TMAX? [<source>]`

The :MEASure:TMAX? query returns the time at which the first maximum voltage occurred.

Returned Format `[:MEASure:TMAX] <time>[,<result_state>]<NL>`

<time> Time at which the first maximum voltage occurred or frequency where the maximum FFT amplitude occurred.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first maximum voltage occurred to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:TMAX? CHANNEL1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MEASure:TMIN

Command `:MEASure:TMIN [<source>]`

The :MEASure:TMIN command measures the time at which the first minimum voltage occurred. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TMIN command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:TMIN? [<source>]`

The :MEASure:TMIN? query returns the time at which the first minimum voltage occurred or the frequency where the minimum FFT amplitude occurred.

Returned Format `[:MEASure:TMIN] <time>[,<result_state>]<NL>`

<time> Time at which the first minimum voltage occurred.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time at which the first minimum voltage occurred to the numeric variable, varTime, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:TMIN? CHANNEL1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MEASure:TVOLt

Command `:MEASure:TVOLt <voltage>,[<slope>]<occurrence> [,<source>]`

The :MEASure:TVOLt command measures the time interval between the trigger event and the defined voltage level and transition. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:TVOLt command.

The TEDGe command can be used to get the time of edges.

Query `:MEASure:TVOLt? <voltage>,<slope><occurrence> [,<source>]`

The :MEASure:TVOLt? query returns the time interval between the trigger event and the specified voltage level and transition.

<voltage> Voltage level at which time will be measured.

<slope> The direction of the waveform change when the specified voltage is crossed - rising (+) or falling (-). If no +/- sign is present, + is assumed.

<occurrence> The number of the crossing to be reported (if one, the first crossing is reported; if two, the second crossing is reported, etc.). The desired crossing must be present on the display. Occurrences are counted with 1 being the first occurrence from the left of the display, and a maximum value of 65534.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Returned Format `[:MEASure:TVOLt] <time>[,<result_state>]<NL>`

<time> The time interval between the trigger event and the specified voltage level and transition.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the time interval between the trigger event and the transition through -0.250 Volts on the third rising occurrence of the source waveform to the numeric variable, varTime. The contents of the variable are then printed to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:TVOLT? -0.250,+3,CHANNEL1"
varTime = myScope.ReadNumber
Debug.Print FormatNumber(varTime, 0)
```

:MEASure:UITouijitter

Command `:MEASure:UITouijitter <source>, <N>`

The :MEASure:UITouijitter command measures the difference between two consecutive N-UI measurements. The measurement then moves over one unit interval and makes another measurement. When N=1, this is analogous to cycle-cycle jitter, but measures unit intervals instead of periods. When N>1, this is analogous to N-Cycle jitter but measures unit intervals instead of periods.

<source> the source on which the measurement is made

<N> An integer greater than or equal to 1.

Example This example measures the UI-UI jitter for 3 consecutive unit intervals on channel 1.

```
myScope.WriteString ":MEASURE:UITouijitter CHAN1, 3"
```

Query `:MEASure:UITouijitter?`

:MEASure:UNITinterval

Command

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/5401A Software is installed.

```
:MEASure:UNITinterval <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval command measures the unit interval value of the selected source. Use the :MEASure:DATarate command/query to measure the data rate of the source

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<data_rate> A real number representing the data rate.

Example This example measures the unit interval of channel 1.

```
myScope.WriteString ":MEASURE:UNITINTERVAL CHANNEL1"
```

Query

```
:MEASure:UNITinterval? <source>[,{AUTO | (SEMI,<data_rate>)}]
```

The :MEASure:UNITinterval? query returns the measured unit interval.

Returned Format

```
[:MEASure:UNITinterval] <value>[,<result_state>]<NL>
```

<value> Unit interval of the source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current unit interval of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:UNITINTERVAL? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:VAMPlitude

Command `:MEASure:VAMPlitude [<source>]`

The :MEASure:VAMPlitude command calculates the difference between the top and base voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAMPlitude command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example calculates the difference between the top and base voltage of the specified source.

```
myScope.WriteString ":MEASURE:VAMPLITUDE CHANNEL1"
```

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

The :MEASure:VAMPlitude? query returns the calculated difference between the top and base voltage of the specified source.

Returned Format `[:MEASure:VAMPlitude] <value>[,<result_state>]<NL>`

<value> Calculated difference between the top and base voltage.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current Vamplitude value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VAMPLITUDE? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:VAVerage

Command `:MEASure:VAVerage {CYCLe | DISPlay}[,<source>]`

The :MEASure:VAVerage command calculates the average voltage over the displayed waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

CYCLe The CYCLe parameter instructs the average measurement to measure the average voltage across the first period on the display.

DISPlay The DISPlay parameter instructs the average measurement to measure all the data on the display.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example calculates the average voltage over the displayed waveform.

```
myScope.WriteString ":MEASURE:VAVERAGE DISPLAY,CHANNEL1"
```

Query `:MEASure:VAVerage? {CYCLe | DISPlay}[,<source>]`

The :MEASure:VAVerage? query returns the calculated average voltage of the specified source. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VAVerage command.

Returned Format `[:MEASure:VAVerage] <value>[,<result_state>]<NL>`

<value> The calculated average voltage.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current average voltage in the numeric variable, varAverage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VAVERAGE? DISPLAY,CHANNEL1 CHANNEL1"
varAverage = myScope.ReadNumber
Debug.Print FormatNumber(varAverage, 0)
```

:MEASure:VBASe

Command `:MEASure:VBASe [<source>]`

The :MEASure:VBASe command measures the statistical base of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VBASe command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the voltage at the base of the waveform.

```
myScope.WriteString ":MEASURE:VBASE CHANNEL1"
```

Query `:MEASure:VBASe? [<source>]`

The :MEASure:VBASe? query returns the measured voltage value at the base of the specified source.

Returned Format `[:MEASure:VBASe] <value>[,<result_state>]<NL>`

<value> Voltage at the base of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the current voltage at the base of the waveform to the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VBASE? CHANNEL1"
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

:MEASure:VLOWer

Command `:MEASure:VLOWer [<source>]`

The :MEASure:VLOWer command measures the voltage value at the lower threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VLOWer command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Query `:MEASure:VLOWer?`

The :MEASure:VLOWer? query returns the measured lower threshold of the selected source.

Returned Format `[:MEASure:VLOWer] <value>[,<result_state>]<NL>`

<value> Voltage value at the lower threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured voltage at the lower threshold of the waveform to the numeric variable, varVlower, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VLOW? CHANNEL1"
varVlower = myScope.ReadNumber
Debug.Print FormatNumber(varVlower, 0)
```

:MEASure:VMAX

Command `:MEASure:VMAX [<source>]`

The :MEASure:VMAX command measures the absolute maximum voltage present on the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMAX command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the absolute maximum voltage on the waveform.

```
myScope.WriteString ":MEASURE:VMAX CHANNEL1"
```

Query `:MEASure:VMAX? [<source>]`

The :MEASure:VMAX? query returns the measured absolute maximum voltage or maximum FFT amplitude present on the selected source waveform.

Returned Format `[:MEASure:VMAX] <value>[,<result_state>]<NL>`

<value> Absolute maximum voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute maximum voltage on the waveform to the numeric variable, varMaximum, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VMAX? CHANNEL1"
```

```
varMaximum = myScope.ReadNumber
Debug.Print FormatNumber(varMaximum, 0)
```

:MEASure:VMIDdle

Command `:MEASure:VMIDdle [<source>]`

The :MEASure:VMIDdle command measures the voltage level at the middle threshold of the waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VMIDdle command.

Query `:MEASure:VMIDdle? [<source>]`

The :MEASure:VMIDdle? query returns the voltage value at the middle threshold of the waveform.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Returned Format `[MEASure:VMIDdle] <value>[,<result_state>]<NL>`

<value> The middle voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured middle voltage on the waveform to the numeric variable, varMiddle, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VMID? CHANNEL1"
varMiddle = myScope.ReadNumber
Debug.Print FormatNumber(varMiddle, 0)
```

:MEASure:VMIN

Command `:MEASure:VMIN [<source>]`

The :MEASure:VMIN command measures the absolute minimum voltage present on the selected source waveform. Sources are specified with :MEASure:SOURce or with the optional parameter following the :MEASure:VMIN command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the absolute minimum voltage on the waveform.

```
myScope.WriteString ":MEASURE:VMIN CHANNEL1"
```

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

The :MEASure:VMIN? query returns the measured absolute minimum voltage or minimum FFT amplitude present on the selected source waveform.

Returned Format `[:MEASure:VMIN] <value>[,<result_state>]<NL>`

<value> Absolute minimum voltage present on the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example returns the measured absolute minimum voltage on the waveform to the numeric variable, varMinimum, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VMIN? CHANNEL1"
```

```
varMinimum = myScope.ReadNumber
Debug.Print FormatNumber(varMinimum, 0)
```

:MEASure:VOVershoot

Command `:MEASure:VOVershoot <source>[,<direction>]`

The :MEASure:VOVershoot command is similar to the overshoot measurement, but instead of returning the ratio of overshoot voltage to amplitude as a percent, it returns the local voltage of the overshoot.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> RISing or FALLing

Example This example measures the local voltage of the overshoot.

```
myScope.WriteString ":MEASURE:VOVershoot CHAN1"
```

Query `:MEASure:VOVershoot? <source>[,<direction>]`

The :MEASure:VOVershoot? query returns the local voltage of the overshoot.

:MEASure:VPP

Command `:MEASure:VPP [<source>]`

The :MEASure:VPP command measures the maximum and minimum voltages on the selected source, then calculates the peak-to-peak voltage as the difference between the two voltages. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VPP command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the peak-to-peak voltage or FFT amplitude range of the previously selected source.

```
myScope.WriteString ":MEASURE:VPP CHANNEL1"
```

Query `:MEASure:VPP? [<source>]`

The :MEASure:VPP? query returns the specified source peak-to-peak voltage.

Returned Format `[:MEASure:VPP] <value>[,<result_state>]<NL>`

<value> Peak-to-peak voltage of the selected source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current peak-to-peak voltage in the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VPP? CHANNEL1"
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

:MEASure:VPReshoot

Command `:MEASure:VPReshoot <source>[,<direction>]`

The :MEASure:VPReshoot command is similar to the preshoot measurement, but instead of returning the ratio of preshoot voltage to amplitude as a percent, it returns the local voltage of the preshoot.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<direction> RISing or FALLing

Example This example measures the local voltage of the preshoot.

```
myScope.WriteString ":MEASURE:VPReshoot CHAN1"
```

Query `:MEASure:VPReshoot? <source>[,<direction>]`

:MEASure:VRMS

Command `:MEASure:VRMS {CYCLe | DISPlay},{AC | DC} [,<source> [,{VOLT | DBM}]]`

The :MEASure:VRMS command measures the RMS voltage of the selected waveform by subtracting the average value of the waveform from each data point on the display. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VRMS command.

CYCLe The CYCLe parameter instructs the RMS measurement to measure the RMS voltage across the first period of the display.

DISPlay The DISPlay parameter instructs the RMS measurement to measure all the data on the display. Generally, RMS voltage is measured across one waveform or cycle, however, measuring multiple cycles may be accomplished with the DISPlay option. The DISPlay parameter is also useful when measuring noise.

AC The AC parameter is used to measure the RMS voltage subtracting the DC component.

DC The DC parameter is used to measure RMS voltage including the DC component.

The AC RMS, DC RMS, and VAVG parameters are related as in this formula:

$DCVRMS^2=ACVRMS^2+VAVG^2$

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

VOLT | DBM Specifies the units of the measurement as either volts or decibels.

Example This example measures the RMS voltage of the previously selected waveform.

```
myScope.WriteString ":MEASURE:VRMS CYCLE,AC,CHANNEL1"
```

Query `:MEASure:VRMS? {CYCLe | DISPlay},{AC | DC} [,<source> [,{VOLT | DBM}]]`

The :MEASure:VRMS? query returns the RMS voltage of the specified source.

Returned Format `[:MEASure:VRMS] <value>[,<result_state>]<NL>`

<value> RMS voltage of the selected waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the current AC RMS voltage over one period of the waveform in the numeric variable, varVoltage, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VRMS? CYCLE,AC,CHANNEL1"
varVoltage = myScope.ReadNumber
Debug.Print FormatNumber(varVoltage, 0)
```

:MEASure:VTIMe

Command `:MEASure:VTIMe <time>[,<source>]`

The :MEASure:VTIMe command measures the voltage at the specified time. The time is referenced to the trigger event and must be on the screen. When an FFT function is the specified source, the amplitude at the specified frequency is measured. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTIMe command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<time> A real number for time from trigger in seconds, or frequency in Hertz for an FFT (when a function is set to FFT or a waveform memory contains an FFT).

Query `:MEASure:VTIMe? <time>[,<source>]`

The :MEASure:VTIMe? query returns the measured voltage or amplitude.

Returned Format `[:MEASure:VTIMe] <value>[,<result_state>]<NL>`

<value> Voltage at the specified time. When the source is an FFT function, the returned value is the vertical value at the horizontal setting passed in the VTIMe <time> parameter. The time parameter is in Hertz when an FFT function is the source.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the voltage at 500 ms in the numeric variable, varValue, then prints the contents to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VTIME? 500E-3,CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:VTOP

Command `:MEASure:VTOP [<source>]`

The :MEASure:VTOP command measures the statistical top of the selected source waveform. Sources are specified with the :MEASure:SOURce command or with the optional parameter following the :MEASure:VTOP command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the voltage at the top of the waveform.

```
myScope.WriteString ":MEASURE:VTOP CHANNEL1"
```

Query `:MEASure:VTOP? [<source>]`

The :MEASure:VTOP? query returns the measured voltage at the top of the specified source.

Returned Format `[:MEASure:VTOP] <value>[,<result_state>]<NL>`

<value> Voltage at the top of the waveform.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the top of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VTOP? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:VUPPer

Command `:MEASure:VUPPer [<source>]`

The :MEASure:VUPPer command measures the voltage value at the upper threshold of the waveform. Sources are specified with the MEASure:SOURce command or with the optional parameter following the :MEASure:VUPPer command.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the voltage at the upper threshold of the waveform.

```
myScope.WriteString ":MEASURE:VUPPer CHANNEL1"
```

Query `:MEASure:VUPPer? [<source>]`

The :MEASure:VUPPer? query returns the measured upper threshold value of the selected source.

Returned Format `[:MEASure:VUPPer] <value>[,<result_state>]<NL>`

<value> Voltage at the upper threshold.

<result_state> If SENDvalid is ON, the result state is returned with the measurement result. See the :MEASure:RESults table in this chapter for a list of the result states.

Example This example places the value of the voltage at the upper threshold of the waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:VUPPER? CHANNEL1"
```

```
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:MEASure:WINDow

Command `:MEASure:WINDow {ZOOM | {MAIN | ALL}}, {MEASN}`

The :MEASure:WINDow command specifies whether measurements are made in the ZOOM window (measurement gating) or over the entire acquisition (MAIN or ALL). The MAIN and ALL parameters are equivalent.

Example This example gates Measurement 1 to the zoom window.

```
myScope.WriteString ":MEASURE:WINDow ZOOM, MEAS1"
```

Query :MEASure:WINDow? {MEASN}

This query returns whether the measurement is being performed on the zoomed portion of the waveform (ZOOM) or the entire acquisition (MAIN or ALL).

:MEASurement<N>:NAME

Command `:MEASurement<N>:NAME <name>`

The :MEASurement<N>:NAME commands sets the name of the specified measurement to whatever string is given to <name>. This lets you give specific names to measurements displayed on the oscilloscope's screen.

<N> An integer, 1-10. This number represents the position of the measurement on screen in the Measurements tab.

<name> A quoted string.

Query `:MEASurement<N>:NAME?`

The :MEASurement<N>:NAME? query returns the name of the corresponding measurement.

:MEASurement<N>:SOURce

Command `:MEASurement<N>:SOURce <source>[,<source>]`

The :MEASurement<N>:SOURce command changes the source of an existing measurement in the Measurements tab of the user interface.

<N> An integer, 1-10. This number represents the position of the measurement on screen in the Measurements tab.

<source> {CHANnel<N> | FUNCtion<F> | DIGital<M> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:MEASurement<N>:SOURce?`

The :MEASurement<N>:SOURce? query returns the source(s) of the selected measurement.

Returned Format `[:MEASurement<N>:SOURce] <source>[,<source>]<NL>`

Example This example places the currently specified measurement 1 source(s) in the string variable, strSource, then prints the contents of the variable to the computer's screen.

```
Dim strSource As String   ' Dimension variable.
myScope.WriteString ":MEASurement1:SOURce?"
strSource = myScope.ReadString
Debug.Print strSource
```

See Also

- ":MEASurement<N>:NAME" on page 738

26
Pod Commands

:POD<N>:DISPlay 742
:POD<N>:PSKew 743
:POD<N>:THReshold 744

NOTE The POD commands only apply to the MSO oscilloscopes.

:POD<N>:DISPlay

Command

NOTE The POD commands only apply to the MSO oscilloscopes.

```
:POD<N>[:DISPlay] {ON | OFF | 1 | 0}
```

The :POD<N>:DISPlay command enables or disables the view for the selected digital channel pod. Pod 1 has the digital channels 0 through 7, and pod 2 has the digital channels 8 through 15.

Displaying a pod automatically enables digital channels. See ENABle command in the root subsystem.

<N> An integer, 1-2.

Example This example turns on the display of bit 5 for the digital channels.

```
myScope.WriteString ":ENABle DIGital"
myScope.WriteString ":POD2:DISPlay ON"
```

Query

```
:POD<N>[:DISPlay]?
```

The :POD<N>:DISPlay? query returns the value of the display setting for the pod.

Returned Format

```
[:POD<N>:DISPlay] {1 | 0}<NL>
```

See Also

- ":DIGital<N>:DISPlay" on page 276
- ":ENABle DIGital" on page 762

:POD<N>:PSKew

Command `:POD<N>:PSKew <skew_value>`

The :POD<N>:PSKew command lets you adjust the digital channels with respect to the analog channels when there is a time delay between the analog and digital channels. This can occur when different length cables are used.

<N> An integer, 1-2.

<skew_value> A real number for the skew value, in the range -1 ms to +1 ms.

Example This example sets the probe skew for all digital channels to 10 µs.

```
myScope.WriteString ":POD1:PSKew 10E-6"
```

Query `:POD<N>:PSKew?`

The :POD<N>:PSKew? query returns the current probe skew setting for all digital channels.

Returned Format `[:POD<N>:PSKew] <skew_value><NL>`

:POD<N>:THReshold

Command

NOTE The POD commands only apply to the MSO oscilloscopes.

```
:POD<N>:THReshold {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFFerential | <value>}
```

The :POD<N>:THReshold command sets the logic threshold value for a pod. Pod 1 has the digital channels 0 through 7, and pod 2 has the digital channels 8 through 15. This command is equivalent to the :DIGital<N>:THReshold command.

The threshold is used for triggering purposes and for displaying the digital data as high (above the threshold) or low (below the threshold). The voltage values for the predefined thresholds are:

- CMOS50 = 2.5 V
- CMOS33 = 1.65 V
- CMOS25 = 1.25 V
- ECL = -1.3 V
- PECL = 3.7 V
- TTL = 1.4 V
- DIFFerential = 0 V

<N> An integer, 1-2.

<value> A real number representing the voltage value which distinguishes a 1 logic level from a 0 logic level. Waveform voltages greater than the threshold are 1 logic levels while waveform voltages less than the threshold are 0 logic levels.

The range of the threshold voltage is from -3.75 volts to 3.75 volts.

Example This example sets the threshold to 1.8 volts for pod 2 (digital channels D15 through D8).

```
myScope.WriteString ":POD2:THReshold 1.8"
```

Query

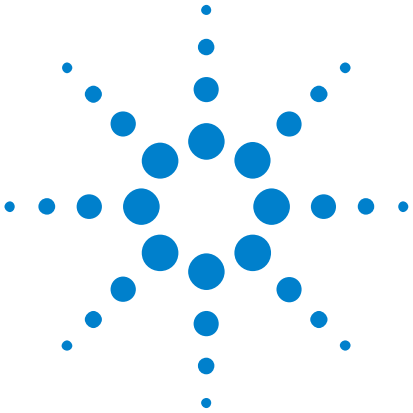
```
:POD<N>:THREShold?
```

The :POD<N>:THReshold? query returns the threshold value for the specified pod.

Returned Format

```
[:POD<N>:THReshold] {CMOS50 | CMOS33 | CMOS25 | ECL | PECL | TTL
   | DIFF | <value>}<NL>
```

See Also • ":DIGital<N>:THReshold" on page 279



27
Root Level Commands

:ADER? 749
:AER? 750
:ASTate? 751
:ATER? 752
:AUToscale 753
:AUToscale:CHANnels 754
:AUToscale:PLACement 755
:AUToscale:VERTical 756
:BEEP 757
:BLANk 758
:CDISplay 759
:DIGitize 760
:DISable DIGital 761
:ENABle DIGital 762
:MTEE 764
:MTER? 765
:MODel? 763
:OPEE 766
:OPER? 767
:OVLRegister? 768
:PDER? 769
:PRINt 770
:RECall:SETup 771
:RSTate? 772
:RUN 773
:SERial 774
:SINGle 775
:STATus? 776
:STOP 777
:STORe:JITTer 778
:STORe:SETup 779
:STORe:WAVeform 780
:TER? 781
:VIEW 782

Root level commands control many of the basic operations of the oscilloscope that you can select by pressing the labeled keys on the front panel. These commands are always recognized by the parser if they are prefixed with a colon, regardless of the current tree position. After executing a root level command, the parser is positioned at the root of the command tree.

:ADER?

(Acquisition Done Event Register)

Query `:ADER?`

The :ADER? query reads the Acquisition Done Event Register and returns 1 or 0. After the Acquisition Done Event Register is read, the register is cleared. The returned value 1 indicates an acquisition completed event has occurred and 0 indicates an acquisition completed event has not occurred.

Once the Done bit is set, it is cleared only by doing :ADER? or by sending a *CLS command.

Returned Format `{1 | 0}<NL>`

:AER?

(Arm Event Register)

Query `:AER?`

The :AER? query reads the Arm Event Register and returns 1 or 0. After the Arm Event Register is read, the register is cleared. The returned value 1 indicates a trigger armed event has occurred and 0 indicates a trigger armed has not occurred.

NOTE

Arm Event Returns

:AER? will allow the Arm Event to return either immediately (if you have armed but not triggered) or on the next arm (if you have already triggered). However, *CLS is always required to get an SRQ again.

Once the AER bit is set, it is cleared only by doing :AER? or by sending a *CLS command.

Returned Format `{1 | 0}<NL>`

:ASTate?

Query `:ASTate?`

The :ASTate? query returns the acquisition state:

- ARM — The trigger is armed and the oscilloscope has acquired all of the pre-trigger data.
- TRIG — The trigger condition has occurred and the oscilloscope is acquiring post trigger data.
- ATRIG — The trigger condition has not been met, but the oscilloscope has auto triggered and is acquiring post trigger data.
- ADONE — The acquisition is done, and the data has been processed and is ready to be unloaded.

Returned Format {ARM | TRIG | ATRIG | ADONE}<NL>

See Also
- ":RSTate?" on page 772

:ATER?

(Auto Trigger Event Register)

Query `:ATER?`

The :ATER? query reads the Auto Trigger Event Register and returns 1 or 0. After the Auto Trigger Event Register is read, the register is cleared. The returned value 1 indicates an auto trigger event has occurred and 0 indicates an auto trigger event has not occurred.

Returned Format `{1 | 0}<NL>`

:AUToscale

Command `:AUToscale`

The :AUToscale command causes the oscilloscope to evaluate all input waveforms and find the optimum conditions for displaying the waveform. It searches each of the channels for input waveforms and shuts off channels where no waveform is found. It adjusts the vertical gain and offset for each channel that has a waveform and sets the time base on the lowest numbered input channel that has a waveform.

The trigger is found by searching each channel, starting with channel 4, then channel 3, channel 2, and channel 1, until a trigger waveform is detected. If waveforms cannot be found on any vertical input, the oscilloscope is returned to its former state.

Autoscale sets the following:

- Channel Display, Scale, and Offset
- Trigger Sweep, Mode, Edge, Source, Level, Slope, Hysteresis, and Holdoff
- Acquisition Sampling Rate and Memory Depth
- Time Base Scale and Position
- Marker Mode Set to Measurement
- Resets Acquisition Completion Criteria to 90%

Autoscale turns off the following:

- Measurements on sources that are turned off
- Functions
- Windows
- Memories
- InfiniiSim

Autoscale does not turn off:

- PrecisionProbe/PrecisionCable

No other controls are affected by Autoscale.

Example This example automatically scales the oscilloscope for the input waveform.

```
myScope.WriteString ":AUToscale"
```

:AUToscale:CHANnels

Command `:AUToscale:CHANnels {ALL | DISPlayed}`

The :AUToscale:CHANnels command selects whether to apply autoscale to all of the input channels or just the input channels that are currently displayed.

Example This example automatically scales only the displayed channels.

```
myScope.WriteString ":AUTOSCALE:CHANnels DISPlayed"
```

:AUToscale:PLACement

Command `:AUToscale:PLACement {STACk | SEParate | OVERlay}`

The :AUToscale:PLACement command controls how the waveforms are displayed on the oscilloscope when the autoscale command is used. If Stack is chosen then each waveform's amplitude is decreased and then the waveforms are offset so each takes up a different vertical portion of the screen. This makes it easier to view them, but decreases the accuracy of any measurements performed on the waveforms because they no longer take up the full dynamic range of th ADC (analog to digital converter). If Separate is chosen then the screen is divided into the same number of grids that there are waveforms (for example, if three waveforms are displayed then the screen will be divided into three grids). Each grid represents the full dynamic range of the ADC so this choice maximizes measurement accuracy while still separating the waveforms so they are easy to see. If the Overlay option is chosen then the waveforms are displayed on top of each other. This maximizes measurement accuracy, but can making viewing difficult.

Example This example automatically overlays the waveforms after an autoscale.

```
myScope.WriteString ":AUTOSCALE:OVERlay ON"
```

Query `:AUToscale:PLACement?`

:AUToscale:VERTical

Command `:AUToscale:VERTical {CHANnel<N>}`

The :AUToscale:VERTical command autoscales the vertical position and scaling for the corresponding channel without changing anything else (for example, trigger or timebase settings).

Example This example automatically autoscales the vertical position and scale for the waveform on Channel 1.

```
myScope.WriteString ":AUTOSCALE:VERTical CHAN1"
```

NOTE If you are using software 2.10 or earlier, the command syntax is (lower-case "t" in "vertical"):

AUToscale:VERTical <CHANnel 1 | CHANnel 2 | CHANnel 3 | CHANnel 4>

:BEEP

Command `:BEEP <frequency>,<duration>`

The :BEEP command makes the oscilloscope beep at a defined frequency and duration.

<frequency> A real number representing frequency of beep in Hertz.

<duration> A real number representing duration of beep in milliseconds.

Example This example will create a beep at 1000 Hz for 500 ms.

```
myScope.WriteString ":BEEP 1000,500"
```

:BLANk

Command

```
:BLANk {CHANnel<N> | BUS<N> | DIGital<M> | FUNCtion<F> | HISTogram
   | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalize | ALL}
```

The :BLANk command turns off an active channel, function, histogram, waveform memory, measurement trend, measurement spectrum, or Feed-Forward Equalized waveform. The :VIEW command turns them on.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15.

Example This example turns off channel 1.

```
myScope.WriteString ":BLANk CHANnel1"
```

See Also

- ":VIEW" on page 782

:CDISplay

Command `: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 of the data in active channels and functions is erased; however, new data is displayed on the next acquisition. Waveform memories are not erased.

Example This example clears the oscilloscope display.

```
myScope.WriteString ":CDISPLAY"
```

:DIGitize

Command `:DIGitize [CHANnel<N> | DIGital<M> | POD<P>][,...]`

<N> An integer, 1-4.

<M> An integer, 0-15.

<P> An integer, 1-2.

The :DIGitize command invokes a special mode of data acquisition that is more efficient than using the :RUN command. This command initializes the selected channels, then acquires them according to the current oscilloscope settings. When all waveforms are completely acquired, the oscilloscope is stopped. The waveform completion criteria is set with the ":ACQuire:COMPlete" command.

If you specify channel parameters, then these are the only waveforms acquired and the display waveforms of the specified channels are turned off.

NOTE

Full Range of Measurement and Math Operators are Available

Even though digitized waveforms are not displayed, you may perform the full range of measurement and math operators on them.

If you use the :DIGitize command with no parameters, the digitize operation is performed on the channels that are being displayed in the Infiniium waveform viewing area. In this case, the display state of the acquired waveforms is not changed after the :DIGitize command is completed. Because the command executes more quickly without parameters, this form of the command is useful for repetitive measurement sequences. You can also use this mode if you want to view the digitize results because the display state of the digitized waveforms is not affected.

See the Chapter 38, “Sample Programs,” starting on page 1071 for examples of how to use :DIGitize and its related commands.

Example This example acquires data on channel 1.

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

The ACQuire subsystem commands set up conditions such as COUNt for the next :DIGitize command. The WAVeform subsystem commands determine how the data is transferred out of the oscilloscope, and how to interpret the data.

:DISable DIGital

Command `:DISable DIGital`

The :DISable DIGital command disables the digital channels 0-15.

Example This example will disable the digital channels.

```
myScope.WriteString ":DISable DIGital"
```

:ENABle DIGital

Command `:ENABle DIGital`

The :ENABle DIGital command enables the digital channels 0-15.

Example This example will enable the digital channels.

```
myScope.WriteString ":ENABle DIGital"
```

:MODel?

Query `:MODel?`

The :MODel? query returns the model number for the oscilloscope.

Returned Format A six-character alphanumeric model number in quotation marks. Output is determined by header and longform status as in Table 25.

Table 25 MODel? Returned Format

:SYSTem:HEADer		:SYSTem:LONGform		Response (for example)
ON	OFF	ON	OFF	
	X		X	DSO90804A
	X	X		DSO90804A
X			X	:MOD DSO90804A
X		X		:MODEL DSO90804A

Example This example places the model number in a string variable, strModel, then prints the contents of the variable on the computer's screen.

```
Dim strModel As String   ' Dimension variable.
myScope.WriteString ":MODel?"
strModel = myScope.ReadString
Debug.Print strModel
```

:MTEE

(Mask Test Enable Register)

Command `:MTEE <enable_mask>`

The :MTEE command is used to set bits in the Mask Test Enable Register.

<enable_mask> The <enable_mask> is a 16-bit signed decimal value that enables the following bits of the Mask Test Event Register:

Bit 0	Mask Test Complete
Bit 1	Mask Test Fail
Bit 2	Mask Low Amplitude
Bit 3	Mask High Amplitude
Bit 4	Mask Align Complete
Bit 5	Mask Align Fail
Bits 6-14	are not used

Query `:MTEE?`

The :MTEE? query returns the value stored in the Mask Test Enable Register.

Returned Format `[:MTEE] <enable_mask>`

Example Suppose your application requires an interrupt whenever a Mask Test Fail occurs in the mask test register. You can enable this bit to generate the summary bit by sending:

```
myScope.WriteString "MTEE 2"
```

Whenever an error occurs, the oscilloscope sets the MASK bit in the Operation Status Register. Because the bits in the Operation Status Enable Register are all enabled, a summary bit is generated to set bit 7 (OPER) in the Status Byte Register.

If bit 7 (OPER) in the Status Byte Register is enabled (via the *SRE command), a service request interrupt (SRQ) is sent to the external computer.

:MTER?

(Mask Test Event Register)

Query `:MTER?`

The :MTER? query returns the value stored in the Mask Test Event Register. The bits stored in the register have the following meanings:

Bit 0	Mask Test Complete bit is set whenever the mask test is complete.
Bit 1	Mask Test Fail bit is set whenever the mask test failed.
Bit 2	Mask Low Amplitude bit is set whenever the signal is below the mask amplitude.
Bit 3	Mask High Amplitude bit is set whenever the signal is above the mask amplitude.
Bit 4	Mask Align Complete bit is set whenever the mask align is complete.
Bit 5	Mask Align Fail bit is set whenever the mask align failed.

The Mask Test Event Register is read and cleared by the MTER? query. The register output is enabled or disabled using the mask value supplied with the MTEE command.

Returned Format `0-63 decimal value.`

NOTE

Disabled Mask Test Event Register Bits Respond, but Do Not Generate a Summary Bit

Mask Test Event 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 in the Operation Status Register.

:OPEE

(Operation Status Enable)

Command `:OPEE <mask>`

<mask> The decimal weight of the enabled bits.

The :OPEE command sets a mask in the Operation Status Enable register. Each bit that is set to a "1" enables that bit to set bit 7 in the status byte register, and potentially causes an SRQ to be generated. Bit 5, Wait for Trig is used. Other bits are reserved.

Query `:OPEE?`

The query returns the current value contained in the Operation Status Enable register as a decimal number.

Returned Format `[OPEE] <value><NL>`

:OPER?

(Operation Status Register)

Query `:OPER?`

The :OPER? query returns the value contained in the Operation Status Register as a decimal number. This register contains the WAIT TRIG bit (bit 5) and the OVLR bit (bit 11).

The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed. The OVLR bit is set by the Overload Event Register.

Returned Format `<value><NL>`

:OVLRegister?

Query `:OVLRegister?`

The :OVLRegister? query returns the value stored in the Overload Event Register.

The integer value returned by this query represents the channels as follows:

- Bit 0 - Channel 1
- Bit 1 - Channel 2
- Bit 2 - Channel 3
- Bit 3 - Channel 4
- Bits 7-4 are not used and are set to zero (0)

Returned Format `<value><NL>`

:PDER?

(Processing Done Event Register)

Query `:PDER?`

The :PDER? query reads the Processing Done Event Register and returns 1 or 0. After the Processing Done Event Register is read, the register is cleared. The returned value 1 indicates indicates that all math and measurements are complete and 0 indicates they are not complete. :PDER? is non-blocking.

:PDER? can be used in place of :ADER?.

Returned Format `{1 | 0}<NL>`

:PRINt

Command `:PRINt`

The :PRINt command outputs a copy of the screen to a printer or other device destination specified in the HARDcopy subsystem. You can specify the selection of the output and the printer using the HARDcopy subsystem commands.

Example This example outputs a copy of the screen to a printer or a disk file.

```
myScope.WriteString ":PRINT"
```

:RECall:SETup

Command `:RECall:SETup <setup_memory_num>`

<setup_memory_num> Setup memory number, an integer, 0 through 9.

The :RECall:SETup command recalls a setup that was saved in one of the oscilloscope's setup memories. You can save setups using either the :STORe:SETup command or the front panel.

Examples This command recalls a setup from setup memory 2.

```
myScope.WriteString ":RECall:SETup 2"
```

:RSTate?

Query `:RSTate?`

The :RSTate? query returns the run state:

- RUN — The oscilloscope is acquiring and displaying new waveforms.
- STOP — The oscilloscope is no longer acquiring new waveforms.
- SING — A single acquisition has been started and the oscilloscope is waiting for the trigger condition to be met.

These are the same run states displayed on the front panel and in the user interface.

Returned Format {RUN | STOP | SING}<NL>

See Also
- ":ASTate?" on page 751

:RUN

Command `:RUN`

The :RUN command starts the oscilloscope running. When the oscilloscope is running, it acquires waveform data according to its current settings. Acquisition runs repetitively until the oscilloscope receives a :STOP command, or until there is only one acquisition if Trigger Sweep is set to Single. However, the :TRIGger:SWEep SINGle should not be used in new programs. The :SINGle command should be used instead to acquire a single acquisition.

Example This example causes the oscilloscope to acquire data repetitively.

```
myScope.WriteString ":RUN"
```

:SERial

(Serial Number)

Command `:SERial {<serial_number>}`

The :SERial command sets the serial number of the oscilloscope. A serial number was entered in your oscilloscope by Agilent Technologies before it was shipped to you. Therefore, setting the serial number is not normally required unless the oscilloscope is serialized for a different application.

The oscilloscope's serial number is part of the string returned for the *IDN? query described in the Common Commands chapter.

<serial_number> A ten-character alphanumeric serial number enclosed with quotation marks.

Example This example sets the serial number for the oscilloscope to "US12345678".

```
myScope.WriteString ":SERIAL ""US12345678"""
```

Query `:SERial?`

The query returns the current serial number string for the oscilloscope.

Returned Format `[:SERial] US12345678`

Example This example places the serial number for the oscilloscope in the string variable strSerial, then prints the contents of the variable to the computer's screen.

```
Dim strSerial As String   ' Dimension variable.
myScope.WriteString ":SERIAL?"
strSerial = myScope.ReadString
Debug.Print strSerial
```

:SINGle

Command `:SINGle`

The :SINGle command causes the oscilloscope to make a single acquisition when the next trigger event occurs. However, this command does not set the :TRIGger:SWEep to SINGle.

Example This example sets up the oscilloscope to make a single acquisition when the next trigger event occurs.

```
myScope.WriteString ":SINGLE"
```

See Also :TRIGger:SWEep AUTO|TRIGgered|SINGle for how to turn the single sweep off.

:STATus?

Query `:STATus? {CHANnel<N> | FUNCtion<F> | HISTogram | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized | BUS<N> | DIGital<M> | POD<L>}`

The :STATus? query shows whether the specified channel, function, wmemory, histogram, measurement trend, measurement spectrum, or equalized waveform is on or off. A return value of 1 means on and a return value of 0 means off.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

POD<L> refers to the two digital channel pods - one that includes all of the activated digital channels in the D0-D7 range and the other that includes all of the activated digital channels in the D8-D15 range.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15.

Returned Format `[:STATus] {0 | 1}<NL>`

Example This example returns and prints the current status of channel 1.

```
myScope.WriteString ":STATus? CHANnel1"
strCurrent = myScope.ReadString
Debug.Print strCurrent
```

See Also

- ":BLANk" on page 758
- ":VIEW" on page 782

:STOP

Command `:STOP`

The :STOP command causes the oscilloscope to stop acquiring data. To restart the acquisition, use the :RUN or :SINGle command.

Example This example stops the current data acquisition.

```
myScope.WriteString ":STOP"
```

:STORe:JITTer

Command `:STORe:JITTer <file_name>`

The :STORe:JITTer command saves all of the RJ/DJ jitter measurement data to the specified file name. The file that is created has a header section followed by the RJ/DJ measurement results section. After the RJ/DJ measurement results section is the data for each of the measurements. Each data section has a header showing what the measurement data is that follows.

<file_name> A character-quoted ASCII string which can include subdirectories with the name of the file.

Example This example stores the RJ/DJ jitter measurements to a file.

```
myScope.WriteString _
":STORE:JITTer ""c:\Document and Settings\All Users\Shared Documents\
Infiniium\Data\jitter"""
```

:STORe:SETup

Command `:STORe:SETup <setup_memory_num>`

<setup_memory_num> Setup memory number, an integer, 0 through 9.

The :STORe:SETup command saves the current oscilloscope setup in one of the setup memories.

Example This example stores the current oscilloscope setup to setup memory 0.

```
myScope.WriteString ":STORE:SETUP 0"
```

:STORe:WAVeform

Command `:STORe:WAVeform {{CHANnel<N> | FUNCtion<F> | WMEMory<N> | MTRend | MSPectrum},{WMEMory<N>}}`

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

<N> An integer, 1-4.

<F> An integer, 1-16.

The :STORe:WAVeform command copies a channel, function, stored waveform, measurement trend, or measurement spectrum to a waveform memory. The parameter preceding the comma specifies the source and can be any channel, function, or waveform memory. The parameter following the comma is the destination, and can be any waveform memory.

The :WAVeform:VIEW command determines the view of the data being stored.

Example This example copies channel 1 to waveform memory 3.

```
myScope.WriteString ":STORE:WAVEFORM CHANNEL1,WMEMORY3"
```

:TER?

(Trigger Event Register)

Query `:TER?`

The :TER? query reads the Trigger Event Register. A "1" is returned if a trigger has occurred. A "0" is returned if a trigger has not occurred. The autotrigger does not set this register. The register is set to a value of 1 only when the waveform meets the trigger criteria.

Returned Format `{1 | 0}<NL>`

Example This example checks the current status of the Trigger Event Register, places the status in the string variable, strCurrent, then prints the contents of the variable to the computer's screen.

```
Dim strCurrent As String   ' Dimension variable.
myScope.WriteString ":TER?"
strCurrent = myScope.ReadString
Debug.Print strCurrent
```

Once this bit is set, you can clear it only by reading the register with the :TER? query, or by sending a *CLS common command. After the Trigger Event Register is read, it is cleared.

:VIEW

Command

```
:VIEW {CHANnel<N> | FUNCtion<F> | HISTogram | WMEMory<N> | MSTrend
      | MSPectrum | BUS<N> | DIGital<M> | POD<P>}
```

The :VIEW command turns on a channel, function, histogram, or waveform memory. The :BLANk command turns them off.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

<N> An integer, 1-4.

<F> An integer, 1-16.

<M> An integer, 0-15.

Example This example turns on channel 1.

```
myScope.WriteString ":VIEW CHANnel1"
```

See Also

- ":BLANk" on page 758

28
Serial Bus Commands

General :SBUS<N> Commands 784
:SBUS<N>:CAN Commands 787
:SBUS<N>:IIC Commands 792
:SBUS<N>:SPI Commands 796

The :SBUS<N> subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE These commands are only valid when the corresponding serial decode option has been licensed.

General :SBUS<N> Commands

- ":SBUS<N>[:DISPlay]" on page 785
- ":SBUS<N>:MODE" on page 786

:SBUS<N>[:DISPlay]

Command `:SBUS<N>[:DISPlay] <display>`

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

The :SBUS<N>[:DISPlay] command turns displaying of the serial decode bus on or off.

NOTE This command is only valid when a serial decode option has been licensed.

Query `:SBUS<N>[:DISPlay]?`

The :SBUS<N>[:DISPlay]? query returns the current display setting of the serial decode bus.

Returned Format `[:SBUS<N>[:DISPlay]] <display><NL>`

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

See Also

- ":SBUS<N>:MODE" on page 786

:SBUS<N>:MODE

Command `:SBUS<N>:MODE <mode>`

`<mode> ::= {CAN | IIC | SPI}`

The :SBUS<N>:MODE command determines the decode mode for the serial bus.

NOTE This command is only valid when a serial decode option has been licensed.

Query `:SBUS<N>:MODE?`

The :SBUS<n>:MODE? query returns the current serial bus decode mode setting.

Returned Format `[:SBUS<N>:MODE] <mode><NL>`

`<mode> ::= {CAN | IIC | SPI}`

See Also

- ":SBUS<N>:CAN Commands" on page 787
- ":SBUS<N>:IIC Commands" on page 792
- ":SBUS<N>:SPI Commands" on page 796

:SBUS<N>:CAN Commands

- ":SBUS<N>:CAN:SAMPlepoint" on page 788
- ":SBUS<N>:CAN:SIGNal:BAUDrate" on page 789
- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 790
- ":SBUS<N>:CAN:SOURce" on page 791

NOTE These commands are only valid when the automotive CAN serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 786

:SBUS<N>:CAN:SAMPlepoint

Command `:SBUS<N>:CAN:SAMPlepoint <value>`

```
<value><NL>

<value> ::= {60 | 62.5 | 65 | 67.5 | 68 | 70 | 72.5 | 75 | 77.5
   | 80 | 82.5 | 85 | 87.5} in NR3 format
```

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

<N> An integer, 1-4.

Query `:SBUS<N>:CAN:SAMPlepoint?`

The :SBUS<N>:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Returned Format

```
<value><NL>

<value> ::= {60 | 62.5 | 65 | 67.5 | 68 | 70 | 72.5 | 75 | 77.5
   | 80 | 82.5 | 85 | 87.5} in NR3 format
```

See Also

- ":SBUS<N>:MODE" on page 786

:SBUS<N>:CAN:SIGNal:BAUDrate

Command `:SBUS<N>:CAN:SIGNal:BAUDrate <baudrate>`

`<baudrate> ::= a real number from 10E3 to 5E6`

The :SBUS<N>:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 5 Mb/s.

If the baud rate you select does not match the system baud rate, false triggers may occur.

<N> An integer, 1-4.

Query `:SBUS<N>:CAN:SIGNal:BAUDrate?`

The :SBUS<N>:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Returned Format `<baudrate><NL>`

`<baudrate> ::= a real number from 10E3 to 5E6`

See Also

- ":SBUS<N>:MODE" on page 786
- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 790
- ":SBUS<N>:CAN:SOURce" on page 791

:SBUS<N>:CAN:SIGNal:DEFinition

Command `:SBUS<N>:CAN:SIGNal:DEFinition <value>`

`<value> ::= {CANH | CANL | DIFFerential | DIFL}`

The :SBUS<N>:CAN:SIGNal:DEFinition command sets the CAN signal type when :SBUS<N>:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signals:

- CANH — the actual CAN_H differential bus signal.

Dominant low signals:

- CANL — the actual CAN_L differential bus signal.
- DIFL — the CAN differential (L-H) bus signal connected to an analog source channel using a differential probe.
- DIFFerential — the CAN differential bus signal connected to an analog source channel using a differential probe. This is the same as DIFL.

<N> An integer, 1-4.

Query `:SBUS<N>:CAN:SIGNal:DEFinition?`

The :SBUS<N>:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Returned Format `<value><NL>`

`<value> ::= {CANH | CANL | DIFL}`

See Also

- ":SBUS<N>:MODE" on page 786
- ":SBUS<N>:CAN:SIGNal:BAUDrate" on page 789
- ":SBUS<N>:CAN:SOURce" on page 791

:SBUS<N>:CAN:SOURce

Command `:SBUS<N>:CAN:SOURce <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:CAN:SOURce command sets the source for the CAN signal.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:CAN:SOURce?`

The :SBUS<N>:CAN:SOURce? query returns the current source for the CAN signal.

Returned Format `<source><NL>`

See Also

- ":SBUS<N>:MODE" on page 786
- ":SBUS<N>:CAN:SIGNal:DEFinition" on page 790

:SBUS<N>:IIC Commands

- ":SBUS<n>:IIC:ASIZe" on page 793
- ":SBUS<N>:IIC:SOURce:CLOCk" on page 794
- ":SBUS<N>:IIC:SOURce:DATA" on page 795

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option has been licensed.

See Also
- ":SBUS<N>:MODE" on page 786

:SBUS<n>:IIC:ASIZe

Command `:SBUS<n>:IIC:ASIZe <size>`

`<size> ::= {BIT7 | BIT8}`

The :SBUS<n>:IIC:ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

Query `:SBUS<n>:IIC:ASIZe?`

The :SBUS<n>:IIC:ASIZe? query returns the current IIC address width setting.

Returned Format `[:SBUS<n>:IIC:ASIZe] <size><NL>`

See Also

- ":SBUS<N>:MODE" on page 786

:SBUS<N>:IIC:SOURce:CLOCk

Command `:SBUS<N>:IIC:SOURce:CLOCk <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:IIC:SOURce:CLOCk command sets the source for the IIC serial clock (SCL).

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 2 as the source for IIC serial clock.

```
myScope.WriteString ":SBUS1:IIC:SOURce:CLOCk CHANnel2"
```

Query `:SBUS<N>:IIC:SOURce:CLOCk?`

The :SBUS<N>:IIC:SOURce:CLOCk? query returns the current source for the IIC serial clock.

Returned Format `[:SBUS<N>:IIC:SOURce:CLOCk] <source><NL>`

See Also

- ":SBUS<N>:IIC:SOURce:DATA" on page 795
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:IIC:SOURce:DATA

Command `:SBUS<N>:IIC:SOURce:DATA <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:IIC:SOURce:DATA command sets the source for IIC serial data (SDA).

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Example This example selects channel 1 as the source for IIC serial data.

```
myScope.WriteString ":SBUS1:IIC:SOURce:DATA CHANnel1"
```

Query `:SBUS<N>:IIC:SOURce:DATA?`

The :SBUS<N>:IIC:SOURce:DATA? query returns the current source for IIC serial data.

Returned Format `[:SBUS<N>:IIC:SOURce:DATA] <source><NL>`

See Also

- ":SBUS<N>:IIC:SOURce:CLOCk" on page 794
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI Commands

- ":SBUS<N>:SPI:BITorder" on page 797
- ":SBUS<N>:SPI:CLOCk:SLOPe" on page 798
- ":SBUS<N>:SPI:CLOCk:TIMeout" on page 799
- ":SBUS<N>:SPI:FRAMe:STATe" on page 800
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:SPI:TYPE" on page 806
- ":SBUS<N>:SPI:WIDTh" on page 807

NOTE These commands are only valid when the low-speed IIC and SPI serial decode option has been licensed.

See Also

- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:BITorder

Command `:SBUS<N>:SPI:BITorder <order>`

`<order> ::= {LSB | MSB}`

The :SBUS<N>:SPI:BITorder command selects the bit order, most significant bit first (MSB) or least significant bit first (LSB), used when displaying data in the serial decode waveform and in the Lister.

<N> An integer, 1-4.

Query `:SBUS<N>:SPI:BITorder?`

The :SBUS<N>:SPI:BITorder? query returns the current SPI decode bit order.

Returned Format `[:SBUS<N>:SPI:BITorder] <order><NL>`

See Also
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:CLOCk:SLOPe

Command `:SBUS<N>:SPI:CLOCk:SLOPe <slope>`

`<slope> ::= {POSitive | RISing | NEGative | FALLing}`

The :SBUS<N>:SPI:CLOCk:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

<N> An integer, 1-4.

Query `:SBUS<N>:SPI:CLOCk:SLOPe?`

The :SBUS<N>:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Returned Format `[:SBUS<N>:SPI:CLOCk:SLOPe] <slope><NL>`

`<slope> ::= {RIS | FALL}`

See Also

- ":SBUS<N>:SPI:CLOCk:TIMeout" on page 799
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:CLOCk:TIMeout

Command `:SBUS<N>:SPI:CLOCk:TIMeout <time_value>`

`<time_value> ::= time in seconds in NR3 format`

The :SBUS<N>:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 100 ns to 10 s when the :SBUS<N>:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

<N> An integer, 1-4.

Query `:SBUS<N>:SPI:CLOCk:TIMeout?`

The :SBUS<N>:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Returned Format `[:SBUS<N>:SPI:CLOCk:TIMeout] <time value><NL>`

See Also

- ":SBUS<N>:SPI:CLOCk:SLOPe" on page 798
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:FRAMe:STATe" on page 800
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:FRAMe:STATe

Command `:SBUS<N>:SPI:FRAMe:STATe <value>`

`<value> ::= {LOW | HIGH}`

The :SBUS<N>:SPI:FRAMe:STATe command sets the SPI trigger frame state.

<N> An integer, 1-4.

Query `:SBUS<N>:SPI:FRAMe:STATe?`

The :SBUS<N>:SPI:FRAMe:STATe? query returns the current SPI frame state.

Returned Format `[:SBUS<N>:SPI:FRAMe:STATe] <value><NL>`

See Also

- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:SOURce:CLOCk

Command `:SBUS<N>:SPI:SOURce:CLOCk <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:SPI:SOURce:CLOCk?`

The :SBUS<N>:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Returned Format `[:SBUS<N>:SPI:SOURce:CLOCk] <source><NL>`

See Also

- ":SBUS<N>:SPI:CLOCk:SLOPe" on page 798
- ":SBUS<N>:SPI:CLOCk:TIMeout" on page 799
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:SOURce:DATA

Command `:SBUS<N>:SPI:SOURce:DATA <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:SPI:SOURce:DATA command sets the source for the SPI serial MOSI data.

This command is the same as the :SBUS<N>:SPI:SOURce:MOSI command.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:SPI:SOURce:DATA?`

The :SBUS<N>:SPI:SOURce:DATA? query returns the current source for the SPI serial MOSI data.

Returned Format `[:SBUS<N>:SPI:SOURce:DATA] <source><NL>`

See Also

- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:SOURce:FRAMe

Command `:SBUS<N>:SPI:SOURce:FRAMe <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:SPI:SOURce:FRAMe command sets the frame source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:SPI:SOURce:FRAMe?`

The :SBUS<N>:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Returned Format `[:SBUS<N>:SPI:SOURce:FRAMe] <source><NL>`

See Also

- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:SPI:FRAMe:STATe" on page 800
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:SOURce:MISO

Command `:SBUS<N>:SPI:SOURce:MISO <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:SPI:SOURce:MISO command sets the source for the SPI serial MISO data.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:SPI:SOURce:MISO?`

The :SBUS<N>:SPI:SOURce:MISO? query returns the current source for the SPI serial MISO data.

Returned Format `[:SBUS<N>:SPI:SOURce:MISO] <source><NL>`

See Also

- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:SOURce:MOSI

Command `:SBUS<N>:SPI:SOURce:MOSI <source>`

`<source> ::= {CHANnel<N> | FUNCtion<F> | WMEMory<N> | DIGital<M>}`

The :SBUS<N>:SPI:SOURce:MOSI command sets the source for the SPI serial MOSI data.

You can also use the equivalent :SBUS<N>:SPI:SOURce:DATA command to set the MOSI data source.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:SBUS<N>:SPI:SOURce:MOSI?`

The :SBUS<N>:SPI:SOURce:MOSI? query returns the current source for the SPI serial MOSI data.

Returned Format `[:SBUS<N>:SPI:SOURce:MOSI] <source><NL>`

See Also

- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:TYPE

Command `:SBUS<N>:SPI:TYPE <value>`

`<value> ::= {WIRE2 | WIRE3 | WIRE4}`

The :SBUS<N>:SPI:TYPE command specifies whether the type of SPI to decode.

<N> An integer, 1-4.

Example To set the 3-wire SPI decode type:

```
myScope.WriteString ":SBUS1:SPI:TYPE WIRE3"
```

Query `:SBUS<N>:SPI:TYPE?`

The :SBUS<N>:SPI:TYPE? query returns the decode type setting.

Returned Format `[:SBUS<N>:SPI:TYPE] <value><NL>`

See Also

- ":SBUS<N>:SPI:BITorder" on page 797
- ":SBUS<N>:SPI:SOURce:CLOCk" on page 801
- ":SBUS<N>:SPI:SOURce:DATA" on page 802
- ":SBUS<N>:SPI:SOURce:FRAMe" on page 803
- ":SBUS<N>:SPI:SOURce:MISO" on page 804
- ":SBUS<N>:SPI:SOURce:MOSI" on page 805
- ":SBUS<N>:MODE" on page 786

:SBUS<N>:SPI:WIDTh

Command `:SBUS<N>:SPI:WIDTh <word_width>`

`<word_width> ::= integer 4-16 in NR1 format`

The :SBUS<N>:SPI:WIDTh command determines the number of bits in a word of data for SPI.

<N> An integer, 1-4.

Query `:SBUS<N>:SPI:WIDTh?`

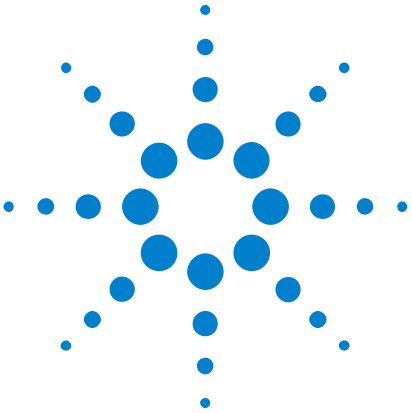
The :SBUS<N>:SPI:WIDTh? query returns the current SPI decode word width.

Returned Format `[:SBUS<N>:SPI:WIDTh] <word_width><NL>`

`<word_width> ::= integer 4-16 in NR1 format`

See Also

- ":SBUS<N>:MODE" on page 786



29
Self-Test Commands

:SELFtest:CANCel 810
:SELFtest:SCOPETEST 811

The SELFtest subsystem commands set up the self-test dialog and run the Infiniium-Series Oscilloscopes Self-Tests.

NOTE

Enclose File Name in Quotation Marks

When specifying a file name, you must enclose it in quotation marks.

:SELFtest:CANCel

Command `:SELFtest:CANCel`

The :SELFtest:CANCel command stops the currently running selftest.

Example This example stops the currently running selftest.

```
myScope.WriteString ":SELF:CANC"
```

:SELFtest:SCOPETEST

Command `:SELFtest:SCOPETEST`

The :SELFtest:SCOPETEST command brings up the self-test dialog in customer self-test mode (Service Extensions Off) and runs the test, "Scope Self Tests." Use the :SELFtest:SCOPETEST? query to determine the status of the test.

Example This example brings up the self-test dialog and runs the oscilloscope self-tests.

```
myScope.WriteString ":SELF:SCOPETEST"
```

Query `:SELFtest:SCOPETEST?`

Returned Format `[:SELFtest:SCOPETEST] <test_name>,<test_status>, <time_stamp><NL>`

<test_status>	Status Description
FAILED	Test completed and failed.
PASSED	Test completed and passed.
WARNING	Test passed but warning message was issued.
CANCELLED	Test was cancelled by user.
NODATA	Self-tests have not been executed on this instrument.
INPROGRESS	Test is in progress.

<test_name> A string as follows: "Scope Self Tests".

<time_stamp> The time stamp follows the test name and test status, and is the part of the returned string that includes the date and time, in the format: "20 May 2009 10:13:35".

Example This example places the current status of the self-test in the string variable, strTxt, then prints the contents of the variable to the computer's screen.

```
Dim strTxt As String
myScope.WriteString ":SELF:SCOPETEST?"
strTxt = myScope.ReadString
Debug.Print strTxt
```

30 Serial Data Equalization Commands

:SPRocessing:CTLequalizer:DISPlay 815
:SPRocessing:CTLequalizer:SOURce 816
:SPRocessing:CTLequalizer:DCGain 817
:SPRocessing:CTLequalizer:NUMPoles 818
:SPRocessing:CTLequalizer:P1 819
:SPRocessing:CTLequalizer:P2 820
:SPRocessing:CTLequalizer:P3 821
:SPRocessing:CTLequalizer:RATe 822
:SPRocessing:CTLequalizer:VERTical 823
:SPRocessing:CTLequalizer:VERTical:OFFSet 824
:SPRocessing:CTLequalizer:VERTical:RANGe 825
:SPRocessing:CTLequalizer:ZERo 826
:SPRocessing:DFEQualizer:STATe 827
:SPRocessing:DFEQualizer:SOURce 828
:SPRocessing:DFEQualizer:NTAPs 829
:SPRocessing:DFEQualizer:TAP 830
:SPRocessing:DFEQualizer:TAP:WIDTh 831
:SPRocessing:DFEQualizer:TAP:DELay 832
:SPRocessing:DFEQualizer:TAP:MAX 833
:SPRocessing:DFEQualizer:TAP:MIN 834
:SPRocessing:DFEQualizer:TAP:GAIN 835
:SPRocessing:DFEQualizer:TAP:UTARget 836
:SPRocessing:DFEQualizer:TAP:LTARget 837
:SPRocessing:DFEQualizer:TAP:AUTomatic 838
:SPRocessing:FFEQualizer:DISPlay 839
:SPRocessing:FFEQualizer:SOURce 840
:SPRocessing:FFEQualizer:NPRecursor 841
:SPRocessing:FFEQualizer:NTAPs 842
:SPRocessing:FFEequalizer:RATe 843
:SPRocessing:FFEQualizer:TAP 844
:SPRocessing:FFEQualizer:TAP:PLENgth 845
:SPRocessing:FFEQualizer:TAP:WIDTh 846
:SPRocessing:FFEQualizer:TAP:DELay 847
:SPRocessing:FFEQualizer:TAP:AUTomatic 848

:SPRocessing:FFEQualizer:TAP:BANDwidth 849
:SPRocessing:FFEQualizer:TAP:BWMode 850
:SPRocessing:FFEQualizer:TAP:TDELay 851
:SPRocessing:FFEQualizer:TAP:TDMode 852
:SPRocessing:FFEQualizer:VERTical 853
:SPRocessing:FFEQualizer:VERTical:OFFSet 854
:SPRocessing:FFEQualizer:VERTical:RANGe 855

The N5461A Serial Data Equalization application is used to re-open partially or completely closed real-time eye diagrams. For additional information on equalization, consult the N5461A Infiniium Serial Data Equalization User's Guide.

:SPRocessing:CTLequalizer:DISPlay

Command `:SPRocessing:CTLequalizer:DISPlay {(OFF | 0) | (ON | 1)}`

The :CTLequalizer:DISPlay command turns the display of a Continuous Time Linear Equalizer (CTLE) real-time eye diagram on or off. Turning CTLE on automatically turns FFE off (and vice versa).

Example This example turns on the display of a CTLE real-time eye diagram.

```
myScope.WriteString ":SPRocessing:CTLequalizer:DISPlay ON"
```

Query `:SPRocessing:CTLequalizer:DISPlay?`

The :SPRocessing:CTLequalizer:DISPlay? query returns whether or not the CTLE real-time eye is displayed.

:SPRocessing:CTLequalizer:SOURce

Command `:SPRocessing:CTLequalizer:SOURce {CHANnel<N> | FUNCtion<F> | WMEMory<N>}`

The :CTLequalizer:SOURce command sets the source for the Continuous Time Linear Equalization.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example sets the CTLE source to Channel 1.

```
myScope.WriteString ":SPRocessing:CTLequalizer:SOURce Channel1"
```

Query `:SPRocessing:CTLequalizer:SOURce?`

The :SPRocessing:CTLequalizer:SOURce? query returns the CTLE source.

:SPRocessing:CTLequalizer:DCGain

Command `:SPRocessing:CTLequalizer:DCGain <dc_gain>`

The :CTLequalizer:DCGain command sets the DC Gain parameter for the Continuous Time Linear Equalization.

<dc_gain> A real number

Example This example sets the CTLE DC Gain parameter to 1.

```
myScope.WriteString ":SPRocessing:CTLequalizer:DCGain 1"
```

Query `:SPRocessing:CTLequalizer:DCGain?`

The :SPRocessing:CTLequalizer:DCGain? query returns the CTLE's DC Gain parameter.

:SPRocessing:CTLequalizer:NUMPoles

Command `:SPRocessing:CTLequalizer:NUMPoles {POLE2 | POLE3 | USB31}`

The :SPRocessing:CTLequalizer:NUMPoles command selects either a 2 Pole, 3 Pole, or USB 3.1 (Gen 2) Continuous Time Linear Equalizer (CTLE).

Example This example selects a 2 Pole CTLE.

```
myScope.WriteString ":SPRocessing:CTLequalizer:NUMPoles POLE2"
```

Query `:SPRocessing:CTLequalizer:NUMPoles?`

The :SPRocessing:CTLequalizer:NUMPoles? query returns the current "number of poles" selection.

:SPRocessing:CTLequalizer:P1

Command `:SPRocessing:CTLequalizer:P1 <pole1_freq>`

The :CTLequalizer:P1 command sets the Pole 1 frequency for the Continuous Time Linear Equalization.

<pole1_freq> A real number

Example This example sets the CTLE Pole 1 frequency to 1GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P1 1e9"
```

Query `:SPRocessing:CTLequalizer:P1?`

The :SPRocessing:CTLequalizer:P1? query returns the CTLE's Pole 1 frequency.

:SPRocessing:CTLequalizer:P2

Command `:SPRocessing:CTLequalizer:P2 <pole2_freq>`

The :CTLequalizer:P1 command sets the Pole 2 frequency for the Continuous Time Linear Equalization.

<pole2_freq> A real number

Example This example sets the CTLE Pole 2 frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P2 4e9"
```

Query `:SPRocessing:CTLequalizer:P2?`

The :SPRocessing:CTLequalizer:P2? query returns the CTLE's Pole 2 frequency.

:SPRocessing:CTLequalizer:P3

Command `:SPRocessing:CTLequalizer:P3 <pole3_freq>`

The :CTLequalizer:P1 command sets the Pole 3 frequency for the Continuous Time Linear Equalization.

<pole3_freq> A real number

Example This example sets the CTLE Pole 3 frequency to 4 GHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:P3 4e9"
```

Query `:SPRocessing:CTLequalizer:P3?`

The :SPRocessing:CTLequalizer:P3? query returns the CTLE's Pole 3 frequency.

:SPRocessing:CTLequalizer:RATe

Command `:SPRocessing:CTLequalizer:RATe <data_rate>`

The :CTLequalizer:RATe command sets the data rate for the CTLE equalizer.

<data_rate> A real number

Example This example sets the CTLE data rate to 3e9.

```
myScope.WriteString ":SPRocessing:CTLequalizer:RATe 3e9"
```

Query `:SPRocessing:CTLequalizer:RATe?`

The :SPRocessing:CTLequalizer:Rate? query returns the CTLE's data rate.

:SPRocessing:CTLequalizer:VERTical

Command `:SPRocessing:CTLequalizer:VERTical {AUTO | MANual}`

The :SPRocessing:CTLequalizer:VERTical command sets the CTLE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the CTLE signal's vertical scale mode to automatic.

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTical AUTO"
```

Query `:SPRocessing:CTLequalizer:VERTical?`

The :SPRocessing:CTLequalizer:VERTical? query returns the current CTLE signal's vertical scale mode setting.

Returned Format `[:SPRocessing:CTLequalizer:VERTical] {AUTO | MANual}`

Example This example places the current setting of the CTLE signal's vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":SPRocessing:CTLequalizer:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:SPRocessing:CTLequalizer:VERTical:OFFSet

Command `:SPRocessing:CTLequalizer:VERTical:OFFSet <offset>`

The :SPRocessing:CTLequalizer:VERTical:OFFSet command sets the CTLE signal's vertical offset.

<offset> A real number for the CTLE signal's vertical offset.

Example This example sets the CTLE signal's vertical offset to 1 volt.

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTICAL:OFFSET 1"
```

Query `:SPRocessing:CTLequalizer:VERTical:OFFSet?`

The:SPRocessing:CTLequalizer:VERTical:OFFSet? query returns the CTLE signal's vertical offset setting.

Returned Format `[:SPRocessing:CTLequalizer:VERTical:OFFSet] <value><NL>`

<value> The CTLE signal's vertical offset setting.

Example This example places the current value of the CTLE signal's vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":SPRocessing:CTLequalizer:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:SPRocessing:CTLequalizer:VERTical:RANGe

Command `:SPRocessing:CTLequalizer:VERTical:RANGe <range>`

The :SPRocessing:CTLequalizer:VERTical:RANGe command sets the CTLE signal's vertical range.

<range> A real number for the full-scale CTLE signal's vertical range.

Example This example sets the CTLE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":SPRocessing:CTLequalizer:VERTICAL:RANGE 16"
```

Query `:SPRocessing:CTLequalizer:VERTical:RANGe?`

The :SPRocessing:CTLequalizer:VERTical:RANGe? query returns the CTLE signal's vertical range setting.

Returned Format `[:SPRocessing:CTLequalizer:VERTical:RANGe] <value><NL>`

<value> The CTLE signal's vertical range setting.

Example This example places the current value of the CTLE signal's vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":SPRocessing:CTLequalizer:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:SPRocessing:CTLequalizer:ZERo

Command `:SPRocessing:CTLequalizer:ZERo <zero_freq>`

The :CTLequalizer:ZERo command sets the zero frequency for the Continuous Time Linear Equalization.

<zero_freq> A real number

Example This example sets the CTLE zero frequency to 900 MHz.

```
myScope.WriteString ":SPRocessing:CTLequalizer:ZERo 9e6"
```

Query `:SPRocessing:CTLequalizer:ZERo?`

The :SPRocessing:CTLequalizer:ZERo? query returns the CTLE's zero frequency.

:SPRocessing:DFEQualizer:STATe

Command `:SPRocessing:DFEQualizer:STATe {(OFF | 0) | (ON | 1)}`

The :DFEQualizer:STATe command turns the Decision Feedback Equalization on or off.

Example This example turns on DFE.

```
myScope.WriteString ":SPRocessing:DFEQualizer:STATe ON"
```

Query `:SPRocessing:DFEQualizer:STATe?`

The :SPRocessing:DFEQualizer:STATe? query returns whether or not DFE is turned on.

:SPRocessing:DFEQualizer:SOURce

Command `:SPRocessing:FFEQualizer:SOURce {CHANnel<N> | FUNCtion<F> | WMEMory<N> | EQUalized}`

The :DFEQualizer:SOURce command sets the source for the Decision Feedback Equalization.

Setting the source to EQUalized means the Feed-Forward Equalized (FFE) waveform is used as the DFE source.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example sets the DFE source to Channel 1.

```
myScope.WriteString ":SPRocessing:DFEQualizer:SOURce Channel1"
```

Query `:SPRocessing:DFEQualizer:SOURce?`

The :SPRocessing:DFEQualizer:SOURce? query returns the DFE source.

:SPRocessing:DFEQualizer:NTAPs

Command `:SPRocessing:DFEQualizer:NTAPs <number>`

The :DFEQualizer:NTAPs command sets the number of taps to be used in the DFE algorithm.

DFE tap indices always begin with 1 and extend to the number of taps.

<number> An integer between 2 and 40

Example This example sets the number of DFE taps to 3.

```
myScope.WriteString ":SPRocessing:DFEQualizer:NTAPs 3"
```

Query `:SPRocessing:DFEQualizer:NTAPs?`

The :SPRocessing:DFEQualizer:NTAPs? query returns the number of DFE taps.

:SPRocessing:DFEQualizer:TAP

Command `:SPRocessing:DFEQualizer:TAP <tap>, <value>`

The :DFEQualizer:TAP command sets the tap value for each DFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

DFE tap indices always start at 1 and extend to the number of taps.

<tap> The tap number; when <tap> == 0, Tap 1 is set

<value> The tap value

Example This example sets the DFE Tap 1 to -1.432.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP 0,-1.432"
```

Query `:SPRocessing:DFEQualizer:TAP?`

The :SPRocessing:DFEQualizer:TAP? query returns the DFE tap values.

:SPRocessing:DFEQualizer:TAP:WIDTh

Command `:SPRocessing:DFEQualizer:TAP:WIDTh <width>`

The :DFEQualizer:TAP:WIDTh command sets the Eye Width field for the DFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for DFE is 0.0. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:WIDTh 0.0"
```

Query `:SPRocessing:DFEQualizer:TAP:WIDTh?`

The :SPRocessing:DFEQualizer:TAP? query returns the eye width used in the DFE tap optimization.

:SPRocessing:DFEQualizer:TAP:DELay

Command `:SPRocessing:DFEQualizer:TAP:DELay <delay>`

The :DFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

<delay> A real number

Query `:SPRocessing:DFEQualizer:TAP:DELay?`

The :SPRocessing:DFEQualizer:TAP:DELay? query returns the value for the DFE Delay field.

:SPRocessing:DFEQualizer:TAP:MAX

Command `:SPRocessing:DFEQualizer:TAP:MAX <max_tap_value>`

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MAX command sets the upper limit on taps determined through optimization.

<max_tap_value> A real number

Example This example sets the Upper Limit field to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:MAX 3.23"
```

Query `:SPRocessing:DFEQualizer:TAP:MAX?`

The :SPRocessing:DFEQualizer:TAP:MAX? query returns the Upper Limit used in the DFE tap optimization.

:SPRocessing:DFEQualizer:TAP:MIN

Command `:SPRocessing:DFEQualizer:TAP:MIN <min_tap_value>`

Some standards have upper and lower limits on the tap values. The :DFEQualizer:TAP:MIN command sets the lower limit on taps determined through optimization.

<min_tap_value> A real number

Example This example sets the Lower Limit field to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:MIN 3.23"
```

Query `:SPRocessing:DFEQualizer:TAP:MIN?`

The :SPRocessing:DFEQualizer:TAP:MIN? query returns the Lower Limit used in the DFE tap optimization.

:SPRocessing:DFEQualizer:TAP:GAIN

Command `:SPRocessing:DFEQualizer:TAP:GAIN <gain>`

The eye diagram drawn after DFE is applied is attenuated. To amplify the eye back to its original size (so you can directly compare the eye at the receiver to the eye at the transmitter), a gain factor needs to be applied. The :DFEQualizer:TAP:GAIN command allows you to set this gain. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

<gain> A real number

Example This example sets the gain to 3.23.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:GAIN 3.23"
```

Query `:SPRocessing:DFEQualizer:TAP:GAIN?`

The :SPRocessing:DFEQualizer:TAP:GAIN? query returns the current gain value.

:SPRocessing:DFEQualizer:TAP:UTARget

Command `:SPRocessing:DFEQualizer:TAP:UTARget <upper_target>`

The Upper Target field dictates the logical high value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical high, its value will be equal to Upper Target. The :DFEQualizer:TAP:UTARget command allows you to set this value.

<upper_target> A real number

Example This example sets the Upper Target to 1.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:UTARget 1.0"
```

Query `:SPRocessing:DFEQualizer:TAP:UTARget?`

The :SPRocessing:DFEQualizer:TAP:UTARget? query returns the current value for the Upper Target field.

:SPRocessing:DFEQualizer:TAP:LTARget

Command `:SPRocessing:DFEQualizer:TAP:LTARget <lower_target>`

The Lower Target field dictates the logical low value used in the DFE algorithm. For example, in DFE, when a bit is determined to be a logical low, its value will be equal to Lower Target. The :DFEQualizer:TAP:LTARget command allows you to set this value.

<lower_target> A real number

Example This example sets the Lower Target to 1.0.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:LTARget 1.0"
```

Query `:SPRocessing:DFEQualizer:TAP:LTARget?`

The :SPRocessing:DFEQualizer:TAP:LTARget? query returns the current value for the Lower Target field.

:SPRocessing:DFEQualizer:TAP:AUTomatic

Command `:SPRocessing:DFEQualizer:TAP:AUTomatic`

The :DFEQualizer:TAP:AUTomatic command starts the DFE tap optimization. Be sure to first specify the number of taps and the max/min tap values.

Example This example starts the DFE tap optimization.

```
myScope.WriteString ":SPRocessing:DFEQualizer:TAP:AUTomatic"
```

:SPRocessing:FFEQualizer:DISPlay

Command `:SPRocessing:FFEQualizer:DISPlay {(OFF | 0) | (ON | 1)}`

The :FFEQualizer:DISPlay command turns the display of a Feed-Forward Equalized (FFE) real-time eye diagram on or off.

Example This example turns on the display of a FFE real-time eye diagram.

```
myScope.WriteString ":SPRocessing:FFEQualizer:DISPlay ON"
```

Query `:SPRocessing:FFEQualizer:DISPlay?`

The :SPRocessing:FFEQualizer:DISPlay? query returns whether or not the FFE real-time eye is displayed.

:SPRocessing:FFEQualizer:SOURce

Command `:SPRocessing:FFEQualizer:SOURce {CHANnel<N> | FUNCtion<F> | WMEMory<N>}`

The :FFEQualizer:SOURce command sets the source for the Feed-Forward Equalization.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example sets the FFE source to Channel 1.

```
myScope.WriteString ":SPRocessing:FFEQualizer:SOURce Channel1"
```

Query `:SPRocessing:FFEQualizer:SOURce?`

The :SPRocessing:FFEQualizer:SOURce? query returns the FFE source.

:SPRocessing:FFEQualizer:NPRecursor

Command `:SPRocessing:FFEQualizer:NPRecursor <number>`

The :FFEQualizer:NPRecursor command sets the number of precursor taps to be used in the FFE algorithm.

<number> An integer between 1 and (NTAPs - 1)

Example This example sets the number of FFE precursor taps to 3.

```
myScope.WriteString ":SPRocessing:FFEQualizer:NPRecursor 3"
```

Query `:SPRocessing:FFEQualizer:NPRecursor?`

The :SPRocessing:FFEQualizer:NPRecursor? query returns the number of FFE precursor taps.

:SPRocessing:FFEQualizer:NTAPs

Command `:SPRocessing:FFEQualizer:NTAPs <number>`

The :FFEQualizer:NTAPs command sets the number of taps to be used in the FFE algorithm.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<number> an integer between 2 and 40

Example This example sets the number of FFE taps to 3.

```
myScope.WriteString ":SPRocessing:FFEQualizer:NTAPs 3"
```

Query `:SPRocessing:FFEQualizer:NTAPs?`

The :SPRocessing:FFEQualizer:NTAPs? query returns the number of FFE taps.

:SPRocessing:FFEequalizer:RATe

Command `:SPRocessing:FFEequalizer:RATe <data_rate>`

The :FFEequalizer:RATe command sets the data rate for the FFE equalizer.

<data_rate> A real number

Example This example sets the FFE data rate to 3e9.

```
myScope.WriteString ":SPRocessing:FFEequalizer:RATe 3e9"
```

Query `:SPRocessing:FFEequalizer:RATe?`

`The :SPRocessing:FFEequalizer:Rate? query returns the FFE's data rate.`

:SPRocessing:FFEQualizer:TAP

Command `:SPRocessing:FFEQualizer:TAP <tap>, <value>`

The :FFEQualizer:TAP command sets the tap value for each FFE tap. For example, when <tap> is equal to 0 then the 0th tap is set to <value>.

The indices of your FFE taps depend on the number of precursor taps being used. For example, if you are using zero precursor taps then your FFE tap indices would range from 0 to (NTAPs - 1). If you are using two precursor taps then your FFE tap indices would range from -2 to (NTAPs - 1 - 2).

<tap> The tap number; when <tap> == 0, Tap 0 is set

<value> The tap value

Example This example sets the second FFE tap to -1.432.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP 2,-1.432"
```

Query `:SPRocessing:FFEQualizer:TAP?`

The :SPRocessing:FFEQualizer:TAP? query returns the FFE tap values.

:SPRocessing:FFEQualizer:TAP:PLENgth

Command `:SPRocessing:FFEQualizer:TAP:PLENgth {CUSTom | PRBS51 | PRBS61 | PRBS71 | PRBS81 | PRBS91 | PRBS101}, <file>`

In order for the tap optimization to work, the algorithm must know the input pattern. You can train the oscilloscope to a known pattern and then use the optimized taps on your live traffic. The :FFEQualizer:TAP:PLENgth command sets the pattern for the FFE tap optimization algorithm.

The file parameter is only used in CUSTom mode.

For more information on this parameter, refer to the *N5461A Infiniium Serial Data Equalization User's Guide*.

<CUSTom> Allows you to import a known pattern via a .prtn file (save at location <file>)

<PRBSX1> Pseudo-random Binary Sequence of length 2^X - 1.

Example This example sets the pattern to PRBS 2^8 - 1.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:PLENgth PRBS81"
```

Query `:SPRocessing:FFEQualizer:TAP:PLENgth?`

The :SPRocessing:FFEQualizer:TAP:PLENgth? query returns the pattern used in optimizing the FFE tap values.

:SPRocessing:FFEQualizer:TAP:WIDTh

Command `:SPRocessing:FFEQualizer:TAP:WIDTh <width>`

The :FFEQualizer:TAP:WIDTh command sets the Eye Width field for the FFE tap optimization. Setting the width to 0.0 means the optimization is only preformed at the location of the clock. Setting the width to 1.0 means the entire acquisition is used in the optimization. The default value for FFE is 0.33. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

<width> A real number between 0.0 and 1.0.

Example This example sets the eye width to 0.0.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:WIDTh 0.0"
```

Query `:SPRocessing:FFEQualizer:TAP:WIDTh?`

The :SPRocessing:FFEQualizer:TAP:WIDTh? query returns the eye width used in the FFE tap optimization.

:SPRocessing:FFEQualizer:TAP:DELay

Command `:SPRocessing:FFEQualizer:TAP:DELay <delay>`

The :FFEQualizer:TAP:DELay command specifies the amount of drift the equalized eye diagram has relative to the unequalized one. This drift is then accounted for so the two eyes overlap. For more information on this parameter, refer to the N5461A Infiniium Serial Data Equalization User's Guide.

<delay> A real number

Query `:SPRocessing:FFEQualizer:TAP:DELay?`

The :SPRocessing:FFEQualizer:TAP:DELay? query returns the value for the FFE Delay field.

:SPRocessing:FFEQualizer:TAP:AUTomatic

Command `:SPRocessing:FFEQualizer:TAP:AUTomatic`

The :FFEQualizer:TAP:AUTomatic command starts the FFE tap optimization. Be sure to first specify the number of taps and specify the Pattern and Eye Width parameters.

Example This example starts the FFE tap optimization.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:AUTomatic"
```

:SPRocessing:FFEQualizer:TAP:BANDwidth

Command `:SPRocessing:FFEQualizer:TAP:BANDwidth <bandwidth>`

The :FFEQualizer:TAP:BANDwidth command is only needed if the FFEQualizer:TAP:BWMode command is set to CUSTom and in this case it sets the bandwidth at which the response generated by equalization rolls off. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

<bandwidth> The bandwidth at which the response generated by equalization rolls off.

Query `:SPRocessing:FFEQualizer:TAP:BANDwidth?`

The :SPRocessing:FFEQualizer:TAP:BANDwidth? query returns the current value for the BANDwidth parameter.

:SPRocessing:FFEQualizer:TAP:BWMode

Command `:SPRocessing:FFEQualizer:TAP:BWMode {TSBandwidth | TTDelay | CUSTom}`

The :FFEQualizer:TAP:BWMode command sets the bandwidth at which the response generated by equalization is rolled off. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

Example This example sets the FFE Bandwidth Mode to TTDELay.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:BWMode TTDelay"
```

Query `:SPRocessing:FFEQualizer:TAP:BWMode?`

The :SPRocessing:FFEQualizer:TAP:BWMode? query returns the FFE Bandwidth Mode.

:SPRocessing:FFEQualizer:TAP:TDELay

Command `:SPRocessing:FFEQualizer:TAP:TDELay <delay_value>`

The :FFEQualizer:TAP:TDELay command is only needed if the FFEQualizer:TAP:TDMode is set to CUSTom. To determine what this value should be, use the equation: tap delay = 1/[(data rate)x(# of taps per bit)]. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

<delay_value> A real number

Query `:SPRocessing:FFEQualizer:TAP:TDELay?`

The :SPRocessing:FFEQualizer:TAP:TDELay? query returns the current value for the tap delay.

:SPRocessing:FFEQualizer:TAP:TDMode

Command `:SPRocessing:FFEQualizer:TAP:TDMode {TBITrate | CUSTom}`

The :FFEQualizer:TAP:TDMode command sets Tap Delay field to either Track Data Rate or Custom. If you are using one tap per bit, use the TBITrate selection. If you are using multiple taps per bit, use CUSTom and then use the FFEQualizer:TAP:TDELay command to set the value. To understand more about this parameter, consult the N5461A Infiniium Serial Data Equalization User's Guide.

Example This example sets the FFE Tap Delay mode to TBITrate.

```
myScope.WriteString ":SPRocessing:FFEQualizer:TAP:TDMode TBITrate"
```

Query `:SPRocessing:FFEQualizer:TAP:TDMode?`

The :SPRocessing:FFEQualizer:TAP:TDMode? query returns the current Tap Delay mode.

:SPRocessing:FFEQualizer:VERTical

Command `:SPRocessing:FFEQualizer:VERTical {AUTO | MANual}`

The :SPRocessing:FFEQualizer:VERTical command sets the FFE signal's vertical scale mode to automatic or manual. In automatic mode, the oscilloscope automatically selects the vertical scaling and offset. In manual mode, you can set your own scaling and offset values.

Example This example sets the FFEE signal's vertical scale mode to automatic.

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTical AUTO"
```

Query `:SPRocessing:FFEQualizer:VERTical?`

The :SPRocessing:FFEQualizer:VERTical? query returns the current FFE signal's vertical scale mode setting.

Returned Format `[:SPRocessing:FFEQualizer:VERTical] {AUTO | MANual}`

Example This example places the current setting of the FFE signal's vertical scale mode in the string variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":SPRocessing:FFEQualizer:VERTICAL?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:SPRocessing:FFEQualizer:VERTical:OFFSet

Command `:SPRocessing:FFEQualizer:VERTical:OFFSet <offset>`

The :SPRocessing:FFEQualizer:VERTical:OFFSet command sets the FFE signal's vertical offset.

<offset> A real number for the FFE signal's vertical offset.

Example This example sets the FFE signal's vertical offset to 1 volt.

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTICAL:OFFSET 1"
```

Query `:SPRocessing:FFEQualizer:VERTical:OFFSet?`

The:SPRocessing:FFEQualizer:VERTical:OFFSet? query returns the FFE signal's vertical offset setting.

Returned Format `[:SPRocessing:FFEQualizer:VERTical:OFFSet] <value><NL>`

<value> The FFE signal's vertical offset setting.

Example This example places the current value of the FFE signal's vertical offset in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":SPRocessing:FFEQualizer:VERTICAL:OFFSET?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:SPRocessing:FFEQualizer:VERTical:RANGe

Command `:SPRocessing:FFEQualizer:VERTical:RANGe <range>`

The :SPRocessing:FFEQualizer:VERTical:RANGe command sets the FFE signal's vertical range.

<range> A real number for the full-scale FFE signal's vertical range.

Example This example sets the FFE signal's vertical range to 16 volts (2 volts times 8 divisions.)

```
myScope.WriteString ":SPRocessing:FFEQualizer:VERTICAL:RANGE 16"
```

Query `:SPRocessing:FFEQualizer:VERTical:RANGe?`

The :SPRocessing:FFEQualizer:VERTical:RANGe? query returns the FFE signal's vertical range setting.

Returned Format `[:SPRocessing:FFEQualizer:VERTical:RANGe] <value><NL>`

<value> The FFE signal's vertical range setting.

Example This example places the current value of the FFE signal's vertical range in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":SPRocessing:FFEQualizer:VERTICAL:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

31 System Commands

:SYSTem:DATE 858
:SYSTem:DEBug 859
:SYSTem:DSP 861
:SYSTem:ERRor? 862
:SYSTem:HEADer 863
:SYSTem:LOCK 864
:SYSTem:LONGform 865
:SYSTem:PRESet 866
:SYSTem:SETup 867
:SYSTem:TIME 869

SYSTem subsystem commands control the way query responses are formatted, send and receive setup strings, and enable reading and writing to the advisory line of the oscilloscope. You can also set and read the date and time in the oscilloscope using the SYSTem subsystem commands.

:SYSTem:DATE

Command `:SYSTem:DATE <day>,<month>,<year>`

The :SYSTem:DATE command sets the date in the oscilloscope, and is not affected by the *RST common command.

<year> Specifies the year in the format <yyyy> | <yy>. The values range from 1992 to 2035.

<month> Specifies the month in the format <1, 2, . . . 12> | <JAN, FEB, MAR . . .>.

<day> Specifies the day in the format <1 . . . 31>.

Example This example sets the date to December 1, 2002.

```
myScope.WriteString ":SYSTEM:DATE 1,12,02"
```

Query `:SYSTem:DATE?`

The :SYSTem:DATE? query returns the current date in the oscilloscope.

Returned Format `[:SYSTem:DATE] <day> <month> <year><NL>`

Example This example queries the date.

```
Dim strDate As String
myScope.WriteString ":SYSTEM:DATE?"
strDate = myScope.ReadString
Debug.Print strDate
```

:SYSTem:DEBug

Command `:SYSTem:DEBug {{ON|1}[,<output_mode>[,"<file_name>" [,<create_mode>]]] | {OFF|0}}`

The :SYSTem:DEBug command turns the debug mode on and off. This mode enables the tracing of incoming remote commands. If you select CREate mode, a new file is created, and/or an existing file is overwritten. If you select APPend mode, the information is appended to an existing file. The :SYSTem:DEBug command shows any header and/or parameter errors.

The default create mode is CREate, the default output mode is FileSCReen, and the default file name is "c:\Document and Settings\All Users\Shared Documents\Infiniium\Data\debug.txt". In debug mode, the File View button lets you view the current debug file, or any other debug file. This is a read-only mode.

<output_mode> {FILE | SCReen | FileSCReen}

<file_name> An MS-DOS compatible name of the file, a maximum of 254 characters long (including the path name, if used). The file name assumes the present working directory if a path does not precede the file name.

<create_mode> {CREate | APPend}

Examples This example turns on the debug/trace mode and creates a debug file.

```
myScope.WriteString ":SYSTEM:DEBUG ON,FILE,
""C:\Document and Settings\All Users\Shared Documents\Infiniium\Data\
pacq8xx.txt"",CREATE"
```

The created file resembles:

```
Debug information file C:\Document and Settings\All Users\
Shared Documents\Infiniium\Data\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
Model: DSO90804A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
```

This example appends information to the debug file.

```
myScope.WriteString ":SYSTEM:DEBUG ON,FILE,
""C:\Document and Settings\All Users\Shared Documents\Infiniium\Data\
pacq8xx.txt"",APPEND"
```

After appending information, the file resembles:

```
Debug information file C:\Document and Settings\All Users\
Shared Documents\Infiniium\Data\pacq8xx.txt
Date:  1 DEC 2002
Time:  09:59:35
Model: DSO90804A
Serial#:  sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$

Debug information file C:\Document and Settings\All Users\
Shared Documents\Infiniium\Data\pacq8xx.txt appended
Date:  1 DEC 2002
Time:  10:10:35
Model: DSO90804A
Serial#:   sn ?
>:syst:err? string$<NL>
<:SYSTEM:ERROR 0,"No error"$
>:ACQuire:BEST FLATness$<NL>

?     ^
?-113, Undefined header
>:syst:err? string$<NL>
<:SYSTEM:ERROR -113,"Undefined header"$
```

Query

```
:SYSTem:DEBug?
```

The :SYSTem:DEBug? query returns the current debug mode settings.

Returned Format

```
[:SYSTem:DEBug]  {{1,<output_mode>,"<file_name>", <create_mode>} | 0} <N
L>
```

:SYSTem:DSP

Command `:SYSTem:DSP "<string>"`

The :SYSTem:DSP command writes a quoted string, excluding quotation marks, to the advisory line of the instrument display. If you want to clear a message on the advisory line, send a null (empty) string.

<string> An alphanumeric character array up to 86 bytes long.

Example This example writes the message, "Test 1" to the advisory line of the oscilloscope.

```
myScope.WriteString ":SYSTEM:DSP ""Test 1"""
```

Query `:SYSTem:DSP?`

The :SYSTem:DSP? query returns the last string written to the advisory line. This may be a string written with a :SYSTem:DSP command, or an internally generated advisory.

The string is actually read from the message queue. The message queue is cleared when it is read. Therefore, the displayed message can only be read once over the bus.

Returned Format `[:SYSTem:DSP] <string><NL>`

Example This example places the last string written to the advisory line of the oscilloscope in the string variable, strAdvisory. Then, it prints the contents of the variable to the computer's screen.

```
Dim strAdvisory As String   ' Dimension variable.
myScope.WriteString ":SYSTEM:DSP?"
strAdvisory = myScope.ReadString
Debug.Print strAdvisory
```

:SYSTem:ERRor?

Query `:SYSTem:ERRor? [{NUMBer | STRing}]`

The :SYSTem:ERRor? query outputs the next error number in the error queue over the remote interface. When either NUMBer or no parameter is specified in the query, only the numeric error code is output. When STRing is specified, the error number is output followed by a comma and a quoted string describing the error. Table 30 lists the error numbers and their corresponding error messages.

Returned Format `[:SYSTem:ERRor] <error_number>[,<quoted_string>]<NL>`

<error_number> A numeric error code.

<quoted_string> A quoted string describing the error.

Example This example reads the oldest error number and message in the error queue into the string variable, strCondition, then prints the contents of the variable to the computer's screen.

```
Dim strCondition As String   ' Dimension variable.
myScope.WriteString ":SYSTEM:ERROR? STRING"
strCondition = myScope.ReadString
Debug.Print strCondition
```

Infiniium Oscilloscopes have an error queue that is 30 errors deep and operates on a first-in, first-out (FIFO) basis. Successively sending the :SYSTem:ERRor? query returns the error numbers in the order that they occurred until the queue is empty. When the queue is empty, this query returns headers of 0, "No error." Any further queries return zeros until another error occurs. Note that front-panel generated errors are also inserted in the error queue and the Event Status Register.

NOTE

Send *CLS Before Other Commands or Queries

Send the *CLS common command to clear the error queue and Event Status Register before you send any other commands or queries.

See Also The "Error Messages" chapter for more information on error messages and their possible causes.

:SYSTem:HEADer

Command `:SYSTem:HEADer {{ON|1} | {OFF|0}}`

The :SYSTem:HEADer command specifies whether the instrument will output a header for query responses. When :SYSTem:HEADer is set to ON, the query responses include the command header.

Example This example sets up the oscilloscope to output command headers with query responses.

```
myScope.WriteString ":SYSTEM:HEADER ON"
```

Query `:SYSTem:HEADer?`

The :SYSTem:HEADer? query returns the state of the :SYSTem:HEADer command.

Returned Format `[:SYSTem:HEADer] {1|0}<NL>`

Example This example prints the system header setting.

```
Dim strSetting As String
myScope.WriteString ":syst:head?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

NOTE

Turn Headers Off when Returning Values to Numeric Variables

Turn headers off when returning values to numeric variables. Headers are always off for all common command queries because headers are not defined in the IEEE 488.2 standard.

:SYSTem:LOCK

Command `:SYSTem:LOCK {{ON | 1} | {OFF | 0}}`

The :SYSTem:LOCK ON command disables the front panel. The front panel can be re-enabled by sending the :SYSTem:LOCK OFF command or by using the mouse to click on the Minimize button in the upper right-hand corner of the oscilloscope screen.

Example This example disables the oscilloscope's front panel.

```
myScope.WriteString ":SYSTEM:LOCK ON"
```

Query `:SYSTem:LOCK?`

The :SYSTem:LOCK? query returns the state of the :SYSTem:LOCK command.

Returned Format `[:SYSTem:LOCK] {1 | 0}<NL>`

:SYSTem:LONGform

Command `:SYSTem:LONGform {{ON | 1} | {OFF | 0}}`

The :SYSTem:LONGform command specifies the format for query responses. If the LONGform is set to OFF, command headers and alpha arguments are sent from the oscilloscope in the short form (abbreviated spelling). If LONGform is set to ON, the whole word is output.

Example This example sets the format for query responses from the oscilloscope to the short form (abbreviated spelling).

```
myScope.WriteString ":SYSTEM:LONGFORM OFF"
```

Query `:SYSTem:LONGform?`

The :SYSTem:LONGform? query returns the current state of the :SYSTem:LONGform command.

Returned Format `[:SYSTem:LONGform] {1 | 0}<NL>`

Example This example checks the current format for query responses from the oscilloscope, and places the result in the string variable, strResult. Then, it prints the contents of the variable to the computer's screen.

```
Dim strResult As String   ' Dimension variable.
myScope.WriteString ":SYSTEM:LONGFORM?"
strResult = myScope.ReadString
Debug.Print strResult
```

NOTE

LONGform Does Not Affect Input Headers and Arguments

LONGform has no effect on input headers and arguments sent to the instrument. You may send headers and arguments to the oscilloscope in either the long form or short form, regardless of the current state of the :SYSTem:LONGform command.

:SYSTem:PRESet

Command `:SYSTem:PRESet [ {DEFault|FACtory} ]`

The :SYSTem:PRESet command performs a Default Setup just like the oscilloscope's Default Setup key. Using this command does not change any of the control settings found in the User Preferences dialog box, display color settings, screen options, probe skew, probe external adapter settings for differential probes, or probe internal attenuation and gain settings for differential probes. The parameters are optional. A default reset will occur if no parameters are used or the DEFault parameter is used. A factory default occurs with the FACtory parameter.

Example This example performs an oscilloscope default setup.

```
myScope.WriteString ":SYSTEM:PRESet"
```

:SYSTem:SETup

Command `:SYSTem:SETup <binary_block_data>`

The :SYSTem:SETup command sets up the oscilloscope as defined by the data in the binary block of data from the computer.

CAUTION

Setups saved from Infiniium software versions prior to 2.00 may not load correctly in software versions 4.30 and greater.

You can remedy this by re-saving any pre-2.00 setups using any version of software from version 2.00 to version 4.20.

Setups saved from software versions between 2.00 and 4.20 should load correctly into version 4.30 and greater.

<binary_block_data> A binary block of data, consisting of bytes of setup information. The number of bytes is a dynamic number that is read and allocated by oscilloscope's software.

Example This example reads setup information from a file and restores it to the oscilloscope.

```
' Read setup from a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Dim varSetup As Variant
Open strPath For Binary Access Read As hFile    ' Open file for input.
Get hFile, , varSetup    ' Read data.
Close hFile    ' Close file.

' Write setup to oscilloscope.
myScope.WriteIEEEBlock ":SYSTem:SETup", varSetup
Debug.Print "Setup bytes restored: " + CStr(LenB(varSetup))
```

Query `:SYSTem:SETup?`

The :SYSTem:SETup? query outputs the oscilloscope's current setup to the computer in binary block data format as defined in the IEEE 488.2 standard.

Returned Format `[:SYSTem:SETup] #NX...X<setup_data_string><NL>`

The first character in the setup data block is a number added for disk operations.

Example This example stores the current oscilloscope setup to the variable, varSetup, and then saves it to a file.

```
' Get setup from the oscilloscope.
Dim varSeup As Variant
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":SYSTem:SETup?"
varSetup = myScope.ReadIEEEBlock(BinaryType_UI1)

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varSetup   ' Write data.
Close hFile   ' Close file.
Debug.Print "Setup bytes saved: " + CStr(LenB(varSetup))
```

NOTE

:SYSTem:SETup Can Operate Just Like *LRN?

When headers and LONGform are on, the :SYSTem:SETup? query operates the same as the *LRN? query in the common commands. Otherwise, *LRN? and :SYSTem:SETup are not interchangeable.

:SYSTem:TIME

Command `:SYSTem:TIME <hour>,<minute>,<second>`

The :SYSTem:TIME command sets the time in the oscilloscope and is not affected by the *RST common command.

<hour> 0...23

<minute> 0...59

<second> 0...59

Example This example sets the oscilloscope time to 10:30:45 a.m.

```
myScope.WriteString ":SYSTEM:TIME 10,30,45"
```

Query `:SYSTem:TIME?`

The :SYSTem:TIME? query returns the current time in the oscilloscope.

Returned Format `[:SYSTem:TIME] <hour>,<minute>,<second>`

32 Time Base Commands

:TIMebase:POSition 872
:TIMebase:RANGe 873
:TIMebase:REFClock 874
:TIMebase:REFerence 875
:TIMebase:SCALe 876
:TIMebase:VIEW 877
:TIMebase:WINDow:DELay 878
:TIMebase:WINDow:POSition 879
:TIMebase:WINDow:RANGe 880
:TIMebase:WINDow:SCALe 881

The TIMebase subsystem commands control the horizontal (X axis) oscilloscope functions.

:TIMebase:POSition

Command `:TIMebase:POSition <position_value>`

The :TIMebase:POSition command sets the time interval between the trigger event and the delay reference point. The delay reference point is set with the :TIMebase:REFerence command.

<position_value> A real number for the time in seconds from trigger to the delay reference point.

Example This example sets the delay position to 2 ms.

```
myScope.WriteString ":TIMEBASE:POSITION 2E-3"
```

Query `:TIMebase:POSition?`

The :TIMebase:POSition? query returns the current delay value in seconds.

Returned Format `[:TIMebase:POSition] <position_value><NL>`

Example This example places the current delay value in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:POSITION?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:TIMebase:RANGe

Command `:TIMebase:RANGe <full_scale_range>`

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds. The range value is ten times the time-per-division value.

<full_scale_range> A real number for the horizontal time, in seconds. The timebase range is 50 ps (5 ps/div) to 200 s (20 s/div).

Example This example sets the full-scale horizontal range to 10 ms.

```
myScope.WriteString ":TIMEBASE:RANGE 10E-3"
```

Query `:TIMebase:RANGe?`

The :TIMebase:RANGe? query returns the current full-scale horizontal time.

Returned Format `[:TIMebase:RANGe] <full_scale_range><NL>`

Example This example places the current full-scale horizontal range value in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:RANGE?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:TIMebase:REFClock

Command `:TIMebase:REFClock {{ON | 1} | {OFF | 0} | HFRequency}`

The :TIMebase:REFClock command enables or disables the 10 MHz REF IN BNC input (ON or OFF) or the 100 MHz REF IN SMA input (HFRequency or OFF) located on the rear panel of the oscilloscope.

When this feature is enabled, the external reference input is used as a reference clock for the oscilloscope's horizontal scale section instead of the internal reference clock.

Example This example turns on the 10 MHz reference clock mode.

```
myScope.WriteString ":TIMebase:REFClock ON"
```

Query `:TIMebase:REFClock?`

The :TIMebase:REFClock? query returns the current state of the reference clock mode control.

Returned Format `[TIMebase:REFClock] {1 | 0 | HFR}<NL>`

Example This example places the current value of the reference clock mode control in the variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMebase:REFClock?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:TIMebase:REFerence

Command `:TIMebase:REFerence {LEFT | CENTer | RIGHt}`

The :TIMebase:REFerence command sets the delay reference to the left, center, or right side of the display.

Example This example sets the delay reference to the center of the display.

```
myScope.WriteString ":TIMEBASE:REFERENCE CENTER"
```

Query `:TIMebase:REFerence?`

The :TIMebase:REFerence? query returns the current delay reference position.

Returned Format `[:TIMebase:REFerence] {LEFT | CENTer | RIGHt}<NL>`

Example This example places the current delay reference position in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":TIMEBASE:REFERENCE?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:TIMebase:SCALe

Command `:TIMebase:SCALe <time>`

The :TIMebase:SCALe command sets the time base scale. This corresponds to the horizontal scale value displayed as time/div on the oscilloscope screen.

<time> A real number for the time value, in seconds per division. The timebase scale is 5 ps/div to 20 s/div.

Example This example sets the scale to 10 ms/div.

```
myScope.WriteString ":TIMEBASE:SCALE 10E-3"
```

Query `:TIMebase:SCALe?`

The :TIMebase:SCALe? query returns the current scale time setting.

Returned Format `[:TIMebase:SCALe] <time><NL>`

Example This example places the current scale value in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:SCALE?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:TIMebase:VIEW

Command `:TIMebase:VIEW {MAIN | WINDow}`

The :TIMebase:VIEW command turns the horizontal zoom mode on and off. This is the same as using the front panel **[Zoom]** key.

Example This example turns the horizontal zoom mode on.

```
myScope.WriteString ":TIMebase:VIEW WINDow"
```

Query `:TIMebase:VIEW?`

The :TIMebase:VIEW? query returns the horizontal zoom mode setting.

Returned Format `[:TIMebase:VIEW] {MAIN | WINDow}<NL>`

Example This example places the current view in the string variable, strState, then prints the contents of the variable to the computer's screen.

```
Dim strState As String   ' Dimension variable.
myScope.WriteString ":TIMebase:VIEW?"
strState = myScope.ReadString
Debug.Print strState
```

:TIMebase:WINDow:DELay

Command `:TIMebase:WINDow:DELay <delay_value>`

The :TIMebase:WINDow:DELay sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

NOTE

This Command is Provided for Compatibility

This command is the same as the :TIMebase:WINDow:POSition command, and is provided for compatibility with programs written for previous oscilloscopes. The preferred command for compatibility with Infiniium Oscilloscopes is :TIMebase:WINDow:POSition.

<delay_value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example This example sets the time base window delay position to 20 ns.

```
myScope.WriteString ":TIMEBASE:WINDOW:DELAY 20E-9"
```

Query `:TIMebase:WINDow:DELay?`

The :TIMebase:WINDow:DELay? query returns the current horizontal position in the delayed view.

Returned Format `[:TIMebase:WINDow:DELay] <delay_position><NL>`

Example This example places the current horizontal position in the delayed view in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:WINDOW:DELAY?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

See Also The :TIMebase:WINDow:POSition command performs the same function as this command and should be used in new programs.

:TIMebase:WINDow:POSition

Command `:TIMebase:WINDow:POSition <position_value>`

The :TIMebase:WINDow:POSition sets the horizontal position in the delayed view of the main sweep. The range for this command is determined by the main sweep range and the main sweep horizontal position. The value for this command must keep the time base window within the main sweep range.

<position_value> A real number for the time in seconds from the trigger event to the delay reference point. The maximum position depends on the main sweep range and the main sweep horizontal position.

Example This example sets the time base window delay position to 20 ns.

```
myScope.WriteString ":TIMEBASE:WINDOW:POSITION 20E-9"
```

Query `:TIMebase:WINDow:POSition?`

The :TIMebase:WINDow:POSition? query returns the current horizontal position in the delayed view.

Returned Format `[:TIMebase:WINDow:POSition] <position_value><NL>`

Example This example places the current horizontal position in the delayed view in the numeric variable, varSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:WINDOW:POSITION?"
varSetting = myScope.ReadNumber
Debug.Print FormatNumber(varSetting, 0)
```

:TIMebase:WINDow:RANGe

Command `:TIMebase:WINDow:RANGe <full_scale_range>`

The :TIMebase:WINDow:RANGe command sets the full-scale range of the delayed view. The range value is ten times the time per division of the delayed view. The maximum range of the delayed view is the current main range. The minimum delayed view range is 10 ps (1 ps/div).

<full_scale_range> A real number for the full-scale range of the time base window, in seconds.

Example This example sets the full-scale range of the delayed view to 100 ns.

```
myScope.WriteString ":TIMEBASE:WINDOW:RANGE 100E-9"
```

Query `:TIMebase:WINDow:RANGe?`

The :TIMebase:WINDow:RANGe? query returns the current full-scale range of the delayed view.

Returned Format `[:TIMebase:WINDow:RANGe] <full_scale_range><NL>`

Example This example reads the current full-scale range of the delayed view into the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":TIMEBASE:WINDOW:RANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:TIMebase:WINDow:SCALe

Command `:TIMebase:WINDow:SCALe <time>`

The :TIMebase:WINDow:SCALe command sets the time/div in the delayed view. This command rescales the horizontal components of displayed waveforms.

<time> A real number for the delayed windows scale.

Example This example sets the scale of the time base window to 2 milliseconds/div.

```
myScope.WriteString ":TIMEBASE:WINDOW:SCALE 2E-3"
```

Query `:TIMebase:WINDow:SCALe?`

The :TIMebase:WINDow:SCALe? query returns the scaled window time, in seconds/div.

Returned Format `[:TIMebase:WINDow:SCALe] <time><NL>`

33 Trigger Commands

General Trigger Commands 885
Comm Trigger Commands 899
Delay Trigger Commands 905
Edge Trigger Commands 915
Glitch Trigger Commands 918
Pattern Trigger Commands 922
Pulse Width Trigger Commands 925
Runt Trigger Commands 931
Sequence Trigger Commands 936
Setup and Hold Trigger Commands 945
State Trigger Commands 952
Timeout Trigger Commands 957
Transition Trigger Commands 961
TV Trigger Commands 966
Window Trigger Commands 977

The oscilloscope trigger circuitry helps you locate the waveform you want to view. There are several different types of triggering, but the one that is used most often is edge triggering. Edge triggering identifies a trigger condition by looking for the slope (rising or falling) and voltage level (trigger level) on the source you select. Any input channel, auxiliary input trigger, or line can be used as the trigger source.

The commands in the TRIGger subsystem define the conditions for triggering. Many of the commands in the TRIGger subsystem are used in more than one of the trigger modes. The command set has been defined to closely represent the front-panel trigger menus. As a trade-off, there may be less compatibility between Infiniium Oscilloscopes and command sets for previous oscilloscopes. Infiniium Oscilloscopes still accept some commands for compatibility with previous instruments. An alternative command that is accepted by the oscilloscope is noted for a particular command.

Summary of Trigger Modes and Commands

Make sure the oscilloscope is in the proper trigger mode for the command you want to send. One method of ensuring that the oscilloscope is in the proper trigger mode is to send the :TRIGger:MODE command in the same program message as the parameter to be set.

For example, these commands place the instrument in the proper triggering mode you select:

```
:TRIGger:MODE <Trigger_mode>
```

<Trigger_mode>

The trigger modes include COMM, DELay, EDGE, GLITch, PATTern, PWIDth, RUNT, SEQuence, SHOLd, STATe, TIMeout, TRANsition, TV, and WINDow. Each mode is described with its command set in this chapter.

General Trigger Commands

- ":TRIGger:AND:ENABle" on page 886
- ":TRIGger:AND:SOURce" on page 887
- ":TRIGger:HOLDoff" on page 888
- ":TRIGger:HOLDoff:MAX" on page 889
- ":TRIGger:HOLDoff:MIN" on page 890
- ":TRIGger:HOLDoff:MODE" on page 891
- ":TRIGger:HTHReshold" on page 892
- ":TRIGger:HYSTeresis" on page 893
- ":TRIGger:LEVel" on page 894
- ":TRIGger:LTHReshold" on page 895
- ":TRIGger:MODE" on page 896
- ":TRIGger:SWEep" on page 898

:TRIGger:AND:ENABle

Command `:TRIGger:AND[{1 | 2}]:ENABle {{ON | 1} | {OFF | 0}}`

The :TRIGger:AND:ENABle command enables the ability to further qualify the trigger using other channels.

The optional [{1 | 2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:AND:ENABle?`

The query returns the current state of the AND qualifier.

Returned Format `[:TRIGger:AND:ENABle] {1 | 0}<NL>`

:TRIGger:AND:SOURce

Command `:TRIGger:AND[{1 | 2}]:SOURce CHANnel<N>,{HIGH | LOW | DONTcare}`

The :TRIGger:AND:SOURce command sets the logic value used to qualify the trigger for the specified channel. The TRIGger:LEVel command determines what voltage level is considered a HIGH or a LOW logic value. If you set more than one channel to a HIGH or a LOW, then the multiple channels are used to qualify the trigger.

The optional [{1 | 2}] parameter sets whether the AND qualifier goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:AND:SOURce? CHANnel<N>`

The query returns the logic value for the designated channel.

Returned Format `[:TRIGger:AND:SOURce CHANnel<N>] {HIGH | LOW | DONTcare}<NL>`

:TRIGger:HOLDoff

Command `:TRIGger:HOLDoff <holdoff_time>`

The :TRIGger:HOLDoff command specifies the amount of time the oscilloscope should wait after receiving a trigger before enabling the trigger again.

<holdoff_time> A real number for the holdoff time, ranging from 80 ns to 10 s.

Query `:TRIGger:HOLDoff?`

The query returns the current holdoff value for the current mode.

Returned Format `[:TRIGger:HOLDoff] <holdoff><NL>`

:TRIGger:HOLDoff:MAX

Command `:TRIGger:HOLDoff:MAX <holdoff_time>`

This command is only used when you set the :TRIGger:HOLDoff:MODe command to RANDom. The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random holdoff mode ensures that the oscilloscope re-arms after each acquisition in a manner that minimizes or eliminates the likelihood of triggering at the beginning of a DDR burst. Randomizing the holdoff increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

<holdoff_time> A real number for the maximum random holdoff time.

Query `:TRIGger:HOLDoff:MAX?`

The query returns the current maximum holdoff value for the random holdoff mode.

Returned Format `[:TRIGger:HOLDoff:MAX] <holdoff><NL>`

:TRIGger:HOLDoff:MIN

Command `:TRIGger:HOLDoff:MIN <holdoff_time>`

This command is only used when you set the :TRIGger:HOLDoff:MODe command to RANDom. The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random holdoff mode ensures that the oscilloscope re-arms after each acquisition in a manner that minimizes or eliminates the likelihood of triggering at the beginning of a DDR burst. Randomizing the holdoff increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

<holdoff_time> A real number for the minimum random holdoff time.

Query `:TRIGger:HOLDoff:MIN?`

The query returns the current minimum holdoff value for the random holdoff mode.

Returned Format `[:TRIGger:HOLDoff:MIN] <holdoff><NL>`

:TRIGger:HOLDoff:MODE

Command `:TRIGger:HOLDoff:MAX {FIXed | RANDom}`

The Fixed mode sets the amount of time that the oscilloscope waits before re-arming the trigger circuitry. It can be used to stabilize the display of complex waveforms.

The RANDom mode varies the trigger holdoff from one acquisition to another by randomizing the time values between triggers. The randomized values can be between the values specified by the :TRIGger:HOLDoff:MAX and :TRIGger:HOLDoff:MIN commands.

The Random holdoff mode ensures that the oscilloscope re-arms after each acquisition in a manner that minimizes or eliminates the likelihood of triggering at the beginning of a DDR burst. Randomizing the holdoff increases the likelihood that the oscilloscope will trigger on different data phases of a multiphase (8 data transfer) burst. This mode mixes up the traffic pattern the oscilloscope triggers on and is very effective when used on repeating patterns.

Query `:TRIGger:HOLDoff:MODE?`

The query returns the current holdoff mode.

Returned Format `[:TRIGger:HOLDoff:MODE] {FIXed | RANDom}<NL>`

:TRIGger:HTHReshold

Command `:TRIGger:HTHReshold CHANnel<N>,<level>`

This command specifies the high threshold voltage level for the selected trigger source. Set the high threshold level to a value considered to be a high level for your logic family; your data book gives two values, V_{IH} and V_{OH}.

<N> An integer, 1-4.

<level> A real number for the voltage level for the trigger source.

Query `:TRIGger:HTHReshold? CHANnel<N>`

The query returns the currently defined high threshold voltage level for the trigger source.

Returned Format `[:TRIGger:HTHReshold CHANnel<N>,] <level><NL>`

:TRIGger:HYSTeresis

Command `:TRIGger:HYSTeresis {NORMal | HSENsitivity}`

The :TRIGger:HYSTeresis command specifies the trigger hysteresis (noise reject) as either normal or high sensitivity. NORMal sensitivity adds hysteresis to the trigger circuitry for rejecting noise and should be used for waveforms of 4 GHz or below. HSENsitivity lowers the hysteresis of the trigger circuitry and should be used for waveforms of 4 GHz and above.

Query `:TRIGger:HYSTeresis?`

The query returns the current hysteresis setting.

Returned Format `[:TRIGger:HYSTeresis] {NORMal | HSENsitivity}<NL>`

:TRIGger:LEVel

Command `:TRIGger:LEVel {{CHANnel<N> | AUX},<level>}`

The :TRIGger:LEVel command specifies the trigger level on the specified channel for the trigger source. Only one trigger level is stored in the oscilloscope for each channel. This level applies to the channel throughout the trigger dialog.

<N> An integer, 1-4.

<level> A real number for the trigger level on the specified channel or Auxiliary Trigger Input.

Query `:TRIGger:LEVel? {CHANnel<N> | AUX}`

The query returns the specified channel's trigger level.

Returned Format `[:TRIGger:LEVel {CHANnel<N> | AUX},] <level><NL>`

:TRIGger:LTHReshold

Command `:TRIGger:LTHReshold CHANnel<N>,<level>`

This command specifies the low threshold voltage level for the selected trigger source. This command specifies the low threshold voltage level for the selected trigger source. Set the low threshold level to a value considered to be a low level for your logic family; your data book gives two values, V_{IL} and V_{OL}.

<N> An integer, 1-4.

<level> A real number for the voltage level for the trigger source.

Query `:TRIGger:LTHReshold? CHANnel<N>`

The query returns the currently defined low threshold for the trigger source.

Returned Format `[:TRIGger:LTHReshold CHANnel<N>,] <level><NL>`

:TRIGger:MODE

Command `:TRIGger:MODE {COMM | DELay | EDGE | GLITch | PATTern | PWIDth | RUNT | SEQuence | SHOLd | STATe | TIMeout | TRANsition | TV | WINDow | SBUS<N>}`

The :TRIGger:MODE command selects the trigger mode.

Table 26 Trigger Mode Settings

Mode	Definition
COMM	COMM mode lets you trigger on a serial pattern of bits in a waveform.
DELay	Delay by Events mode lets you view pulses in your waveform that occur a number of events after a specified waveform edge. Delay by Time mode lets you view pulses in your waveform that occur a long time after a specified waveform edge.
EDGE	Edge trigger mode.
GLITch	Trigger on a pulse that has a width less than a specified amount of time.
PATTern	Pattern triggering lets you trigger the oscilloscope using more than one channel as the trigger source. You can also use pattern triggering to trigger on a pulse of a given width.
PWIDth	PWIDth or pulse width trigger is used to find pulse that are either narrower or wider than pulse width and conditions that you specify.
RUNT	Runt trigger allows you to find pulses in your waveform that are shorter in amplitude than other pulses in your waveform.
SBUS<N>	Serial triggering on SBUS1, SBUS2, SBUS3, or SBUS4.
SEQuence	Sequential triggering allows you to use multiple events or time/pattern qualifications to define your trigger.
SHOLd	SHOLd or setup and hold time trigger lets you find violations of setup and hold times in your waveform.
STATe	State triggering lets you set the oscilloscope to use several channels as the trigger source, with one of the channels being used as a clock waveform.
TIMeout	Timeout trigger lets you find pulses that are high, low, or idle for too long a period of time.
TRANsition	Transition trigger lets you find pulses that have edges that are too slow or too fast in your waveform.

Table 26 Trigger Mode Settings (continued)

Mode	Definition
TV	TV trigger mode lets you trigger the oscilloscope on one of the standard television waveforms. You can also use this mode to trigger on a custom television waveform that you define.
WINDow	Window trigger lets you set a range of voltages (window) which can be used to determine when your waveform is inside or out side of the window.

Query `:TRIGger:MODE?`

The query returns the currently selected trigger mode.

Returned Format

```
[:TRIGger:MODE] {COMM | DELay | EDGE | GLITch | PATTern | PWIDth | RUNT
                | SEQuence | SHOLd | STATe | TIMeout | TRANsition | TV
                | WINDow | SBUS<N>}<NL>
```

:TRIGger:SWEep

Command `:TRIGger:SWEep {AUTO | TRIGgered | SINGle}`

The :TRIGger:SWEep command selects the oscilloscope sweep mode. New programs should use :RUN and :SINGle for run control and this command for AUTO and TRIGgered for sweep control. The SINGle sweep control should not be used.

AUTO When you select AUTO, if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 50 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.

TRIGgered When you select TRIGgered, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen.

SINGle When you select SINGle, if no trigger occurs, the oscilloscope will not sweep, and the previously acquired data will remain on the screen. Do not use in new programs.

Query `:TRIGger:SWEep?`

The query returns the specified channel's trigger level.

Returned Format `[:TRIGger:SWEep] {AUTO | TRIGgered}<NL>`

Comm Trigger Commands

- ":TRIGger:COMM:BWIDth" on page 900
- ":TRIGger:COMM:ENCode" on page 901
- ":TRIGger:COMM:PATTern" on page 902
- ":TRIGger:COMM:POLarity" on page 903
- ":TRIGger:COMM:SOURce" on page 904

:TRIGger:COMM:BWIDth

Command `:TRIGger:COMM:BWIDth <bwidth_value>`

The :TRIGger:COMM:BWIDth command is used to set the width of a bit for your waveform. The bit width is usually defined in the mask standard for your waveform.

<bwidth_value> A real number that represents the width of a bit.

Query `:TRIGger:COMM:BWIDth?`

The query returns the current bit width.

Returned Format `[:TRIGger:COMM:BWIDth] <bwidth_value><NL>`

:TRIGger:COMM:ENCode

Command `:TRIGger:COMM:ENCode {RZ | NRZ}`

This :TRIGger:COMM:ENCode command sets the type of waveform encoding for your waveform. You should use NRZ for CMI type waveforms and RZ for all other type of waveforms.

Query `:TRIGger:COMM:ENCode?`

The :TRIGger:COMM:ENCode? query returns the current value of encoding

Returned Format `[:TRIGger:COMM:ENCode] {RZ | NRZ}<NL>`

:TRIGger:COMM:PATTern

Command `:TRIGger:COMM:PATTern <bit>[,<bit[,<bit[,<bit[,<bit[,<bit]]]]]`

The :TRIGger:COMM:PATTern command sets the pattern used for triggering the oscilloscope when in communication trigger mode. The pattern can be up to 6 bits long. For NRZ type waveforms with positive polarity, there must be at least one logic 0 to logic 1 transition in the pattern. For NRZ waveforms with negative polarity there must be at least one logic 1 to logic 0 transition in the pattern. For RZ type waveforms the pattern must have at least one logic 1 bit for positive polarity. For RZ type waveforms the pattern must have at least one logic -1 bit for negative polarity.

<bit> A 1, -1, or 0.

Query `:TRIGger:COMM:PATTern?`

The :TRIGger:COMM:PATTern? query returns the current communication trigger pattern.

Returned Format `[:TRIGger:COMM:PATTern] <pattern><NL>`

<pattern> A string of up to 6 characters.

:TRIGger:COMM:POLarity

Command `:TRIGger:COMM:POLarity {POSitive | NEGative}`

The :TRIGger:COMM:POLarity command directly controls the trigger slope used for communication trigger. When set to a positive value, the rising edge of a pulse or waveform is used to trigger the oscilloscope. When set to a negative value, the falling edge of a pulse or waveform is used.

The polarity setting is also used to check for valid patterns. If you are trying to trigger on an isolated 1 pattern, you should set the polarity to positive. If you are trying to trigger on an isolated -1 pattern, you should set the polarity to negative.

Query `:TRIGger:COMM:POLarity?`

The :TRIGger:COMM:POLarity? query returns the current setting for polarity.

Returned Format `[:TRIGger:COMM:POLarity} {POSitive | NEGative}<NL>`

:TRIGger:COMM:SOURce

Command `:TRIGger:COMM:SOURce CHANnel<N>`

The :TRIGger:COMM:SOURce command selects the channel used for the communication trigger.

<N> An integer, 1-4.

Query `:TRIGger:COMM:SOURce?`

The :TRIGger:COMM:SOURce? query returns the currently selected communication trigger source.

Returned Format `[:TRIGger:COMM:SOURce] CHANnel<N><NL>`

Delay Trigger Commands

- ":TRIGger:DELay:ARM:SOURce" on page 906
- ":TRIGger:DELay:ARM:SLOPe" on page 907
- ":TRIGger:DELay:EDELay:COUNt" on page 908
- ":TRIGger:DELay:EDELay:SOURce" on page 909
- ":TRIGger:DELay:EDELay:SLOPe" on page 910
- ":TRIGger:DELay:MODE" on page 911
- ":TRIGger:DELay:TDELay:TIME" on page 912
- ":TRIGger:DELay:TRIGger:SOURce" on page 913
- ":TRIGger:DELay:TRIGger:SLOPe" on page 914

:TRIGger:DELay:ARM:SOURce

Command `:TRIGger:DELay:ARM:SOURce {CHANnel<N> | DIGital<M>}`

This command sets the Arm On source for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:DELay:ARM:SOURce?`

The query returns the currently defined Arm On source for the Delay trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:ARM:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:DELay:ARM:SLOPe

Command `:TRIGger:DELay:ARM:SLOPe {NEGative | POSitive}`

This command sets a positive or negative slope for arming the trigger circuitry when the oscilloscope is in the Delay trigger mode.

Query `:TRIGger:DELay:ARM:SLOPe?`

The query returns the currently defined slope for the Delay trigger mode.

Returned Format `[:TRIGger:DELay:ARM:SLOPe] {NEGative | POSitive}<NL>`

:TRIGger:DELay:EDELay:COUNt

Command `:TRIGger:DELay:EDELay:COUNt <edge_number>`

This command sets the event count for a Delay By Event trigger event.

<edge_num> An integer from 0 to 16,000,000 specifying the number of edges to delay.

Query `:TRIGger:DELay:EDELay:COUNt?`

The query returns the currently defined number of events to delay before triggering on the next Trigger On condition in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:COUNt] <edge_number><NL>`

:TRIGger:DELay:EDELay:SOURce

Command `:TRIGger:DELay:EDELay:SOURce {CHANnel<N> | DIGital<M>}`

This command sets the Event source for a Delay By Event trigger event.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:DELay:EDELay:SOURce?`

The query returns the currently defined Event source in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:DELay:EDELay:SLOPe

Command `:TRIGger:DELay:EDELay:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay By Event trigger event.

Query `:TRIGger:DELay:EDELay:SLOPe?`

The query returns the currently defined slope for an event in the Delay By Event trigger mode.

Returned Format `[:TRIGger:DELay:EDELay:SLOPe] {NEGative | POSitive}<NL>`

:TRIGger:DELay:MODE

Command `:TRIGger:DELay:MODE {EDELay | TDELay}`

The :TRIGger:DELay:MODE command selects the type of delay trigger mode to either events or to time.

Query `:TRIGger:DELay:MODE?`

The query returns the currently selected delay trigger mode.

Returned Format `[:TRIGger:DELay:MODE] {EDELay | TDELay}<NL>`

:TRIGger:DELay:TDELay:TIME

Command `:TRIGger:DELay:TDELay:TIME <delay>`

This command sets the delay for a Delay By Time trigger event.

<delay> Time, in seconds, set for the delay trigger, from 10 ns to 10 s.

Query `:TRIGger:DELay:TDELay:TIME?`

The query returns the currently defined time delay before triggering on the next Trigger On condition in the Delay By Time trigger mode.

Returned Format `[:TRIGger:DELay:TDELay:TIME] <delay><NL>`

:TRIGger:DELay:TRIGger:SOURce

Command `:TRIGger:DELay:TRIGger:SOURce {CHANnel<N> | DIGital<M>}`

This command sets the Trigger On source for a Delay trigger event.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:DELay:TRIGger:SOURce?`

The query returns the currently defined Trigger On source in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:DELay:TRIGger:SLOPe

Command `:TRIGger:DELay:TRIGger:SLOPe {NEGative | POSitive}`

This command sets the trigger slope for the Delay trigger event.

Query `:TRIGger:DELay:TRIGger:SLOPe?`

The query returns the currently defined slope for an event in the Delay trigger mode.

Returned Format `[:TRIGger:DELay:TRIGger:SLOPe] {NEGative | POSitive}<NL>`

Edge Trigger Commands

- ":TRIGger:EDGE:SLOPe" on page 916
- ":TRIGger:EDGE:SOURce" on page 917

:TRIGger:EDGE:SLOPe

Command `:TRIGger:EDGE[{1 | 2}]:SLOPe {POSitive | NEGative | EITHer}`

The :TRIGger:EDGE:SLOPe command sets the slope of the trigger source previously selected by the :TRIGger:EDGE:SOURce command. The LINE source has no slope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:EDGE:SLOPe?`

The query returns the currently selected slope for the specified edge trigger source.

Returned Format `[:TRIGger:EDGE:SLOPe] {POSitive | NEGative | EITHer}<NL>`

:TRIGger:EDGE:SOURce

Command `:TRIGger:EDGE[{1 | 2}]:SOURce {CHANnel<N> | DIGital<M> | AUX | LINE}`

The :TRIGger:EDGE:SOURce command selects the source for edge mode triggering. This is the source that will be used for subsequent :TRIGger:EDGE:SLOPe commands or queries.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:EDGE:SOURce?`

The query returns the currently selected edge mode trigger source.

Returned Format `[:TRIGger:EDGE:SOURce] {CHANnel<N> | DIGital<M> | AUX | LINE}<NL>`

Glitch Trigger Commands

- ":TRIGger:GLITch:POLarity" on page 919
- ":TRIGger:GLITch:SOURce" on page 920
- ":TRIGger:GLITch:WIDTh" on page 921

:TRIGger:GLITch:POLarity

Command `:TRIGger:GLITch[{1 | 2}]:POLarity {POSitive | NEGative}`

This command defines the polarity of the glitch as positive or negative. The trigger source must be set using the :TRIGger:GLITch:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:GLITch:POLarity?`

The query returns the currently selected glitch polarity.

Returned Format `[:TRIGger:GLITch:POLarity] {POS | NEG}<NL>`

:TRIGger:GLITch:SOURce

Command `:TRIGger:GLITch[{1 | 2}]:SOURce {CHANnel<N> | DIGital<M>}`

This command sets the source for the glitch trigger mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:GLITch:SOURce?`

The query returns the currently selected source for the glitch trigger mode.

Returned Format `[:TRIGger:GLITch:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:GLITch:WIDTh

Command `:TRIGger:GLITch[{1 | 2}]:WIDTh <width>`

This command sets the glitch width. The oscilloscope will trigger on a pulse that has a width less than the specified width.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<width> A real number for the glitch width, ranging from 250 ps to 10 s.

Query `:TRIGger:GLITch:WIDTh?`

The query returns the currently specified glitch width.

Returned Format `[:TRIGger:GLITch:WIDTh] <width><NL>`

Pattern Trigger Commands

- ":TRIGger:PATTern:CONDition" on page 923
- ":TRIGger:PATTern:LOGic" on page 924

:TRIGger:PATTern:CONDition

Command `:TRIGger:PATTern[{1 | 2}]:CONDition {ENTered | EXITed | {GT,<time>[,PEXits|TIMeout]} | {LT,<time>} | {RANGe,<gt_time>,<lt_time>}}`

This command describes the condition applied to the trigger pattern to actually generate a trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<gt_time> The minimum time (greater than time) for the trigger pattern, from 10 ns to 9.999 s.

<lt_time> The maximum time (less than time) for the trigger pattern, from 10.5 ps to 10 s.

<time> The time condition, in seconds, for the pattern trigger, from 100 ps to 10 s.

When using the GT (Present >) parameter, the PEXits (Pattern Exits) or the TIMeout parameter controls when the trigger is generated.

Query `:TRIGger:PATTern:CONDition?`

The query returns the currently defined trigger condition.

Returned Format `[:TRIGger:PATTern:CONDition] {ENTered|EXITed | {GT,<time>[,PEXits|TIMeout]} | {LT,<time>} | {RANGe,<gt_time>,<lt_time>}}<NL>`

:TRIGger:PATTern:LOGic

Command `:TRIGger:PATTern[{1 | 2}]:LOGic {CHANnel<N> | DIGital<M>},`
`{HIGH | LOW | DONTcare | RISing | FALLing}`

This command defines the logic criteria for a selected channel.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:PATTern:LOGic? {CHANnel<N> | DIGital<M>}`

The query returns the current logic criteria for a selected channel.

Returned Format `[:TRIGger:PATTern:LOGic {CHANnel<N> | DIGital<M>},]`
`{HIGH | LOW | DONTcare | RISing | FALLing}<NL>`

Pulse Width Trigger Commands

- ":TRIGger:PWIDth:DIRection" on page 926
- ":TRIGger:PWIDth:POLarity" on page 927
- ":TRIGger:PWIDth:SOURce" on page 928
- ":TRIGger:PWIDth:TPOint" on page 929
- ":TRIGger:PWIDth:WIDTh" on page 930

:TRIGger:PWIDth:DIRection

Command `:TRIGger:PWIDth[{1 | 2}]:DIRection {GTHan | LTHan}`

This command specifies whether a pulse must be wider or narrower than the width value to trigger the oscilloscope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:DIRection?`

The query returns the currently defined direction for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:DIRection] {GTHan | LTHan}<NL>`

:TRIGger:PWIDth:POLarity

Command `:TRIGger:PWIDth[{1 | 2}]:POLarity {NEGative | POSitive}`

This command specifies the pulse polarity that the oscilloscope uses to determine a pulse width violation. For a negative polarity pulse, the oscilloscope triggers when the rising edge of a pulse crosses the trigger level. For a positive polarity pulse, the oscilloscope triggers when the falling edge of a pulse crosses the trigger level.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:POLarity?`

The query returns the currently defined polarity for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:POLarity] {NEGative | POSitive}<NL>`

:TRIGger:PWIDth:SOURce

Command `:TRIGger:PWIDth[{1 | 2}]:SOURce {CHANnel<N> | DIGital<M>}`

This command specifies the channel source used to trigger the oscilloscope with the pulse width trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:PWIDth:SOURce?`

The query returns the currently defined channel source for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:SOURce] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:PWIDth:TPOint

Command `:TRIGger:PWIDth[{1 | 2}]:TPOint {EPULse | TIMeout}`

This command specifies whether the pulse width trigger should occur at the end of the pulse or at a specified timeout period. This command is only available if the pulse direction is set to GTHan.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:PWIDth:TPOint?`

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:TPOint] {EPULse | TIMeout}<NL>`

:TRIGger:PWIDth:WIDTh

Command `:TRIGger:PWIDth[{1 | 2}]:WIDTh <width>`

This command specifies how wide a pulse must be to trigger the oscilloscope.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<width> Pulse width, which can range from 250 ps to 10 s.

Query `:TRIGger:PWIDth:WIDTh?`

The query returns the currently defined width for the pulse.

Returned Format `[:TRIGger:PWIDth:WIDTh] <width><NL>`

Runt Trigger Commands

- ":TRIGger:RUNT:POLarity" on page 932
- ":TRIGger:RUNT:QUALified" on page 933
- ":TRIGger:RUNT:SOURce" on page 934
- ":TRIGger:RUNT:TIME" on page 935

:TRIGger:RUNT:POLarity

Command `:TRIGger:RUNT[{1 | 2}]:POLarity {POSitive | NEGative}`

This command defines the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:RUNT:POLarity?`

The query returns the currently selected runt pulse polarity.

Returned Format `[:TRIGger:RUNT:POLarity] {POSitive | NEGative}<NL>`

:TRIGger:RUNT:QUALified

Command `:TRIGger:RUNT[{1 | 2}]:QUALified {{ON | 1} | {OFF | 0}}`

This command enables the time qualified runt pulse feature the polarity of the runt pulse as positive or negative. The trigger source must be set using the :TRIGger:RUNT:SOURce command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:RUNT:QUALified?`

The query returns the current state of the time qualified runt pulse feature.

Returned Format `[:TRIGger:RUNT:QUALified] {1 | 0}<NL>`

:TRIGger:RUNT:SOURce

Command `:TRIGger:RUNT[{1 | 2}]:SOURce CHANnel<N>`

This command sets the source for the runt trigger mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:RUNT:SOURce?`

The query returns the currently selected source for the runt trigger mode.

Returned Format `[:TRIGger:RUNT:SOURce] CHANnel<N><NL>`

:TRIGger:RUNT:TIME

Command `:TRIGger:RUNT[{1 | 2}]:TIME <time>`

This command sets the time qualifier. The oscilloscope will trigger on a runt pulse that has a width greater than the specified time.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> A real number for the time greater than qualifier, ranging from 250 ps to 30 ns.

Query `:TRIGger:RUNT:TIME?`

The query returns the currently specified glitch width.

Returned Format `[:TRIGger:RUNT:TIME] <time><NL>`

Sequence Trigger Commands

- ":TRIGger:SEQuence:TERM1" on page 937
- ":TRIGger:SEQuence:TERM2" on page 938
- ":TRIGger:SEQuence:RESet:ENABle" on page 939
- ":TRIGger:SEQuence:RESet:TYPE" on page 940
- ":TRIGger:SEQuence:RESet:EVENt" on page 941
- ":TRIGger:SEQuence:RESet:TIME" on page 942
- ":TRIGger:SEQuence:WAIT:ENABle" on page 943
- ":TRIGger:SEQuence:WAIT:TIME" on page 944

:TRIGger:SEQuence:TERM1

Command `:TRIGger:SEQuence:TERM1 { EDGE1 | GLITch1 | PWIDth1 | RUNT1 | SHOLD1 | STATe1 | TIMeout1 | TRANsition1 | WINDow1 }`

This command specifies the trigger mode for the TERM1 state in the sequential trigger (the Find (A) state in the Trigger Setup dialog box on the oscilloscope).

There are five limitations associated with sequential triggering:

1. The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.
2. The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.
3. The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.
4. You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.
5. The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query `:TRIGger:SEQuence:TERM1?`

The query returns the currently defined trigger mode for the TERM1 state.

:TRIGger:SEQuence:TERM2

Command `:TRIGger:SEQuence:TERM2 { EDGE2 | GLITch2 | PWIDth2 | RUNT2 | SHOLD2 | STATe2 | TIMeout2 | TRANsition2 | WINDow2 }`

This command specifies the trigger mode for the TERM2 state in the sequential trigger (the Trigger (B) state in the Trigger Setup dialog box on the oscilloscope).

There are five limitations associated with sequential triggering:

1. The Edge followed by Edge and Video trigger modes cannot be used in sequential triggering.
2. The AND qualifier cannot be used when the Reset condition is based upon a logical pattern.
3. The Pattern/State trigger mode that uses range as the When Pattern selection can only be used for either the Term1 state or the Term2 state, but not both.
4. You can only use one long timer (>30 ns). Therefore, trigger modes that use timers greater than 30 ns can only be used for either the Term1 state or the Term2 state, but not both. Some examples of trigger modes where you might use a long timer include Pulse Width, Glitch, Window, Edge Transition, and Timeout.
5. The alternating edge trigger mode cannot be used in sequential triggering.

Limitations (3) and (4) deal with extended resources. Extended resources refer to trigger modes or conditions that are only available to either the Term1 state or the Term2 state, but not both at the same time. The oscilloscope will figure out which state has access to these extended resources based upon the conditions you setup in each of these states. If you want Term2 to have a timer longer than 30 ns, you must first change the timer associated with Term1 to be less than 30 ns.

Query `:TRIGger:SEQuence:TERM2?`

The query returns the currently defined trigger mode for the TERM2 state.

:TRIGger:SEQuence:RESet:ENABle

Command `:TRIGger:SEQuence:RESet:ENABle {{ON | 1} | {OFF | 0}}`

This command turns the Reset feature on or off for the sequential trigger.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

If the Reset feature is enabled, the AND qualifier cannot be used for the TERM1 state.

Query `:TRIGger:SEQuence:RESet:ENABle?`

The query returns whether or not the Reset feature is enabled.

:TRIGger:SEQuence:RESet:TYPE

Command `:TRIGger:SEQuence:RESet:TYPE { TIME | EVENt }`

This command specifies whether the Reset condition is based upon a length of time or a logical pattern.

The Reset feature allows you to specify a length of time such that if this time is exceeded between when the TERM1 event occurs and when the TERM2 event occurs, the sequential trigger is reset and the oscilloscope returns to looking for the TERM1 event without triggering. If the Delay feature (remote command :WAIT) is used as well then the Reset timer does not start counting down until after the delay period is complete.

You can also base the Reset condition on a logical pattern. If the specified pattern is found between when the TERM1 occurs and the TERM2 event occurs, the sequential trigger resets and goes back to looking for the TERM1 event without triggering. The delay feature does not impact a logical pattern Reset as the pattern is searched for immediately after the TERM1 event occurs regardless of whether of not the Delay period is complete.

Query `:TRIGger:SEQuence:RESet:TYPE?`

The query returns whether the Reset condition is based upon a length of time or an event.

:TRIGger:SEQuence:RESet:EVENt

Command `:TRIGger:SEQuence:RESet:EVENt {CHANnel<N>}, { HIGH | LOW | DONTcare }`

This command defines the logical pattern used for an event Reset condition.

You can specify for each channel (1-4) whether you want the value to be high (1), low (0), or you don't care (X).

<N> An integer, 1-4.

:TRIGger:SEQuence:RESet:TIME

Command `:TRIGger:SEQuence:RESet:TIME <time>`

This command defines the length of time to use for the time-based Reset condition.

<time> A length of time in seconds.

Query `:TRIGger:SEQuence:RESet:TIME?`

The query returns the length of time used for the Reset condition.

:TRIGger:SEQuence:WAIT:ENABle

Command `:TRIGger:SEQuence:WAIT:ENABle { {ON|1} | {OFF|0}}`

This command turns the Delay feature on or off for the sequential trigger.

The Delay feature allows you to define a length of time for the sequential trigger system to wait after the TERM1 event occurs before it starts searching for the TERM2 event.

Query `:TRIGger:SEQuence:RESet:ENABle?`

The query returns whether or not the Delay feature is turned on.

:TRIGger:SEQuence:WAIT:TIME

Command `:TRIGger:SEQuence:WAIT:TIME <time>`

This command defines the length of time to use for the Delay condition.

<time> A length of time in seconds.

Query `:TRIGger:SEQuence:WAIT:TIME?`

The query returns the length of time used for the Delay condition.

Setup and Hold Trigger Commands

- ":TRIGger:SHOLd:CSOurce" on page 946
- ":TRIGger:SHOLd:CSOurce:EDGE" on page 947
- ":TRIGger:SHOLd:DSOurce" on page 948
- ":TRIGger:SHOLd:HoldTIMe (HTIMe)" on page 949
- ":TRIGger:SHOLd:MODE" on page 950
- ":TRIGger:SHOLd:SetupTIMe" on page 951

:TRIGger:SHOLd:CSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:CSOurce CHANnel<N>`

This command specifies the clock source for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:SHOLd:CSOurce?`

The query returns the currently defined clock source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:CSOurce] CHANnel<N><NL>`

:TRIGger:SHOLd:CSOurce:EDGE

Command `:TRIGger:SHOLd[{1 | 2}]:CSOurce:EDGE {RISing | FALLing}`

This command specifies the clock source trigger edge for the clock used for the trigger setup and hold violation. The clock must pass through the voltage level you have set before the trigger circuitry looks for a setup and hold time violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:SHOLd:CSOurce:EDGE?`

The query returns the currently defined clock source edge for the trigger setup and hold violation level for the clock source.

Returned Format [:TRIGger:SHOLd:CSOurce:EDGE] {RISing | FALLing}<NL>

:TRIGger:SHOLd:DSOurce

Command `:TRIGger:SHOLd[{1 | 2}]:DSOurce CHANnel<N>`

The data source commands specify the data source for the trigger setup and hold violation.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:SHOLd:DSOurce?`

The query returns the currently defined data source for the trigger setup and hold violation.

Returned Format `[:TRIGger:SHOLd:DSOurce] CHANnel<N><NL>`

:TRIGger:SHOLd:HoldTIMe (HTIMe)

Command `:TRIGger:SHOLd[{1 | 2}]:HoldTIMe <time>`

This command specifies the amount of hold time used to test for both a setup and hold trigger violation. The hold time is the amount of time that the data must be stable and valid after a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> Hold time, in seconds.

Query `:TRIGger:SHOLD:HoldTIMe?`

The query returns the currently defined hold time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLD:HoldTIMe] <time><NL>`

:TRIGger:SHOLd:MODE

Command `:TRIGger:SHOLd[{1 | 2}]:MODE {SETup | HOLD | SHOLd}`

SETup When using the setup time mode, a time window is defined where the right edge is the clock edge and the left edge is the selected time before the clock edge. The waveform must stay outside of the trigger level thresholds during this time window. If the waveform crosses a threshold during this time window, a violation event occurs and the oscilloscope triggers.

HOLD When using the hold time mode, the waveform must not cross the threshold voltages after the specified clock edge for at least the hold time you have selected. Otherwise, a violation event occurs and the oscilloscope triggers.

SHOLd When using the setup and hold time mode, if the waveform violates either a setup time or hold time, the oscilloscope triggers. The total time allowed for the sum of setup time plus hold time is 24 ns maximum.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:SHOLd:MODE?`

The query returns the currently selected trigger setup violation mode.

Returned Format `[:TRIGger:SHOLd:MODE] {SETup | HOLD | SHOLd}<NL>`

:TRIGger:SHOLd:SetupTIMe

Command `:TRIGger:SHOLd[{1 | 2}]:SetupTIMe <time>`

This command specifies the amount of setup time used to test for both a setup and hold trigger violation. The setup time is the amount of time that the data must be stable and valid before a clock edge.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> Setup time, in seconds.

Query `:TRIGger:SHOLd:SetupTIMe?`

The query returns the currently defined setup time for the setup and hold trigger violation.

Returned Format `[:TRIGger:SHOLd:SetupTIMe] <time><NL>`

State Trigger Commands

- ":TRIGger:STATe:CLOCk" on page 953
- ":TRIGger:STATe:LOGic" on page 954
- ":TRIGger:STATe:LTYPe" on page 955
- ":TRIGger:STATe:SLOPe" on page 956

:TRIGger:STATe:CLOCk

Command `:TRIGger:STATe[{1 | 2}]:CLOCk {CHANnel<N> | DIGital<M>}`

This command selects the source for the clock waveform in the State Trigger Mode.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:STATe:CLOCk?`

The query returns the currently selected clock source.

Returned Format `[:TRIGger:STATe:CLOCk] {CHANnel<N> | DIGital<M>}<NL>`

:TRIGger:STATe:LOGic

Command `:TRIGger:STATe[{1 | 2}]:LOGic {CHANnel<N> | DIGital<M>}, {LOW | HIGH | DONTcare | RISing | FALLing}`

This command defines the logic state of the specified source for the state pattern. The command produces a settings conflict on a channel that has been defined as the clock.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

<M> An integer, 0-15. Digital channels are available on mixed-signal oscilloscopes.

Query `:TRIGger:STATe:LOGic? {CHANnel<N> | DIGital<M>}`

The query returns the logic state definition for the specified source.

Returned Format `[:TRIGger:STATe:LOGic {CHANnel<N> | DIGital<M>},] {LOW | HIGH | DONTcare | RISing | FALLing}<NL>`

:TRIGger:STATe:LTYPe

Command `:TRIGger:STATe[{1 | 2}]:LTYPe {AND | NAND}`

This command defines the state trigger logic type. If the logic type is set to AND, then a trigger is generated on the edge of the clock when the input waveforms match the pattern specified by the :TRIGger:STATe:LOGic command. If the logic type is set to NAND, then a trigger is generated on the edge of the clock when the input waveforms do not match the specified pattern.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:STATe:LTYPe?`

The query returns the currently specified state trigger logic type.

Returned Format `[:TRIGger:STATe:LTYPe] {AND | NAND}<NL>`

:TRIGger:STATe:SLOPe

Command `:TRIGger:STATe[{1 | 2}]:SLOPe {RISing | FALLing | EITHer}`

This command specifies the edge of the clock that is used to generate a trigger. The waveform source used for the clock is selected by using the :TRIGger:STATe:CLOCk command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:STATe:SLOPe?`

The query returns the currently defined slope for the clock in State Trigger Mode.

Returned Format `[:TRIGger:STATe:SLOPe] {RISing | FALLing | EITHer}<NL>`

Timeout Trigger Commands

- ":TRIGger:TIMeout:CONDition" on page 958
- ":TRIGger:TIMeout:SOURce" on page 959
- ":TRIGger:TIMeout:TIME" on page 960

:TRIGger:TIMeout:CONDition

Command `:TRIGger:TIMeout[{1 | 2}]:CONDition {HIGH | LOW | UNCHanged}`

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

This command sets the condition used for the timeout trigger.

HIGH Trigger when the waveform has been high for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

LOW Trigger when the waveform has been low for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

UNCHanged Trigger when the waveform has not changed state for a period time longer than the time value which is set by the TRIGger:TIMeout:TIME command.

Query `:TRIGger:TIMeout:CONDition?`

The query returns the currently defined trigger condition for the timeout trigger.

Returned Format `[:TRIGger:TIMeout:CONDition] {HIGH | LOW | UNCHanged}<NL>`

:TRIGger:TIMeout:SOURce

Command `:TRIGger:TIMeout[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the timeout trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:TIMeout:SOURce?`

The query returns the currently defined channel source for the timeout trigger.

Returned Format `[:TRIGger:TIMeout:SOURce] CHANnel<N><NL>`

:TRIGger:TIMeout:TIME

Command `:TRIGger:TIMeout[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the timeout trigger, in seconds.

Query `:TRIGger:TIMeout:TIME?`

The query returns the currently defined time for the trigger trigger.

Returned Format `[:TRIGger:TIMeout:TIME] <time><NL>`

Transition Trigger Commands

- ":TRIGger:TRANsition:DIRection" on page 962
- ":TRIGger:TRANsition:SOURce" on page 963
- ":TRIGger:TRANsition:TIME" on page 964
- ":TRIGger:TRANsition:TYPE" on page 965

:TRIGger:TRANsition:DIRection

Command `:TRIGger:TRANsition[{1 | 2}]:DIRection {GTHan | LTHan}`

This command lets you look for transition violations that are greater than or less than the time specified by the :TRIGger:TRANsition:TIME command.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:TRANsition:DIRection?`

The query returns the currently defined direction for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:DIRection] {GTHan | LTHan}]<NL>`

:TRIGger:TRANsition:SOURce

Command `:TRIGger:TRANsition[{1 | 2}]:SOURce CHANnel<N>`

The transition source command lets you find any edge in your waveform that violates a rise time or fall time specification. The oscilloscope finds a transition violation trigger by looking for any pulses in your waveform with rising or falling edges that do not cross two voltage levels in the amount of time you have specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:TRANsition:SOURce?`

The query returns the currently defined transition source for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:SOURce] CHANnel<N><NL>`

:TRIGger:TRANsition:TIME

Command `:TRIGger:TRANsition[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the trigger violation transition, in seconds.

Query `:TRIGger:TRANsition:TIME?`

The query returns the currently defined time for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:TIME] <time><NL>`

:TRIGger:TRANsition:TYPE

Command `:TRIGger:TRANsition[{1 | 2}]:TYPE {RISetime | FALLtime}`

This command lets you select either a rise time or fall time transition violation trigger event.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:TRANsition:TYPE?`

The query returns the currently defined transition type for the trigger transition violation.

Returned Format `[:TRIGger:TRANsition:TYPE] {RISetime | FALLtime}<NL>`

TV Trigger Commands

- ":TRIGger:TV:LINE" on page 967
- ":TRIGger:TV:MODE" on page 968
- ":TRIGger:TV:POLarity" on page 969
- ":TRIGger:TV:SOURce" on page 970
- ":TRIGger:TV:STANdard" on page 971
- ":TRIGger:TV:UDTV:ENUMber" on page 972
- ":TRIGger:TV:UDTV:HSYNc" on page 973
- ":TRIGger:TV:UDTV:HTIMe" on page 974
- ":TRIGger:TV:UDTV:PGTHan" on page 975
- ":TRIGger:TV:UDTV:POLarity" on page 976

:TRIGger:TV:LINE

Command `:TRIGger:TV:LINE <line_number>`

The :TRIGger:TV:LINE command selects the horizontal line that you want to examine. The allowable line number entry depends on the :TRIGger:TV:MODE selected. Once the vertical sync pulse of the selected field is received, the trigger is delayed by the number of lines specified.

<line_number> Horizontal line number as shown below.

Video Standard	Field 1	Field 2	Alternate Field
NTSC/PAL-M	1 to 263	1 to 262	1 to 262
PAL/SECAM	1 to 313	314 to 625	1 to 312

EDTV/HDTV	Line numbers
EDTV 480p/60	1 to 525
EDTV 576p/50	1 to 625
HDTV 720p/60	1 to 750
HDTV 720p/50	1 to 750
HDTV 1080i/60	1 to 1125
HDTV 1080i/50	1 to 1125
HDTV1080p/60	1 to 1125
HDTV 1080p/50	1 to 1125
HDTV 1080p/30	1 to 1125
HDTV 1080p/25	1 to 1125
HDTV 1080p/24	1 to 1125

Query `:TRIGger:TV:LINE?`

The query returns the current line number.

Returned Format `[:TRIGger:TV:LINE] <line_number><NL>`

:TRIGger:TV:MODE

Command `:TRIGger:TV:MODE {AFIelds | ALINes | ALTernate | FIElds1 | FIElds2 | LINE}`

The :TRIGger:TV:MODE command determines which portion of the video waveform is used to trigger.

Query `:TRIGger:TV:MODE?`

The query returns the current TV trigger mode.

Returned Format `[:TRIGger:TV:MODE] {AFIelds | ALINes | ALTernate | FIElds1 | FIElds2 | LINE}<NL>`

:TRIGger:TV:POLarity

Command `:TRIGger:TV:POLarity {NEGative | POSitive}`

The :TRIGger:TV:POLarity command specifies the vertical sync pulse polarity for the selected field used during TV mode triggering.

Query `:TRIGger:TV:POLarity?`

The query returns the currently selected sync pulse polarity.

Returned Format `[:TRIGger:TV:POLarity] {NEGative | POSitive}<NL>`

:TRIGger:TV:SOURce

Command `:TRIGger:TV:SOURce CHANnel<N>`

The :TRIGger:TV:SOURce command selects the source for the TV mode triggering. This is the source that will be used for subsequent :TRIGger:TV commands and queries.

<N> An integer, 1-4.

Query `:TRIGger:TV:SOURce?`

The query returns the currently selected standard TV trigger mode source.

Returned Format `[:TRIGger:TV:SOURce] CHANnel<N>NL>`

:TRIGger:TV:STANdard

Command

```
:TRIGger:TV:STANdard {I1080L60HZ | I1080L50HZ | L525 | L625 | P480L60HZ
    | P576L50HZ | P720L60HZ | P720L50HZ | P1080L60HZ | P1080L50HZ
    | P1080L30HZ | P1080L25HZ | P1080L24HZ | UDTV}
```

The TRIGger:TV:STANdard command sets triggering to one of the standard video types. There is also a user defined TV type that can be used to set the triggering to one of the non-standard types of video.

Query

```
:TRIGger:TV:STANdard?
```

The query returns the currently selected video standard.

Returned Format

```
[:TRIGger:TV:STANdard] {I1080L60HZ | I1080L50HZ | L525 | L625
    | P480L60HZ | P576L50HZ | P720L60HZ | P720L50HZ | P1080L60HZ
    | P1080L50HZ | P1080L40HZ | P1080L30HZ | P1080L25HZ | P1080L24HZ
    | UDTV}<NL>
```

:TRIGger:TV:UDTV:ENUMber

Command `:TRIGger:TV:UDTV:ENUMber <count>`

The :TRIGger:TV:UDTV:ENUMber command specifies the number of events (horizontal sync pulses) to delay after arming the trigger before looking for the trigger event. Specify conditions for arming the trigger using:

TRIGger:TV:UDTV:PGTHan, and

TRIGger:TV:UDTV:POLarity.

<count> An integer for the number of events to delay. Allowable values range from 1 to 16,000,000.

Query `:TRIGger:TV:UDTV:ENUMber?`

The query returns the currently programmed count value.

Returned Format `[:TRIGger:TV:UDTV:ENUMber] <count><NL>`

:TRIGger:TV:UDTV:HSYNc

Command `:TRIGger:TV:UDTV:HSYNc {ON | 1} | {OFF | 0}}`

This command enables the horizontal sync mode of triggering.

Query `:TRIGger:TV:UDTV:HSYNc?`

The query returns the current state of the horizontal sync mode of triggering.

Returned Format `[:TRIGger:TV:UDTV:HSYNc] {1 | 0}<NL>`

:TRIGger:TV:UDTV:HTIMe

Command `:TRIGger:TV:UDTV:HTIMe <time>`

The :TRIGger:TV:UDTV:HTIMe command sets the time that a sync pulse must be present to be considered a valid sync pulse.

<time> A real number that is the time width for the sync pulse.

Query `:TRIGger:TV:UDTV:HTIMe?`

The query returns the currently defined time for the sync pulse width.

Returned Format `[:TRIGger:TV:UDTV:HTIMe] <time><NL>`

:TRIGger:TV:UDTV:PGTHan

Command `:TRIGger:TV:UDTV:PGTHan <lower_limit>`

The :TRIGger:TV:UDTV:PGTHan (Present Greater THan) command specifies the minimum pulse width of the waveform used to arm the trigger used during user-defined trigger mode.

<lower_limit> Minimum pulse width (time >), from 5 ns to 9.9999999 s.

Query `:TRIGger:TV:UDTV:PGTHan?`

The query returns the currently selected minimum pulse width.

Returned Format `[:TRIGger:TV:UDTV:PGTHan] <lower_limit><NL>`

:TRIGger:TV:UDTV:POLarity

Command `:TRIGger:TV:UDTV:POLarity {NEGative | POSitive}`

The :TRIGger:TV:UDTV:POLarity command specifies the polarity for the sync pulse used to arm the trigger in the user-defined trigger mode.

Query `:TRIGger:TV:UDTV:POLarity?`

The query returns the currently selected UDTV sync pulse polarity.

Returned Format `[:TRIGger:TV:UDTV:POLarity] {NEGative | POSitive}<NL>`

Window Trigger Commands

- ":TRIGger:WINDow:CONDition" on page 978
- ":TRIGger:WINDow:SOURce" on page 979
- ":TRIGger:WINDow:TIME" on page 980
- ":TRIGger:WINDow:TPOint" on page 981

:TRIGger:WINDow:CONDition

Command

```
:TRIGger:WINDow[{1 | 2}]:CONDition {ENTer | EXIT |
INSide [,{GTHan | LTHan}] |
OUTSide [,{GTHan | LTHan}]}
```

This command describes the condition applied to the trigger window to actually generate a trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query

```
:TRIGger:WINDow:CONDition?
```

The query returns the currently defined trigger condition.

Returned Format

```
[:TRIGger:WINDow:CONDition] {ENTer | EXIT |
INSide,{GTHan | LTHan} | OUTSide,{GTHan | LTHan}}<NL>
```

:TRIGger:WINDow:SOURce

Command `:TRIGger:WINDow[{1 | 2}]:SOURce CHANnel<N>`

This command specifies the channel source used to trigger the oscilloscope with the window trigger.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<N> An integer, 1-4.

Query `:TRIGger:WINDow:SOURce?`

The query returns the currently defined channel source for the window trigger.

Returned Format `[:TRIGger:WINDow:SOURce] CHANnel<N><NL>`

:TRIGger:WINDow:TIME

Command `:TRIGger:WINDow[{1 | 2}]:TIME <time>`

This command lets you look for transition violations that are greater than or less than the time specified.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

<time> The time for the trigger violation transition, in seconds.

Query `:TRIGger:WINDow:TIME?`

The query returns the currently defined time for the trigger window timeout.

Returned Format `[:TRIGger:WINDow:TIME] <time><NL>`

:TRIGger:WINDow:TPOint

Command `:TRIGger:WINDow[{1 | 2}]:TPOint {BOUNdary | TIMeout}`

This command specifies whether the window trigger should occur at the boundary of the window or at a specified timeout period.

The optional [{1 | 2}] parameter sets whether the trigger mode goes with the TERM1 or TERM2 state if sequential triggering is being used.

Query `:TRIGger:WINDow:TPOint?`

The query returns the currently defined trigger on point for the pulse width trigger.

Returned Format `[:TRIGger:PWIDth:TPOint] {BOUNdary | TIMeout}<NL>`

34 Waveform Commands

:WAVeform:BANDpass? 986
:WAVeform:BYTeorder 987
:WAVeform:COMPlete? 988
:WAVeform:COUNt? 989
:WAVeform:COUPling? 990
:WAVeform:DATA? 991
:WAVeform:FORMat 1013
:WAVeform:POINts? 1016
:WAVeform:PREamble? 1017
:WAVeform:SEGMented:ALL 1022
:WAVeform:SEGMented:COUNt? 1023
:WAVeform:SEGMented:TTAG? 1024
:WAVeform:SEGMented:XLISt? 1025
:WAVeform:SOURce 1026
:WAVeform:STReaming 1027
:WAVeform:TYPE? 1028
:WAVeform:VIEW 1029
:WAVeform:XDISplay? 1031
:WAVeform:XINCrement? 1032
:WAVeform:XORigin? 1033
:WAVeform:XRANge? 1034
:WAVeform:XREFerence? 1035
:WAVeform:XUNits? 1036
:WAVeform:YDISplay? 1037
:WAVeform:YINCrement? 1038
:WAVeform:YORigin? 1039
:WAVeform:YRANge? 1040
:WAVeform:YREFerence? 1041
:WAVeform:YUNits? 1042

The WAVeform subsystem is used to transfer waveform data between a computer and the oscilloscope. It contains commands to set up the waveform transfer and to send or receive waveform records to or from the oscilloscope.

Data Acquisition

When data is acquired using the DIGitize command, the data is placed in the channel or function memory of the specified source. After the DIGitize command executes, the oscilloscope is stopped. If the oscilloscope is restarted by your program or from the front panel, the data acquired with the DIGitize command is overwritten.

You can query the preamble, elements of the preamble, or waveform data while the oscilloscope is running, but the data will reflect only the current acquisition, and subsequent queries will not reflect consistent data. For example, if the oscilloscope is running and you query the X origin, the data is queried in a separate command, it is likely that the first point in the data will have a different time than that of the X origin. This is due to data acquisitions that may have occurred between the queries. For this reason, Agilent Technologies does not recommend this mode of operation. Instead, you should use the DIGitize command to stop the oscilloscope so that all subsequent queries will be consistent.

NOTE Function and channel data are volatile and must be read following a DIGitize command or the data will be lost when the oscilloscope is turned off.

Waveform Data and Preamble

The waveform record consists of two parts: the preamble and the waveform data. The waveform data is the actual sampled data acquired for the specified source. The preamble contains the information for interpreting the waveform data, including the number of points acquired, the format of the acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data.

The values in the preamble are set when you execute the DIGitize command. The preamble values are based on the current settings of the oscilloscope's controls.

Data Conversion

Data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y origins and X and Y increments. These values can be read using the :WAVeform:XORigin?, WAVeform:YORigin?, WAVeform:XINCrement?, and WAVeform:YINCreament? queries.

Conversion from Data Values to Units

To convert the waveform data values (essentially A/D counts) to real-world units, such as volts, use the following scaling formulas:

Y-axis Units = data value x Yincrement + Yorigin (analog channels) X-axis Units = data index x Xincrement + Xorigin, where the data index starts at zero: 0, 1, 2, ..., n-1.

The first data point for the time (X-axis units) must be zero, so the time of the first data point is the X origin.

Data Format for Data Transfer

There are four types of data formats that you can select using the :WAVeform:FORMat command: ASCii, BYTE, WORD, and BINary. Refer to the FORMat command in this chapter for more information on data formats.

:WAVeform:BANDpass?

Query `:WAVeform:BANDpass?`

The :WAVeform:BANDpass? query returns an estimate of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limits are computed as a function of the coupling and the selected filter mode. The cutoff frequencies are derived from the acquisition path and software filtering.

Returned Format `[:WAVeform:BANDpass]<lower_cutoff>,<upper_cutoff><NL>`

<lower_cutoff> Minimum frequency passed by the acquisition system.

<upper_cutoff> Maximum frequency passed by the acquisition system.

Example This example places the estimated maximum and minimum bandwidth limits of the source waveform in the string variable, strBandwidth, then prints the contents of the variable to the computer's screen.

```
Dim strBandwidth As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:BANDPASS?"
strBandwidth = myScope.ReadString
Debug.Print strBandwidth
```

:WAVeform:BYTeorder

Command `:WAVeform:BYTeorder {MSBFirst | LSBFirst}`

The :WAVeform:BYTeorder command selects the order in which bytes are transferred to and from the oscilloscope using WORD and LONG formats. If MSBFirst is selected, the most significant byte is transferred first. Otherwise, the least significant byte is transferred first. The default setting is MSBFirst.

NOTE

The data transfer rate is faster using the LSBFirst byte order.

MSBFirst is for microprocessors, where the most significant byte resides at the lower address. LSBFirst is for microprocessors, where the least significant byte resides at the lower address.

Example This example sets up the oscilloscope to send the most significant byte first during data transmission.

```
myScope.WriteString ":WAVEFORM:BYTEORDER MSBFIRST"
```

Query `:WAVeform:BYTeorder?`

The :WAVeform:BYTeorder? query returns the current setting for the byte order.

Returned Format `[:WAVeform:BYTeorder] {MSBFirst | LSBFirst}<NL>`

Example This example places the current setting for the byte order in the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:BYTEORDER?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:WAVeform:COMPlete?

Query `:WAVeform:COMPlete?`

The :WAVeform:COMPlete? query returns the percent of time buckets that are complete for the currently selected waveform.

For the NORMal, RAW, and INTerpolate waveform types, the percent complete is the percent of the number of time buckets that have data in them, compared to the memory depth.

For the AVERage waveform type, the percent complete is the number of time buckets that have had the specified number of hits divided by the memory depth. The hits are specified by the :ACQuire:AVERage:COUNt command.

For the VERSus waveform type, percent complete is the least complete of the X-axis and Y-axis waveforms.

Returned Format `[:WAVeform:COMPlete] <criteria><NL>`

<criteria> 0 to 100 percent, rounded down to the closest integer.

Example This example places the current completion criteria in the string variable, strCriteria, then prints the contents of the variable to the computer's screen.

```
Dim strCriteria As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:COMPLETE?"
strCriteria = myScope.ReadString
Debug.Print strCriteria
```

:WAVeform:COUNt?

Query `:WAVeform:COUNt?`

The :WAVeform:COUNt? query returns the fewest number of hits in all of the time buckets for the currently selected waveform. For the AVERage waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value specified with the :ACQuire:AVERage:COUNt command.

For the NORMal, RAW, INTerpolate, and VERSus waveform types, the count value returned is one, unless the data contains holes (sample points where no data is acquired). If the data contains holes, zero is returned.

Returned Format `[:WAVeform:COUNt] <number><NL>`

<number> An integer. Values range from 0 to 1 for NORMal, RAW, or INTerpolate types, and VERSus type. If averaging is on values range from 0 to 65536.

Example This example places the current count field value in the string variable, strCount, then prints the contents of the variable to the computer's screen.

```
Dim strCount As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:COUNT?"
strCount = myScope.ReadString
Debug.Print strCount
```

:WAVeform:COUPling?

Query `:WAVeform:COUPling?`

The :WAVeform:COUPling? query returns the input coupling of the currently selected source and always returns DC. This query is provided for compatibility to other Infiniium oscilloscopes.

Returned Format `[:WAVeform:COUPling] DC<NL>`

Example This example places the current input coupling of the selected waveform in the string variable, strSetting, then prints the contents of the variable.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:COUPLING?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

See Also The :CHANnel<N>:INPut command sets the coupling for a particular channel.

You can use the :WAVeform:SOURce command to set the source for the coupling query.

Source	Return Value
CGRade	Coupling of the lowest numbered channel that is on.
HISTogram	The coupling of the selected channel. For functions, the coupling of the lowest numbered channel in the function.
CHANnel	The coupling of the channel number
FUNCtion	The coupling of the lowest numbered channel in the function
WMEMory	The coupling value of the source that was loaded into the waveform memory. If channel 1 was loaded, it would be the channel 1 coupling value.

:WAVeform:DATA?

Query `:WAVeform:DATA? [<start>[,<size>]]`

The :WAVeform:DATA? query outputs waveform data to the computer over the remote interface. The data is copied from a waveform memory, function, channel, bus, pod, or digital channel previously specified with the :WAVeform:SOURce command.

NOTE The data's returned response depends upon the setting of the :WAVeform:STReaming command. See "Streaming Off" on page 991 or "Streaming On" on page 992 for more detail.

NOTE If the waveform data is ASCII formatted, no header information indicating the number of bytes being downloaded is included, and the waveform data is separated by commas.

The preamble queries, such as :WAVeform:XINCrement, can be used to determine the vertical scaling, the horizontal scaling, and so on.

<start> An integer value which is the starting point in the source memory which is the first waveform point to transfer.

<size> An integer value which is the number of points in the source memory to transfer. If the size specified is greater than the amount of available data then the size is adjusted to be the maximum available memory depth minus the <start> value.

Streaming Off The returned waveform data response depends upon the setting of the :WAVeform:STReaming command. When the data format is BYTE and streaming is off, the number of waveform points must be less than 1,000,000,000 or an error occurs and only 999,999,999 bytes of data are sent. When the data format is WORD and streaming is off, the number of waveform points must be less than 500,000,000 or an error occurs and only 499,999,999 words of data are sent.

The returned waveform data in response to the :WAVeform:DATA? query is in the following order.

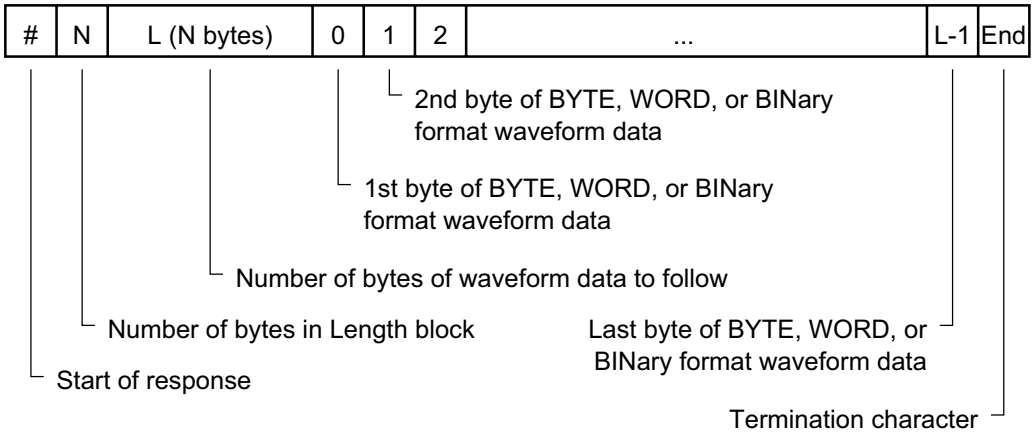


Figure 6 Streaming Off

Streaming On

When streaming is on there is no limit on the number of waveform data points that are returned. It is recommended that any new programs use streaming on to send waveform data points. The waveform data response when streaming is on is as follows.

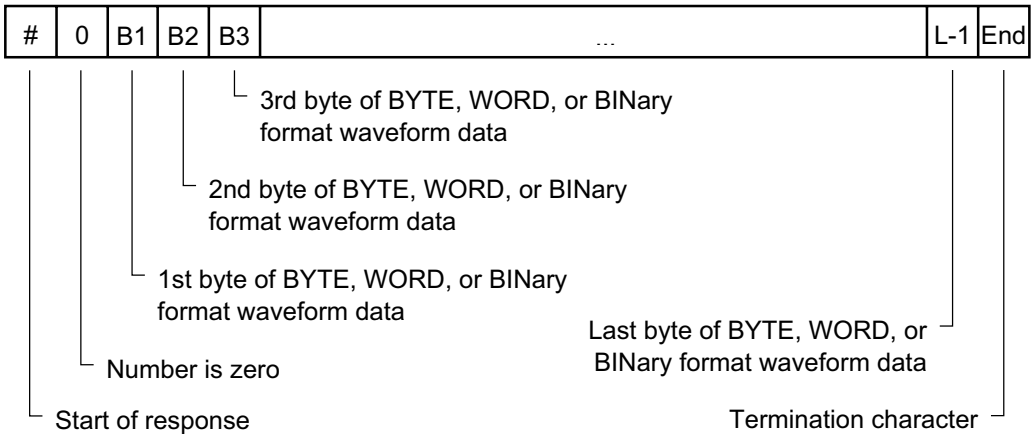


Figure 7 Streaming On

Returned Format

```
[:WAVeform:DATA] <block_data>[,<block_data>]<NL>
```

Example

This example places the current waveform data from channel 1 into the varWavData array in the word format.

```
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:SOURce CHANnel1    ' Select source.
myScope.WriteString ":WAVeform:FORMat WORD"    ' Select word format.
myScope.WriteString ":WAVeform:DATA?"
varWavData = myScope.ReadIEEEBlock(BinaryType_I2)
```

The format of the waveform data must match the format previously specified by the :WAVeform:FORMat and :WAVeform:BYTeorder commands.

DATA? Example for Analog Channels

The following C example shows how to transfer WORD formatted waveform data for analog channels to a computer.

```
/* readdata. c */

/* Reading Word format example.  This program demonstrates the order
 * of commands suggested for operation of the Infiniium oscilloscope
 * via LAN.  This program initializes the oscilloscope, acquires data,
 * transfers data in WORD format, converts the data into time and
 * voltage values, and stores the data in a file as comma-separated
 * ascii values. This format is useful for spreadsheet and MATLAB
 * applications. It requires a waveform which is connected to Channel 1.
*/

#include <stdio.h>      /* location of: printf() */
#include <stdlib.h>     /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"

/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetVoltageConversionFactors( double *yInc, double *yOrg );
void GetTimeConversionFactors( double *xInc, double *xOrg );
void WriteCsvToFile( unsigned long ByteToRead );
void SetupDataTransfer( void );

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Globals */
INST bus;
INST scope;
char buffer[MAX_LENGTH];  /* Buffer for reading data */
double xOrg=0L, xInc=0L;  /* Values used to create time data */
double yOrg=0L, yInc=0L;  /* Values used to convert data to volts */


void main( void )
{
   unsigned long BytesToRead;

   if ( !InitIO() )
```

```
   {
      exit( 1 );
   }

   AcquireData();

   WriteIO( ":WAVeform:FORMat WORD" );    /* Setup transfer format */
   WriteIO( ":WAVeform:BYTeorder LSBFirst" ); /* Setup transfer of
                                                 LSB first */
   WriteIO( ":WAVeform:SOURce CHANnel1" ); /* Waveform data source
                                              channel 1 */
   WriteIO( ":WAVeform:STReaming 1" ); /* Turn on waveform
                                          streaming of data */

   GetVoltageConversionFactors( &yInc, &yOrg );
   GetTimeConversionFactors( &xInc, &xOrg );
   BytesToRead = MAX_LENGTH;
   SetupDataTransfer();
   WriteCsvToFile( BytesToRead );

   CloseIO( );

}

/*****************************************************************
 * Function name:  InitIO
 *   Parameters:   none
 *   Return value: TRUE if successful otherwise FALSE
 *   Description:  This routine initializes the SICL environment.
 *                 It sets up error handling, opens both an interface
 *                 and device session, sets timeout values, clears
 *                 the LAN interface card, and clears the
 *                 oscilloscope's LAN interface by performing a
 *                 Selected Device Clear.
*****************************************************************/
int InitIO( void )
{

   ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

   bus = iopen( INTERFACE );   /* open interface session */
   if ( bus == 0 )
   {
      printf( "Bus session invalid\n" );
      return FALSE;
   }

   itimeout( bus, IO_TIMEOUT );  /* set bus timeout */
   iclear( bus );                /* clear the interface */

   scope = bus;                  /* open the scope device session */

   return TRUE;
}

/*****************************************************************
 * Function name:  WriteIO
```

```
 *   Parameters:   char *buffer which is a pointer to the character
 *                    string to be output
 *   Return value: none
 *   Description:  This routine outputs strings to the oscilloscope
 *                 device session using SICL commands.
*********************************************************************/

void WriteIO( char *buffer )
{
   unsigned long actualcnt;
   unsigned long BytesToRead;
   int send_end = 1;

   BytesToRead = strlen( buffer );

   iwrite( scope, buffer, BytesToRead, send_end, &actualcnt );

}

/*********************************************************************
 * Function name:  ReadByte
 *   Parameters:   char *buffer which is a pointer to the array to
 *                   store the read bytes
 *                 unsigned long BytesToRead which indicates the
 *                   maximum number of bytes to read
 *   Return value: integer which indicates the actual number of bytes
 *                 read
 *   Description: This routine inputs strings from the scope device
 *                session using SICL commands.
*********************************************************************/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
   unsigned long BytesRead;
   int reason;

   BytesRead = BytesToRead;

   iread( scope, buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}

/*********************************************************************
 * Function name:  ReadWord
 *   Parameters:   short *buffer which is a pointer to the word array
 *                    to store the bytes read
 *                 int reason which is the reason that the read
 *                    terminated
 *                 unsigned long BytesToRead which indicates the
 *                    maximum number of bytes to read
 *   Return value: integer which indicates the actual number of
 *                    bytes read
 *   Description: This routine inputs an array of short values from
 *                the oscilloscope device session using SICL commands.
*********************************************************************/
```

```
unsigned long ReadWord( char *buffer, int *reason,
                        unsigned long BytesToRead )
{
   long BytesRead;

   iread( scope, buffer, BytesToRead, reason, &BytesRead );

   return BytesRead;
}

/*********************************************************************
 * Function name:  ReadDouble
 *   Parameters:   double *buffer which is a pointer to the float
 *                    value to read
 *   Return value: none
 *   Description: This routine inputs a float value from the
 *                oscilloscope device session using SICL commands.
*********************************************************************/

void ReadDouble( double *buffer )
{
   iscanf( scope, "%lf", buffer );
}

/*********************************************************************
 * Function name: close_IO
 *   Parameters:   none
 *   Return value: none
 *   Description:  This routine closes device and interface sessions
 *                 for the SICL environment, and calls the routine
 *                 _siclcleanup which de-allocates resources
 *                 used by the SICL environment.
*********************************************************************/

void CloseIO( void )
{

   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

}

/*********************************************************************
 * Function name:  AcquireData
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine acquires data using the current
 *                oscilloscope settings.
*********************************************************************/

void AcquireData( void )
{
   /*
    * The root level :DIGitize command is recommended for
       * acquiring new waveform data. It initializes the
```

```
       * oscilloscope's data buffers, acquires new data,
       * and ensures that acquisition criteria are met before the
       * acquisition is stopped.  Note that the display is
       * automatically turned off when you use this form of the
       * :DIGitize command and must be turned on to view the
       * captured data on screen.
   */

   WriteIO(":DIGitize CHANnel1");
   WriteIO(":CHANnel1:DISPlay ON");

}

/*********************************************************************
 * Function name: GetVoltageConversionFactors
 *   Parameters:  double yInc which is the voltage difference
 *                   represented by adjacent waveform data digital codes
.
 *                double yOrg which is the voltage value of digital
 *                  code 0.
 *   Return value: none
 *   Description: This routine reads the conversion factors used to
 *                convert waveform data to volts.
*********************************************************************/

void GetVoltageConversionFactors( double *yInc, double *yOrg )
{

   /* Read values which are used to convert data to voltage values */

   WriteIO(":WAVeform:YINCrement?");
   ReadDouble( yInc );

   WriteIO(":WAVeform:YORigin?");
   ReadDouble( yOrg );

}

/*********************************************************************
 * Function name:  SetupDataTransfer
 *   Parameters:   none
 *   Return value: none
 *   Description: This routine sets up the waveform data transfer and
 *                removes the # and 0 characters.
*********************************************************************/

void SetupDataTransfer( void )
{
   char cData;

   WriteIO( ":WAVeform:DATA?" );   /* Request waveform data */

   /* Find the # character */

   do
   {
      ReadByte( &cData, 1L );
```

```
   } while ( cData != '#' );

   /* Find the 0 character */

   do
   {
      ReadByte( &cData, 1L );
   } while ( cData != '0' );
}

/*********************************************************************
 * Function name:  GetTimeConversionFactors
 *   Parameters:   double xInc which is the time between consecutive
 *                    sample points.
 *                 double xOrg which is the time value of the first
 *                    data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
*********************************************************************/

void GetTimeConversionFactors( double *xInc, double *xOrg )
{

   /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( xOrg );

}

/*********************************************************************
 * Function name:  WriteCsvToFile
 *   Parameters:   unsigned long BytesToRead which is the number of
 *                   data points to read
 *   Return value: none
 *   Description: This routine stores the time and voltage
 *                information about the waveform as time and
 *                voltage separated by commas to a file.
*********************************************************************/

void WriteCsvToFile( unsigned long BytesToRead )
{
   FILE *fp;
   int done = FALSE;
   int reason = 0;
   unsigned long i;
   unsigned long j = 0;
   unsigned long BytesRead = 0L;
   double Time;
   double Volts;
   short *buff;

   fp = fopen( "pairs.csv", "wb" ); /* Open file in binary mode - clear
```

```
                                                file if it already exists */

   if (fp != NULL)
   {
      while( !done )
      {
         BytesRead = ReadWord( buffer, &reason, BytesToRead );

         switch( reason )
         {
         case I_TERM_MAXCNT:
            done = FALSE;
            break;
         case I_TERM_END:
            done = TRUE;
            break;
         case I_TERM_CHR:
            done = TRUE;
            break;
         default:
            done = TRUE;
            break;
         };

         buff = (short *) buffer;

         for( i = 0; i < ((BytesRead - 1)/2); i++)
         {
            Time = (j * xInc) + xOrg; /* calculate time */
            j = j + 1;

            Volts = (buff[i] * yInc) + yOrg;/* calculate voltage */

            fprintf( fp, "%e,%f\n", Time, Volts );
         }
      }
      fclose( fp );

   }
   else
   {
      printf("Unable to open file 'pairs.csv'\n");
   }
}
```

DATA? Example for Digital Channels

The following C example shows how to transfer both BYTE and WORD formatted waveform data for digital channels to a computer. There is a file on the Infiniium Oscilloscope Example Programs disk called readdig.c in the c directory that contains this program.

```
/* readdig. c */

/* Reading Byte and Word format Example.  This program demonstrates the
order of
 commands suggested for operation of the Infiniium oscilloscope by LAN o
r GPIB.
```

```
 This program initializes the scope, acquires data, transfers data in bo
th the
 BYTE and WORD formats, converts the data into hex, octal, binary and ti
me values,
 and stores the data in a file as comma-separated values.  This format i
s useful
 for spreadsheet applications.
*/

#include <stdio.h>       /* location of: printf() */
#include <stdlib.h>      /* location of: atof(), atoi() */
#include <string.h>      /* location of: strlen() */
#include "sicl.h"

/* Prototypes */
int InitIO( void );
void WriteIO( char *buffer );
unsigned long ReadByte( char *buffer, unsigned long BytesToRead);
unsigned long ReadWord( short *buffer, unsigned long BytesToRead);
void ReadDouble( double *buffer );
void CloseIO( void );
void AcquireData( void );
void GetTimeConversionFactors( void );
void CreateTimeData( unsigned long AcquiredLength,
                     double *TimeValues );
void WriteCsvToFile( double *TimeValues, unsigned short *wordData,
                     unsigned char *byteData, unsigned long AcquiredLeng
th);
unsigned long SetupDataTransfer( double lTime, double rTime );
int Round( double number );

/* Defines */
#define MAX_LENGTH 8192000

#define LAN

#ifdef LAN
   #define INTERFACE "lan[130.29.71.202]:hpib7,7"
#else
   #define INTERFACE "hpib7"
#endif

#define DEVICE_ADDR "hpib7,7"
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Globals */
INST bus;
INST scope;
double TimeValues[MAX_LENGTH];         /* Time value of data */
unsigned short wordData[MAX_LENGTH/2];/* Buffer for reading word format
data */
unsigned char byteData[MAX_LENGTH];    /* Buffer for reading byte format
data */
double xOrg, xInc;                    /* Values necessary to create time
data */
```

```
int Start;

void main( void )
{
   char Term;
   unsigned long BytesToRead;

   if ( !InitIO() ) {
      exit( 1 );
   }

   AcquireData();

   WriteIO( ":SYStem:HEADer OFF" );
   WriteIO( ":SYStem:LONGform OFF" );
   WriteIO( ":WAVeform:BYTEorder LSBFirst" ); /* Setup byte order */
   WriteIO( ":WAVeform:FORMat WORD" );        /* Setup transfer format *
/
   WriteIO( ":WAVeform:SOURce POD1" );      /* Waveform data source pod
1 */

   GetTimeConversionFactors( );

   BytesToRead = SetupDataTransfer( -25E-6, 25E-6);
   ReadWord( wordData, BytesToRead );
   ReadByte( &Term, 1L );                /* Read termination character *
/

   WriteIO(":WAVeform:FORMat BYTE");     /* Setup transfer format */

   BytesToRead = SetupDataTransfer( -25E-6, 25E-6);
   ReadByte( byteData, BytesToRead );
   ReadByte( &Term, 1L );                /* Read termination character *
/

   CreateTimeData( BytesToRead, TimeValues );

   WriteCsvToFile( TimeValues, wordData, byteData, BytesToRead );

   CloseIO( );

}

/*********************************************************************
********
*   Function name: InitIO
*   Parameters:    none
*   Return value:  none
*   Description:   This routine initializes the SICL environment.  It se
ts up
*                  errorhandling, opens both an interface and device ses
sion,
*                  sets timeout values, clears the GPIB interface card,
and
*                  clears the oscilloscope's GPIB card by performing a
*                  Selected Device Clear.
```

```
*************************************************************************
*******/

int InitIO( void )
{

   ionerror( I_ERROR_EXIT );   /* set-up interface error handling */

   bus = iopen( INTERFACE );   /* open interface session */
   if ( bus == 0 ) {
      printf( "Bus session invalid\n" );
      return FALSE;
   }

   itimeout( bus, IO_TIMEOUT );       /* set bus timeout */
   iclear( bus );                     /* clear the interface */

#ifdef LAN
   scope = bus;
#else
   scope = iopen( DEVICE_ADDR );      /* open the scope device session *
/
   if ( scope == 0) {
      printf( "Scope session invalid\n" );
      iclose( bus );     /* close interface session */
      _siclcleanup();    /* required for 16-bit applications */
      return FALSE;
   }

   itimeout( scope, IO_TIMEOUT );     /* set device timeout */
   iclear( scope );      /* perform Selected Device Clear on oscilloscop
e */
#endif

   return TRUE;
}

/*************************************************************************
********
*   Function name: WriteIO
*   Parameters:    char *buffer which is a pointer to the character stri
ng to
*                  be output
*   Return value:  none
*   Description: This routine outputs strings to the oscilloscope device
*                session using SICL commands.
*************************************************************************
*******/

void WriteIO( char *buffer )
{
   unsigned long actualcnt;
   unsigned long BytesToWrite;
   int send_end = 1;

   BytesToWrite = strlen( buffer );
```

```
   iwrite( scope, buffer, BytesToWrite, send_end, &actualcnt );

}


/*********************************************************************
*******
*   Function name: ReadByte
*   Parameters:    char *buffer which is a pointer to the array to store
*                  the read bytes unsigned long BytesToRead which indica
tes
*                  the maximum number of bytes to read
*   Return value:  integer which indicates the actual number of bytes re
ad
*   Description: This routine inputs strings from the scope device sessi
on
*                using SICL commands.
*********************************************************************
******/

unsigned long ReadByte( char *buffer, unsigned long BytesToRead )
{
   unsigned long BytesRead=0L;
   int reason;

   BytesRead = BytesToRead;
    iread( scope, buffer, BytesToRead, &reason, &BytesRead );

    return BytesRead;
}

/*********************************************************************
*******
*   Function name: ReadWord
*   Parameters:    short *buffer which is a pointer to the word array to
 store
*                  the bytes read unsigned long BytesToRead which indica
tes
*                  the maximum number of bytes to read
*   Return value:  integer which indicates the actual number of bytes re
ad
*   Description: This routine inputs an array of short values from the
*                 oscilloscope device session using SICL commands.
*********************************************************************
******/

unsigned long ReadWord( short *buffer, unsigned long BytesToRead )
{
   long BytesRead=0L;
   int reason;

   BytesRead = BytesToRead;
   iread( scope, (char *) buffer, BytesToRead, &reason, &BytesRead );

   return BytesRead;
}
```

```
/*********************************************************************
*******
*   Function name: ReadDouble
*   Parameters:    double *buffer which is a pointer to the float value
to read
*   Return value:  none
*   Description:   This routine inputs a float value from the oscillosco
pe
*                   device session using SICL commands.
*********************************************************************
*******/

void ReadDouble( double *buffer )
{
   int error;
    error = iscanf( scope, "%lf", buffer );
}

/*********************************************************************
*******
*  Function name: close_IO
*   Parameters:   none
*   Return value: none
*   Description:  This routine closes device and interface sessions for
*                 the SICL environment, and calls the routine _siclclean
up
*                 which de-allocates resources used by the SICL environm
ent.
*********************************************************************
******/

void CloseIO( void )
{

   iclose( scope );   /* close device session */
   iclose( bus );     /* close interface session */

   _siclcleanup();    /* required for 16-bit applications */

}

/*********************************************************************
*******
*   Function name:   AcquireData
*   Parameters:     none
*   Return value:   none
*   Description:   This routine acquires data using the current
*                  oscilloscope settings.
*********************************************************************
******/

void AcquireData( void )
{
   /*
    *   The root level :DIGitize command is recommended for acquiring ne
w
    *   waveform data. It initializes the oscilloscope's data buffers,
```

```
    *   acquires new data, and ensures that acquisition criteria are met
    *   before the acquisition is stopped.  Note that the display is
    *   automatically turned off when you use this form of the
    *   :DIGitize command and must be turned on to view the captured dat
a
    *   on screen.
   */

   WriteIO(":DIGitize POD1");
   WriteIO(":POD1:DISPlay ON");

}

/**************************************************************************
*******
*   Function name: SetupDataTransfer
*   Parameters:    double lTime which is the time value of the first
*                  waveform memory location of data.
*                  double rTime which is the time value of the last
*                  waveform memory location of data.
*   Return value:  Number of bytes of waveform data to read.
*   Description: This routine sets up the waveform data transfer and get
s
*                the number of bytes to be read.  The beginning of data
*                starts with the # character followed by a number which
*                tells how many bytes to read for the integer which is t
he
*                total number of data bytes that are being transfered.
*                Following this is the waveform data.  For example, if 1
024
*                bytes of waveform data is being transfered then this
*                information will be as follows:
*                             #41024 <1024 data bytes>
**************************************************************************
******/

unsigned long SetupDataTransfer( double lTime, double rTime )
{
   unsigned long BytesToRead;
   char header_str[8];
   char cData;
   unsigned long BytesRead;
   int Size;
   char Range[100];

   /* Find the index value of the first data memory location */

   Start = Round((lTime - xOrg)/xInc);
   if (Start < 1) {
      Start = 1;
   }

   /* Find the number of data bytes that you want */

   Size = Round((rTime - lTime)/xInc);

   sprintf( Range, ":WAVeform:DATA? %d,%d", Start, Size);
```

```
   WriteIO( Range );                /* Request waveform data */

   /* Find the # character */

   do {
         ReadByte( &cData, 1L );
   } while ( cData != '#' );

   /* Read the next byte which tells how many bytes to read for the numb
er
   * of waveform data bytes to transfer value.
   */

   ReadByte( &cData, 1L );
   BytesToRead = cData - '0';  /* Convert to a number */

   /* Reads the number of data bytes that will be transfered */

   BytesRead = ReadByte( header_str, BytesToRead );
   header_str[BytesRead] = '\0';
   BytesToRead = atoi( header_str );

   return BytesToRead;

}

/*************************************************************************
*******
*   Function name: GetTimeConversionFactors
*   Parameters:    none
*   Return value:  none
*   Description:   This routine transfers the waveform conversion
*                  factors for the time values.
*************************************************************************
*****/

Void GetTimeConversionFactors( void )
{

   /* Read values which are used to create time values */

   WriteIO(":WAVeform:XINCrement?");
   ReadDouble( &xInc );

   WriteIO(":WAVeform:XORigin?");
   ReadDouble( &xOrg );

}

/*************************************************************************
*******
*   Function name: CreateTimeData
*   Parameters:    unsigned long AcquiredLength which is the number of d
ata
*                  points
*                  double TimeValues is a pointer to the array where tim
e
```

```
*                   values are stored
*   Return value:  none
*   Description: This routine converts the data to time values using
*                the values that describe the waveform.  These values ar
e stored
*                in global variables.
************************************************************************
*****/

void CreateTimeData( unsigned long AcquiredLength, double *TimeValues )
{
   unsigned long i;

   for (i = 0; i < AcquiredLength; i++) {
      TimeValues[i] =((Start + i) * xInc) + xOrg;   /* calculate time va
lues */
   }

}

/************************************************************************
*******
*   Function name: WriteCsvToFile
*   Parameters:    double *TimeValues which is a pointer to an array of
*                  calculated time values
*                  unsigned short *wordData which is a pointer to an arr
ay of
*                  word format digital values
*                  unsigned char *byteData which is a pointer to an arra
y of
*                  byte format digital values
*                  unsigned long AcquiredLength which is the number of d
ata
*                  points read
*   Return value:  none
*   Description:   This routine stores the time and digital information
about
*                  the waveform as time, word format, and byte format
*                  separated by commas to a file.
************************************************************************
******/

void WriteCsvToFile( double *TimeValues, unsigned short *wordData,
                unsigned char *byteData, unsigned long AcquiredLength )
{
  FILE *fp;
  char Binary[9];
  unsigned long i;
  int j;
  int k;

  fp = fopen( "digital.csv", "wb" ); /* Open file in binary mode - clear
 file
                                        if it already exists */

  if (fp != NULL) {
```

```
     fprintf( fp, "Time,Decimal Word Data,Hex Word Data,Hex Byte Data,Bi
nary Byte Data\n");
     Binary[8] = '\0';

     for ( i = 0; i < AcquiredLength; i++ ) {

        // Create the binary formated byte data
        for ( j = 7, k = 0; j >= 0; j--, k++ ) {
               Binary[k] = ((byteData[i] & ( 1 << j )) >> j) + '0';
          }

        fprintf( fp, "%e,%d,%04X,%02X,%s\
n", TimeValues[i], wordData[i], wordData[i],
                                              byteData[i], Binary );
     }

       fclose( fp );
  }
  else {
       printf("Unable to open file 'digital.csv'\n");
  }

}

/*************************************************************************
*******
*   Function name: Round
*   Parameters:    double number which is a floating point number
*                  to be converted.
*   Return value:  The rounded integer value for the number parameter.
*   Description:   This routine takes a floating point number and create
s an
*                  integer.
*************************************************************************
*****/

int Round( double number )
{
   if (number < 0.0f ) {
      return ( (int) (number - 0.5f) );
   }
   else {
      return ( (int) (number + 0.5f) );
   }
}
```

Understanding WORD and BYTE Formats

Before you can understand how the WORD and BYTE downloads work, it is necessary to understand how Infiniium creates waveform data.

Analog-to-digital Conversion Basics

The input channel of every digital sampling oscilloscope contains an analog-to-digital converter (ADC) as shown in Figure 8. The 8-bit ADC in Infiniium consists of 256 voltage comparators. Each comparator has two inputs. One input is connected to a reference dc voltage level and the other input is connected to the channel input. When the voltage of the waveform on the channel input is greater than the dc level, then the

comparator output is a 1 otherwise the output is a 0. Each of the comparators has a different reference dc voltage. The output of the comparators is converted into an 8-bit integer by the encoder.

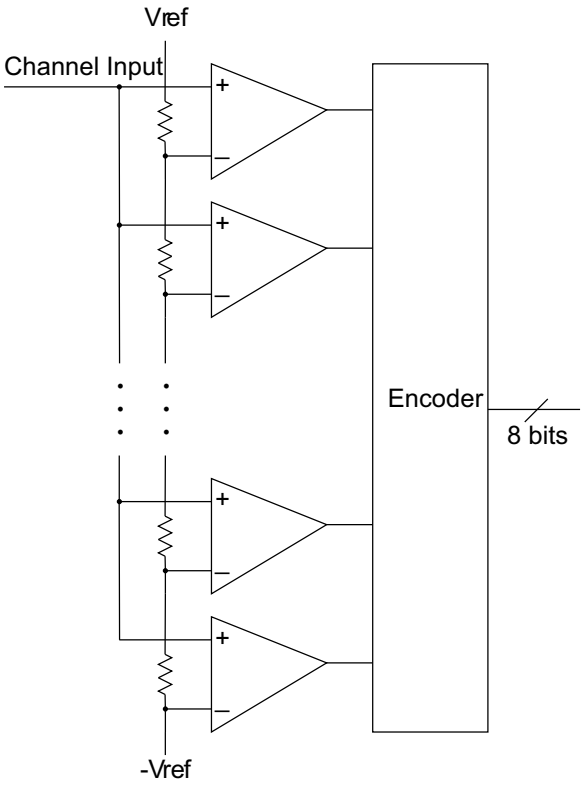


Figure 8 Block Diagram of an ADC

All ADCs have non-linearity errors which, if not corrected, can give less accurate vertical measurement results. For example, the non-linearity error for a 3-bit ADC is shown in the following figure.

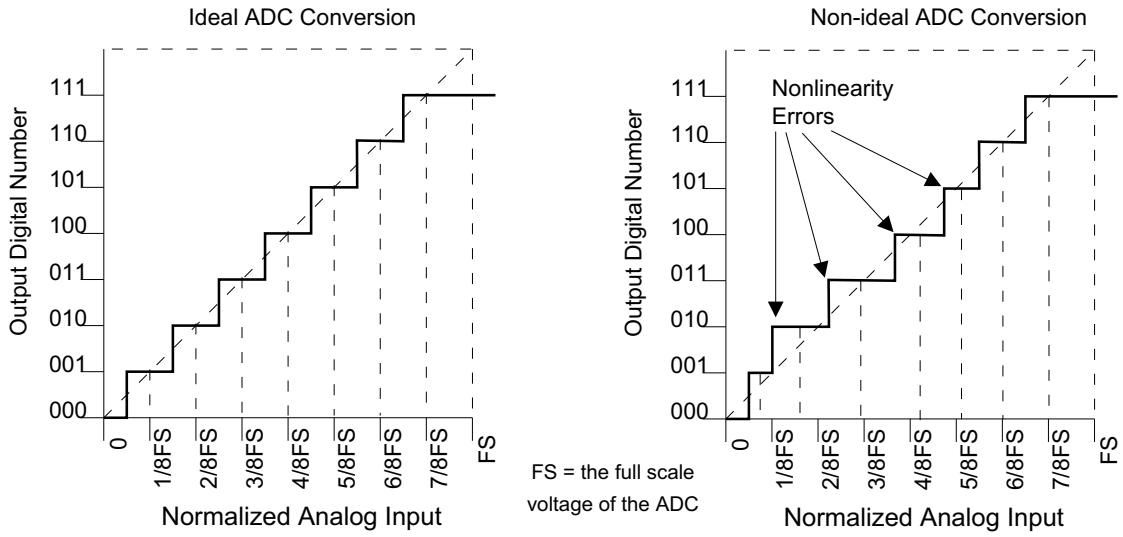


Figure 9 ADC Non-linearity Errors for a 3-bit ADC

The graph on the left shows an ADC which has no non-linearity errors. All of the voltage levels are evenly spaced producing output codes that represent evenly spaced voltages. In the graph on the right, the voltages are not evenly spaced with some being wider and some being narrower than the others.

When you calibrate your Infiniium, the input to each channel, in turn, is connected to the Aux Out connector. The Aux Out is connected to a 16-bit digital-to-analog converter (DAC) whose input is controlled by Infiniium's CPU. There are 65,536 dc voltage levels that are produced by the 16-bit DAC at the Aux Out. At each dc voltage value, the output of the ADC is checked to see if a new digital code is produced. When this happens, a 16-bit correction factor is calculated for that digital code and this correction factor is stored in a Calibration Look-up Table.

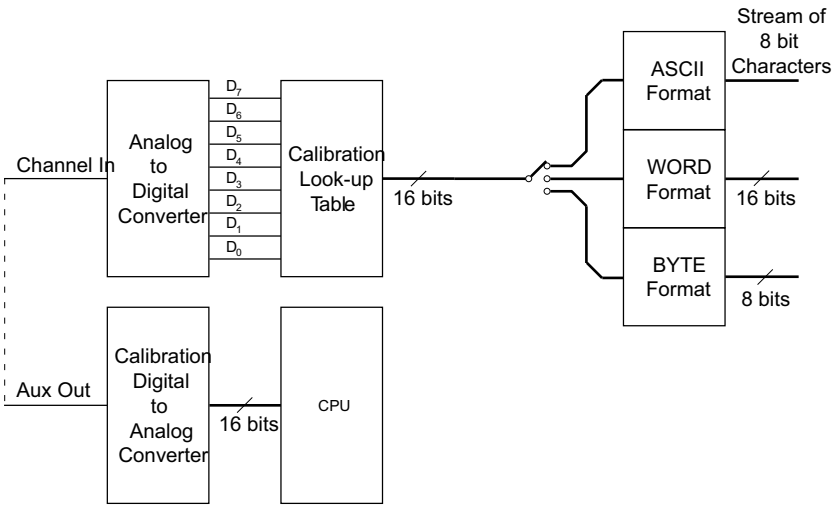


Figure 10 Data Flow in Infiniium

This process continues until all 256 digital codes are calibrated. The calibration process removes most of the non-linearity error of the ADC which yields more accurate vertical voltage values.

During normal operation of the oscilloscope, the output of the ADC is used as an address to the Calibration Look-up Table which produces 16-bit data for the oscilloscope to process and display. The output of the ADC is a signed 8-bit integer and the output of the Calibration Look-up Table is a signed 16-bit integer. If the amplitude of the input waveform is larger than the maximum dc reference level of the ADC, the ADC will output the maximum 8-bit value that it can (255). This condition is called ADC clipping. When the 255 digital code is applied to the Calibration Look-up Table, a 16-bit value, such as 26,188 could be produced which represents an ADC clipped value. This number will vary from one oscilloscope to the next.

WORD and BYTE Data Formats

When downloading the waveform data in WORD format, the 16-bit signed integer value for each data point is sent in two consecutive 8-bit bytes over the remote interface. Whether the least significant byte (LSB) or the most significant byte (MSB) is sent first depends on the byte order determined by the BYTeorder command.

Before downloading the waveform data in BYTE format, each 16-bit signed integer is converted into an 8-bit signed integer. Because there are more possible 16-bit integers than there are 8-bit integers, a range of 16-bit integers is converted into single 8-bit numbers. For example, the following 16-bit numbers are all converted into one 8-bit number.

16-Bit Integers			8-Bit Integer	
Decimal	**Hex**		**Hex**	**Decimal**
26,240	0x6680	Truncated to >>	0x66	102
26,200	0x6658			
26,160	0x6630			
26,120	0x6608			

This conversion is what makes the BYTE download format less accurate than the WORD format.

:WAVeform:FORMat

Command `:WAVeform:FORMat {ASCii | BINary | BYTE | WORD}`

The :WAVeform:FORMat command sets the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope, and pertains to all waveforms.

The default format is ASCii.

Table 27 Selecting a Format

Type	Advantages	Disadvantages
ASCii	• Data is returned as voltage values and does not need to be converted. • Is as accurate as WORD format. • Supports HISTogram SOURce.	• Very slow data download rate.
BYTE	• Data download rate is twice as fast as the WORD format.	• Data is less accurate than the WORD format for analog channels. • Not compatible with digital bus and pod data.
WORD	• Data is the most accurate for analog channels.	• Data download rate takes twice as long as the BYTE format.
BINary	• Supports HISTogram SOURce. • Can be used for analog channels.	• Data download rate takes twice as long as the BYTE format for analog channels.

ASCii ASCii-formatted data consists of waveform data values converted to the currently selected units, such as volts, and are output as a string of ASCII characters with each value separated from the next value by a comma. The values are formatted in floating point engineering notation. For example:

```
8.0836E+2,8.1090E+2,...,-3.1245E-3
```

NOTE The ASCii format does not send out the header information indicating the number of bytes being downloaded.

In ASCii format:

- The value "99.999E+36" represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

BYTE BYTE-formatted data is formatted as *signed* 8-bit integers. Depending on your programming language and IO library, you may need to create a function to convert these signed bytes to signed integers. In BYTE format:

- The value 125 represents a hole value. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

The waveform data values are converted from 16-bit integers to 8-bit integers before being downloaded to the computer. For more information, see "Understanding WORD and BYTE Formats" on page 1008.

WORD WORD-formatted data is transferred as *signed* 16-bit integers in two bytes. If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each word is sent first. In WORD format:

- The value 31232 represents a hole level. A hole can occur when you are using the equivalent time sampling mode when during a single acquisition not all of the acquisition memory locations contain sampled waveform data. It can take several acquisitions in the equivalent time sampling mode to fill all of the memory locations.

For more information, see "Understanding WORD and BYTE Formats" on page 1008.

BINary BINary-formatted data can be used with any SOURce. When a source is any valid source except for histogram, the data is returned in WORD format.

When the source is set to HISTogram, the data is transferred as signed 64-bit integers in 8 bytes. The are no hole values in the histogram data.

If :WAVeform:BYTeorder is set to MSBFirst, the most significant byte of each long word is sent first. If the BYTeorder is LSBFirst, the least significant byte of each long word is sent first.

Example This example selects the WORD format for waveform data transmission.

```
myScope.WriteString ":WAVeform:FORMat WORD"
```

Query `:WAVeform:FORMat?`

The :WAVeform:FORMat? query returns the current output format for transferring waveform data.

Returned Format `[:WAVeform:FORMat] {ASCii | BINary | BYTE | WORD}<NL>`

Example This example places the current output format for data transmission in the string variable, strMode, then prints the contents of the variable to the computer's screen.

```
Dim strMode As String   ' Dimension variable.
myScope.WriteString ":WAVeform:FORMat?"
strMode = myScope.ReadString
Debug.Print strMode
```

:WAVeform:POINts?

Query `:WAVeform:POINts?`

The :WAVeform:POINts? query returns the points value in the current waveform preamble. The points value is the number of time buckets contained in the waveform selected with the :WAVeform:SOURce command. If the Sin(x)/x interpolation filter is enabled, the number of points can be larger than the oscilloscope's memory depth setting because the waveform includes the interpolated points.

Returned Format `[:WAVeform:POINts] <points><NL>`

<points> An integer. See the :ACQuire:POINts command for a table of possible values.

Example This example places the current acquisition length in the numeric variable, varLength, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:POINTS?"
varLength = myScope.ReadNumber
Debug.Print FormatNumber(varLength, 0)
```

NOTE

Turn Headers Off

When you are receiving numeric data into numeric variables, you should turn the headers off. Otherwise, the headers may cause misinterpretation of returned data.

See Also The :ACQuire:POINts command in the ACQuire Commands chapter.

:WAVeform:PREamble?

Query `:WAVeform:PREamble?`

The :WAVeform:PREamble? query outputs a waveform preamble to the computer from the waveform source, which can be a waveform memory or channel buffer.

Returned Format `[:WAVeform:PREamble] <preamble_data><NL>`

The preamble can be used to translate raw data into time and voltage values. The following lists the elements in the preamble.

<preamble_data> <format>, <type>, <points>, <count> , <X increment>, <X origin>, < X reference>, <Y increment>, <Y origin>, <Y reference>, <coupling>, <X display range>, <X display origin>, <Y display range>, <Y display origin>, <date>, <time>, <frame model #>, <acquisition mode>, <completion>, <X units>, <Y units>, <max bandwidth limit>, <min bandwidth limit>

<format> 0 for ASCii format. 1 for BYTE format. 2 for WORD format. 3 for LONG format. 4 for LONGLONG

<type> Returned type values can be:

- 1 — RAW
- 2 — AVERage
- 3 — VHIStogram
- 4 — HHIStogram
- 5 — not used
- 6 — INTerpolate
- 7 — not used
- 8 — not used
- 9 — DIGITAL
- 10 — PDETect

<points> The number of data points or data pairs contained in the waveform data. (See ":ACQuire:POINts[:ANALog]" on page 161.)

<count> For the AVERAGE waveform type, the count value is the fewest number of hits for all time buckets. This value may be less than or equal to the value requested with the :ACQuire:AVERage:COUNt command. For RAW and INTerpolate waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble. (See ":WAVeform:TYPE?" on page 1028 and ":ACQuire[:AVERage]:COUNt" on page 151.)

<X increment> The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired. (See the ":WAVeform:XINCrement?" on page 1032 query.)

<X origin> The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired. (See the ":WAVeform:XORigin?" on page 1033 query.)

<X reference> The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the ":WAVeform:XREFerence?" on page 1035 query.)

<Y increment> The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero then no data has been acquired. (See the ":WAVeform:YINCrement?" on page 1038 query.)

<Y origin> The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero then no data has been acquired. (See the ":WAVeform:YORigin?" on page 1039 query.)

<Y reference> The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero. (See the ":WAVeform:YREFerence?" on page 1041 query.)

<coupling> 0 for AC coupling. 1 for DC coupling. 2 for DCFIFTY coupling. 3 for LFREJECT coupling.

<X display range> The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired. (See the ":WAVeform:XRANge?" on page 1034 query.)

<X display origin> The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired. (See the ":WAVeform:XDISplay?" on page 1031 query.)

<Y display range> The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero then no data has been acquired. (See the ":WAVeform:YRANge?" on page 1040 query.)

<Y display origin> The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero then no data has been acquired. (See the ":WAVeform:YDISplay?" on page 1037 query.)

<date> A string containing the date in the format DD MMM YYYY, where DD is the day, 1 to 31; MMM is the month; and YYYY is the year.

<time> A string containing the time in the format HH:MM:SS:TT, where HH is the hour, 0 to 23, MM is the minutes, 0 to 59, SS is the seconds, 0 to 59, and TT is the hundreds of seconds, 0 to 99.

<frame_model_#> A string containing the model number and serial number of the oscilloscope in the format of MODEL#:SERIAL#.

<acquisition_mode> 0 for RTIMe mode. 1 for ETIMe mode. 2 not used. 3 for PDETect.

<completion> The completion value is the percent of time buckets that are complete. The completion value is ignored when it is sent to the oscilloscope in the preamble. (See the ":WAVeform:COMPlete?" on page 988 query.)

<x_units> **<y_units>** 0 for UNKNOWN units. 1 for VOLT units. 2 for SECOND units. 3 for CONSTANT units. 4 for AMP units. 5 for DECIBEL units.

<max bandwidth limit> <min bandwidth limit> The band pass consists of two values that are an estimation of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the ":WAVeform:BANDpass?" on page 986 query.)

See Table 28 for descriptions of all the waveform preamble elements.

Example This example outputs the current waveform preamble for the selected source to the string variable, strPreamble.

```
Dim strPreamble As String   ' Dimension variable.
myScope.WriteString ":SYSTem:HEADer OFF"    ' Response headers off.
myScope.WriteString ":WAVeform:PREamble?"
strPreamble = myScope.ReadString
```

Table 28 Waveform Preamble Elements

Element	Description
Format	The format value describes the data transmission mode for waveform data output. This command controls how the data is formatted when it is sent from the oscilloscope. (See ":WAVeform:FORMat" on page 1013.)
Type	This value describes how the waveform was acquired. (See also the ":WAVeform:TYPE?" on page 1028 query.)
Points	The number of data points or data pairs contained in the waveform data. (See ":ACQuire:POINts[:ANALog]" on page 161.)
Count	For the AVERAGE waveform type, the count is the number of averages that have occurred. For RAW and INTERPOLATE waveform types, this value is 0 or 1. The count value is ignored when it is sent to the oscilloscope in the preamble. (See ":WAVeform:TYPE?" on page 1028 and ":ACQuire[:AVERage]:COUNt" on page 151.)

Table 28 Waveform Preamble Elements (continued)

Element	Description
X Increment	The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. (See the ":WAVeform:XINCrement?" on page 1032 query.)
X Origin	The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. (See the ":WAVeform:XORigin?" on page 1033 query.)
X Reference	The X reference is the data point associated with the X origin. It is at this data point that the X origin is defined. In this oscilloscope, the value is always zero. (See the ":WAVeform:XREFerence?" on page 1035 query.)
Y Increment	The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. (See the ":WAVeform:YINCrement?" on page 1038 query.)
Y Origin	The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. (See the ":WAVeform:YORigin?" on page 1039 query.)
Y Reference	The Y reference is the level associated with the Y origin. It is at this level that the Y origin is defined. In this oscilloscope, this value is always zero. (See the ":WAVeform:YREFerence?" on page 1041 query.)
Coupling	The input coupling of the waveform. The coupling value is ignored when sent to the oscilloscope in the preamble. (See the ":WAVeform:COUPling?" on page 990 query.)
X Display Range	The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. (See the ":WAVeform:XRANge?" on page 1034 query.)
X Display Origin	The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. (See the ":WAVeform:XDISplay?" on page 1031 query.)
Y Display Range	The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. (See the ":WAVeform:YRANge?" on page 1040 query.)
Y Display Origin	The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. (See the ":WAVeform:YDISplay?" on page 1037 query.)
Date	The date that the waveform was acquired or created.
Time	The time that the waveform was acquired or created.
Frame Model #	The model number of the frame that acquired or created this waveform. The frame model number is ignored when it is sent to an oscilloscope in the preamble.

Table 28 Waveform Preamble Elements (continued)

Element	Description
Acquisition Mode	The acquisition sampling mode of the waveform. (See ":ACQuire:MODE" on page 159.)
Complete	The complete value is the percent of time buckets that are complete. The complete value is ignored when it is sent to the oscilloscope in the preamble. (See the ":WAVeform:COMPlete?" on page 988 query.)
X Units	The X-axis units of the waveform. (See the ":WAVeform:XUNits?" on page 1036 query.)
Y Units	The Y-axis units of the waveform. (See the ":WAVeform:YUNits?" on page 1042 query.)
Band Pass	The band pass consists of two values that are estimates of the maximum and minimum bandwidth limits of the source waveform. The bandwidth limit is computed as a function of the selected coupling and filter mode. (See the ":WAVeform:BANDpass?" on page 986 query.)

See Also

- ":WAVeform:DATA?" on page 991

:WAVeform:SEGMented:ALL

Command `:WAVeform:SEGMented:ALL {{ON | 1} | {OFF | 0}}`

The :WAVeform:SEGmented:ALL command configures the DATA query for rapidly downloading all segments in one query.

The <start> and <size> optional parameters for the DATA query are still supported and represent the start and size of the data for each segment.

Powering on the oscilloscope or performing a Default Setup sets this command to OFF.

There is complete backwards compatibility when this command is set to OFF.

The ON setting only supports channel and pod sources. If other sources such as functions are selected, a settings conflict message appears during the DATA query and no data is downloaded.

In segmented acquisition mode, with this command set to ON, the number of segments is appended to end of the waveform preamble.

Example This example turns on this command.

```
myScope.WriteString ":WAVeform:SEGMented:ALL ON"
```

Query `:WAVeform:SEGMented:ALL?`

This query returns the status of this command.

:WAVeform:SEGMented:COUNt?

Query `:WAVeform:SEGMented:COUNt?`

The :WAVeform:SEGMented:COUNt? query returns the index number of the last captured segment. A return value of zero indicates that the :ACQuire:MODE is not set to SEGMented.

<index_number> An integer number representing the index value of the last segment.

Returned Format [:WAVeform:SEGMented:COUNt] <index_number><NL>

Example This example returns the number of the last segment that was captured in the variable varIndex and prints it to the computer screen.

```
myScope.WriteString ":WAVEFORM:SEGMENTED:COUNT?"
varIndex = myScope.ReadNumber
Debug.Print FormatNumber(varIndex, 0)
```

:WAVeform:SEGMented:TTAG?

Query `:WAVeform:SEGMented:TTAG?`

The :WAVeform:SEGMented:TTAG? query returns the time difference between the first segment's trigger point and the trigger point of the currently displayed segment.

<delta_time> A real number in exponential format representing the time value difference between the first segment's trigger point and the currently displayed segment.

Returned Format [:WAVeform:SEGMented:TTAG] <delta_time><NL>

Example This example returns the time from the first segment's trigger point and the currently displayed segment's trigger point in the variable varDtime and prints it to the computer screen.

```
myScope.WriteString ":WAVEFORM:SEGMENTED:TTAG?"
varDtime = myScope.ReadNumber
Debug.Print FormatNumber(vardtime, 0)
```

:WAVeform:SEGMented:XLISt?

Query `:WAVeform:SEGMented:XLISt? {RELXorigin | ABSXorigin | TTAG}`

The :WAVeform:SEGMented:XLISt? query rapidly downloads x-parameter values for all segments.

RELXorigin = relative X origin for each segment.

ABSXorigin = relative origin + time tag for each segment

TTAG = time tag for each segment

This query uses the DATA query format for the returned data and supports all waveform command options including: BYTeorder, FORmat (only ASCii or BINary (float64 with 8 bytes per value), SOURce (only CHANnel<N>), STReaming, VIEW.

:WAVeform:SOURce

Command

```
:WAVeform:SOURce {CHANnel<N> | FUNCtion<F> | HISTogram | WMEMory<N>
                 | CLOCk | MTRend | MSPectrum | EQUalized | BUS<N>
                 | POD1 | POD2 | PODALL}
```

The :WAVeform:SOURce command selects a channel, function, waveform memory, or histogram as the waveform source.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

POD1 is the activated digital channels in the D0-D7 set while POD2 is the activated digital channels in the D8-D15 set. PODALL is all the activated digital channels.

<N> CHANnel<N> is an integer, 1-4.

BUS<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example selects channel 1 as the waveform source.

```
myScope.WriteString ":WAVeform:SOURce CHANnel1"
```

Query

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected waveform source.

Returned Format

```
[:WAVeform:SOURce] {CHANnel<N> | FUNCtion<F> | HISTogram |  WMEMory<N>
                   | CLOCk | MTRend | MSPectrum | EQUalized | BUS<N>
                   | POD1 | POD2 | PODALL}<NL>
```

Example This example places the current selection for the waveform source in the string variable, strSelection, then prints the contents of the variable to the computer's screen.

```
Dim strSelection As String   ' Dimension variable.
myScope.WriteString ":WAVeform:SOURce?"
strSelection = myScope.ReadString
Debug.Print strSelection
```

:WAVeform:STReaming

Command `:WAVeform:STReaming {{ON | 1} | {OFF | 0}}`

When enabled, :WAVeform:STReaming allows more than 999,999,999 bytes of data to be transfered from the Infiniium oscilloscope to a PC when using the :WAVeform:DATA? query. See the :WAVeform:DATA? query for information on receiving this much data.

Example This example turns on the streaming feature.

```
myScope.WriteString ":WAVeform:STReaming ON"
```

Query `:WAVeform:STReaming?`

The :WAVeform:STReaming? query returns the status of the streaming feature.

Returned Format `[:WAVeform:STReaming] {1 | 0}<NL>`

:WAVeform:TYPE?

Query `:WAVeform:TYPE?`

The :WAVeform:TYPE? query returns the current acquisition data type for the currently selected source. The type returned describes how the waveform was acquired. The waveform type may be:

- RAW — data consists of one data point in each time bucket with no interpolation.
- AVERage — data consists of the average of the first n hits in a time bucket, where n is the value in the count portion of the preamble. Time buckets that have fewer than n hits return the average of the data they contain. If the :ACQuire:COMPlete parameter is set to 100%, then each time bucket must contain the number of data hits specified with the :ACQuire:AVERage:COUNt command.
- VHIStogram — data is a vertical histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.
- HHIStogram — data is a horizontal histogram. Histograms are transferred using the LONGLONG format. They can be generated using the Histogram subsystem commands.
- INTerpolate — In the INTerpolate acquisition type, the last data point in each time bucket is stored, and additional data points between the acquired data points are filled by interpolation.
- DIGITAL — data consists of digital pod or bus values for each time bucket.
- PDETect — data consists of two data points in each time bucket: the minimum values and the maximum values.

Returned Format `[:WAVeform:TYPE] {RAW | AVER | VHIS | HHIS | INT | DIGITAL | PDET}<NL>`

Example This example places the current acquisition data type in the string variable, strType, then prints the contents of the variable to the computer's screen.

```
Dim strType As String   ' Dimension variable.
myScope.WriteString ":WAVeform:TYPE?"
strType = myScope.ReadString
Debug.Print strType
```

:WAVeform:VIEW

Command `:WAVeform:VIEW {ALL | MAIN | WINDow}`

The :WAVeform:VIEW command selects which view of the waveform is selected for data and preamble queries. You can set the command to ALL, MAIN, or WINDow. The view has different meanings depending upon the waveform source selected. The default setting for this command is ALL.

Channels For channels, you may select ALL, MAIN, or WINDow views. If you select ALL, all of the data in the waveform record is referenced. If you select MAIN, only the data in the main time base range is referenced. The first value corresponds to the first time bucket in the main time base range, and the last value corresponds to the last time bucket in the main time base range. If WINDow is selected, only data in the delayed view is referenced. The first value corresponds to the first time bucket in the delayed view and the last value corresponds to the last time bucket in the delayed view.

Memories For memories, if you specify ALL, all the data in the waveform record is referenced. WINDow and MAIN refer to the data contained in the memory time base range for the particular memory. The first value corresponds to the first time bucket in the memory time base range, and the last value corresponds to the last time bucket in the memory time base range.

Functions For functions, ALL, MAIN, and WINDow refer to all of the data in the waveform record.

Table 29 summarizes the parameters for this command for each source.

Example This example sets up the oscilloscope to view all of the data.

```
myScope.WriteString ":WAVEFORM:VIEW ALL"
```

Table 29 Waveform View Parameters

Source/Parameter	ALL	MAIN	WINDow
CHANNEL	All data	Main time base	Zoom
MEMORY	All data	Memory time base	Memory time base
FUNCTION	All data	All data	All data

Query `:WAVeform:VIEW?`

The :WAVeform:VIEW? query returns the currently selected view.

Returned Format `[:WAVeform:VIEW] {ALL | MAIN | WINDow}<NL>`

Example This example returns the current view setting to the string variable, strSetting, then prints the contents of the variable to the computer's screen.

```
Dim strSetting As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:VIEW?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:WAVeform:XDISplay?

Query `:WAVeform:XDISplay?`

The :WAVeform:XDISplay? query returns the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. For VERSus type waveforms, it is the value at the center of the X-axis of the display. This value is treated as a double precision 64-bit floating point number.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XDISplay] <value><NL>`

<value> A real number representing the X-axis value at the left edge of the display.

Example This example returns the X-axis value at the left edge of the display to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:XDISPLAY?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:WAVeform:XINCrement?

Query `:WAVeform:XINCrement?`

The :WAVeform:XINCrement? query returns the duration between consecutive data points for the currently specified waveform source. For time domain waveforms, this is the time difference between consecutive data points. For VERSus type waveforms, this is the duration between levels on the X axis. For voltage waveforms, this is the voltage corresponding to one level.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XINCrement] <value><NL>`

<value> A real number representing the duration between data points on the X axis.

Example This example places the current X-increment value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:XINCREMENT?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also You can obtain the X-increment value through the :WAVeform:PREamble? query.

:WAVeform:XORigin?

Query `:WAVeform:XORigin?`

The :WAVeform:XORigin? query returns the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. For VERSus type waveforms, it is the X-axis value at level zero. For voltage waveforms, it is the voltage at level zero. The value returned by this query is treated as a double precision 64-bit floating point number.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XORigin] <value><NL>`

<value> A real number representing the X-axis value of the first data point in the data record.

Example This example places the current X-origin value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:XORIGIN?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also You can obtain the X-origin value through the :WAVeform:PREamble? query.

:WAVeform:XRANge?

Query `:WAVeform:XRANge?`

The :WAVeform:XRANge? query returns the X-axis duration of the displayed waveform. For time domain waveforms, it is the duration of the time across the display. For VERSus type waveforms, it is the duration of the waveform that is displayed on the X axis.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:XRANge] <value><NL>`

<value> A real number representing the X-axis duration of the displayed waveform.

Example This example returns the X-axis duration of the displayed waveform to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:XRANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:WAVeform:XREFerence?

Query `:WAVeform:XREFerence?`

The :WAVeform:XREFerence? query returns the data point or level associated with the X-origin data value. It is at this data point or level that the X origin is defined. In this oscilloscope, the value is always zero.

Returned Format `[:WAVeform:XREFerence] 0<NL>`

Example This example places the current X-reference value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:XREFERENCE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also You can obtain the X-reference value through the :WAVeform:PREamble? query.

:WAVeform:XUNits?

Query `:WAVeform:XUNits?`

The :WAVeform:XUNits? query returns the X-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format `[:WAVeform:XUNits] {UNKNown | VOLT | SECond | CONStant | AMP | DECibels | HERTz | WATT}<NL>`

Example This example returns the X-axis units of the currently selected waveform source to the string variable, strUnit, then prints the contents of the variable to the computer's screen.

```
Dim strUnit As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:XUNITS?"
strUnit = myScope.ReadString
Debug.Print strUnit
```

:WAVeform:YDISplay?

Query `:WAVeform:YDISplay?`

The :WAVeform:YDISplay? query returns the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:YDISplay] <value><NL>`

<value> A real number representing the Y-axis value at the center of the display.

Example This example returns the current Y-display value to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":":WAVEFORM:YDISPLAY?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:WAVeform:YINCrement?

Query `:WAVeform:YINCrement?`

The :WAVeform:YINCrement? query returns the y-increment voltage value for the currently specified source. This voltage value is the voltage difference between two adjacent waveform data digital codes. Adjacent digital codes are codes that differ by one least significant bit. For example, the digital codes 24680 and 24681 vary by one least significant bit.

- For BYTE and WORD data, and voltage waveforms, it is the voltage corresponding to one least significant bit change.
- For ASCii data format, the YINCrement is the full scale voltage range covered by the A/D converter.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:YINCrement] <real_value><NL>`

<real_value> A real number in exponential format.

Example This example places the current Y-increment value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:YINCREMENT?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1008.

You can also obtain the Y-increment value through the :WAVeform:PREamble? query.

:WAVeform:YORigin?

Query `:WAVeform:YORigin?`

The :WAVeform:YORigin? query returns the y-origin voltage value for the currently specified source. The voltage value returned is the voltage value represented by the waveform data digital code 00000.

- For BYTE and WORD data, and voltage waveforms, it is the voltage at digital code zero.
- For ASCii data format, the YORigin is the Y-axis value at the center of the data range. Data range is returned in the Y increment.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:YORigin] <real_value><NL>`

<real_value> A real number in exponential format.

Example This example places the current Y-origin value in the numeric variable, varCenter, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:YORIGIN?"
varCenter = myScope.ReadNumber
Debug.Print FormatNumber(varCenter, 0)
```

See Also For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1008.

You can obtain the Y-origin value through the :WAVeform:PREamble? query.

:WAVeform:YRANge?

Query `:WAVeform:YRANge?`

The :WAVeform:YRANge? query returns the Y-axis duration of the displayed waveform. For voltage waveforms, it is the voltage across the entire display.

NOTE A "Waveform data is not valid" error occurs when there is no data available for a channel. When this occurs, a zero value is returned.

Returned Format `[:WAVeform:YRANge] <value><NL>`

<value> A real number representing the Y-axis duration of the displayed waveform.

Example This example returns the current Y-range value to the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:YRANGE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

:WAVeform:YREFerence?

Query `:WAVeform:YREFerence?`

The :WAVeform:YREFerence? query returns the y-reference voltage value for the currently specified source. It is at this level that the Y origin is defined. In this oscilloscope, the value is always zero.

Returned Format `[:WAVeform:YREFerence] 0<NL>`

Example This example places the current Y-reference value for the currently specified source in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":WAVEFORM:YREFERENCE?"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

See Also For more information on BYTE and WORD formats, see "Understanding WORD and BYTE Formats" on page 1008.

You can obtain the Y-reference value through the :WAVeform:PREamble? query.

:WAVeform:YUNits?

Query `:WAVeform:YUNits?`

The :WAVeform:YUNits? query returns the Y-axis units of the currently selected waveform source. The currently selected source may be a channel, function, or waveform memory.

Returned Format `[:WAVeform:YUNits] {UNKNown | VOLT | SECond | HITS | DECibels | CONStant | AMP}<NL>`

Example This example returns the Y-axis units of the currently selected waveform source to the string variable, strUnit, then prints the contents of the variable to the computer's screen.

```
Dim strUnit As String   ' Dimension variable.
myScope.WriteString ":WAVEFORM:YUNITS?"
strUnit = myScope.ReadString
Debug.Print strUnit
```

35 Waveform Memory Commands

:WMEMory<N>:CLEar 1044
:WMEMory<N>:DISPlay 1045
:WMEMory<N>:LOAD 1046
:WMEMory<N>:SAVE 1047
:WMEMory<N>:XOFFset 1048
:WMEMory<N>:XRANge 1049
:WMEMory<N>:YOFFset 1050
:WMEMory<N>:YRANge 1051

The Waveform Memory Subsystem commands let you save and display waveforms, memories, and functions.

NOTE

<N> in WMEMory<N> Indicates the Waveform Memory Number

In Waveform Memory commands, the <N> in WMEMory<N> represents the waveform memory number (1-4).

:WMEMory<N>:CLEar

Command `:WMEMory<N>:CLEar`

The :WMEMory<N>:CLEar clears the associated wave memory.

<N> The memory number is an integer from 1 to 4.

Example This example clears the waveform memory 1.

```
myScope.WriteString ":WMEMORY1:CLEar"
```

:WMEMory<N>:DISPlay

Command `:WMEMory<N>:DISPlay {{ON | 1} | {OFF | 0}}`

The :WMEMory<N>:DISPlay command enables or disables the viewing of the selected waveform memory.

<N> The memory number is an integer from 1 to 4.

Example This example turns on the waveform memory 1 display.

```
myScope.WriteString ":WMEMORY1:DISPLAY ON"
```

Query `:WMEMory<N>:DISPlay?`

The :WMEMory<N>:DISPlay? query returns the state of the selected waveform memory.

Returned Format `[:WMEMory<N>:DISPlay] {1 | 0}<NL>`

:WMEMory<N>:LOAD

Command `:WMEMory<N>:LOAD <file_name>`

The :WMEMory<N>:LOAD command loads an oscilloscope waveform memory location with a waveform from a file that has an internal waveform format (extension .wfm), comma separated xypairs, (extension .csv), tab separated xypairs (extension .tsv), and yvalues text (extension .txt). You can load the file from either the c: or a: drive, or any lan connected drive. See the examples below.

The oscilloscope assumes that the default path for waveforms is c:\Document and Settings\All Users\Shared Documents\Infiniium\Data. To use a different path, specify the path and file name completely.

<N> The memory number is an integer from 1 to 4.

<file_name> A quoted string which specifies the file to load, and has a .wfm, .csv, .tsv, or .txt extension.

Examples This example loads waveform memory 4 with a file.

```
myScope.WriteString _
":WMEMORY4:LOAD ""c:\Document and Settings\All Users\Shared Documents\
Infiniium\Data\waveform.wfm"""
```

This example loads waveform memory 3 with a file that has the internal waveform format and is stored on drive U:.

```
myScope.WriteString ":WMEMORY3:LOAD ""U:\waveform.wfm"""
```

Related Commands `:DISK:LOAD`
`:DISK:STORe`

See Also

- ":DISK:LOAD" on page 286
- ":DISK:SAVE:SETup" on page 295
- ":DISK:SAVE:WAVeform" on page 296

:WMEMory<N>:SAVE

Command `:WMEMory<N>:SAVE {CHANnel<N> |CLOCk | FUNCtion<F> | MTRend | MSPectrum | WMEMory<N>}`

The :WMEMory<N>:SAVE command stores the specified channel, waveform memory, or function to the waveform memory. You can save waveforms to waveform memories regardless of whether the waveform memory is displayed or not.

The :WAVeform:VIEW command determines the view of the data being saved.

<N> CHANnel<N> is an integer, 1-4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

Example This example saves channel 1 to waveform memory 4.

```
myScope.WriteString ":WMEMORY4:SAVE CHANNEL1"
```

:WMEMory<N>:XOFFset

Command `:WMEMory<N>:XOFFset <offset_value>`

The :WMEMory<N>:XOFFset command sets thex-axis, horizontal position for the selected waveform memory's display scale. The position is referenced to center screen.

<N> The memory number is an integer from 1 to 4.

<offset_value> A real number for the horizontal offset (position) value.

Example This example sets the X-axis, horizontal position for waveform memory 3 to 0.1 seconds (100 ms).

```
myScope.WriteString ":WMEMORY3:XOFFSET 0.1"
```

Query `:WMEMory<N>:XOFFset?`

The :WMEMory<N>:XOFFset? query returns the current X-axis, horizontal position for the selected waveform memory.

Returned Format `[:WMEMory<N>:XOFFset] <offset_value><NL>`

:WMEMory<N>:XRANge

Command `:WMEMory<N>:XRANge <range_value>`

The :WMEMory<N>:XRANge command sets theX-axis, horizontal range for the selected waveform memory's display scale. The horizontal scale is the horizontal range divided by 10.

<N> The memory number is an integer from 1 to 4.

<range_value> A real number for the horizontal range value.

Example This example sets the X-axis, horizontal range of waveform memory 2 to 435 microseconds.

```
myScope.WriteString ":WMEMORY2:XRANGE 435E-6"
```

Query `:WMEMory<N>:XRANge?`

The :WMEMory<N>:XRANge? query returns the current X-axis, horizontal range for the selected waveform memory.

Returned Format `[:WMEMory<N>:XRANge] <range_value><NL>`

:WMEMory<N>:YOFFset

Command `:WMEMory<N>:YOFFset <offset_value>`

The :WMEMory<N>:YOFFset command sets the Y-axis (vertical axis) offset for the selected waveform memory.

<N> The memory number is an integer from 1 to 4.

<offset_value> A real number for the vertical offset value.

Example This example sets the Y-axis (vertical) offset of waveform memory 2 to 0.2V.

```
myScope.WriteString ":WMEMORY2:YOFFSET 0.2"
```

Query `:WMEMory<N>:YOFFset?`

The :WMEMory<N>:YOFFset? query returns the current Y-axis (vertical) offset for the selected waveform memory.

Returned Format `[:WMEMory<N>:YOFFset] <offset_value><NL>`

:WMEMory<N>:YRANge

Command `:WMEMory<N>:YRANge <range_value>`

The :WMEMory<N>:YRANge command sets the Y-axis, vertical range for the selected memory. The vertical scale is the vertical range divided by 8.

<N> The memory number is an integer from 1 to 4.

<range_value> A real number for the vertical range value.

Example This example sets the Y-axis (vertical) range of waveform memory 3 to 0.2 volts.

```
myScope.WriteString ":WMEMORY3:YRANGE 0.2"
```

Query `:WMEMory<N>:YRANge?`

The :WMEMory<N>:YRANge? query returns the Y-axis, vertical range for the selected memory.

Returned Format `[:WMEMory<N>:YRANge]<range_value><NL>`

36 Obsolete and Discontinued Commands

Obsolete commands are deprecated, older forms of commands that still work but have been replaced by newer commands.

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:CLOCk:METHod (see page 1054)	• :MEASure:CLOCk:METHod (see page 502) • :MEASure:CLOCk:METHod:JTF (see page 508) • :MEASure:CLOCk:METHod:OJTF (see page 510)	The command options for specifying clock recovery PLL options have been moved to the new commands :MEASure:CLOCk:METHod:JTF and :MEASure:CLOCk:METHod:OJTF.
:MEASure:DDPWS (see page 1057)	:MEASure:RJDJ:ALL? (see page 624)	The :MEASure:RJDJ:ALL? query returns all of the RJDJ jitter measurements.

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the Infiniium 90000 Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
:DISK:STORe	":DISK:SAVE:SETup" on page 295 ":DISK:SAVE:WAVeform" on page 296	For saving setups and waveforms to disk.

:MEASure:CLOCk:METHod

(deprecated)

Command

NOTE Some of these command options have been deprecated — options for specifying clock recovery PLL options have been moved to the new commands ":MEASure:CLOCk:METHod:JTF" on page 508 and ":MEASure:CLOCk:METHod:OJTF" on page 510. See also ":MEASure:CLOCk:METHod" on page 502.

NOTE :MEASure:CLOCk commands are available when clock recovery is used by licensed software (like the E2688A High Speed Serial software or the N5400A/N5401A EZJIT Plus jitter analysis software).

```
:MEASure:CLOCk:METHod {FOPLL,<data_rate>,<loop_bandwidth>}
    | {EQFOPLL,<data_rate>,<loop_bandwidth>}
    | {SOPLL,<data_rate>,<loop_bandwidth>, <damping_factor>}
    | {EQSOPLL,<data_rate>,<loop_bandwidth>, <damping_factor>}
    | {PCIE,{DEEMphasis | TRANsition | BOTH}}
    | {FC,{FC1063 | FC2125 | FC425}}
    | {EXPFOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<loop_bandwidth>}
    | {EXPSOPLL,<source>,{RISing | FALLing | BOTH},
      <multiplier>,<clock_freq>,<loop_bandwidth>,<damping_fact>}
    | {EXPlicit,<source>,{RISing | FALLing | BOTH}[,<multiplier>]}
    | {FIXed,{AUTO | {SEMI[,<data_rate>]} | <data_rate>}}
    | {FLEXR,<baud_rate>}
    | {FLEXT,<baud_rate>}
```

The :MEASure:CLOCk:METHod command sets the clock recovery method to:

- FOPLL (first order phase-locked loop).
- SOPLL (second order phase-locked loop).
- EQFOPLL (equalized first order phase-locked loop).
- EQSOPLL (equalized second order phase-locked loop).
- PCIE (PCI Express).
- FC (Fibre Channel).
- EXPFOPLL (Explicit First Order PLL).
- EXPSOPLL (Explicit Second Order PLL).
- EXPlicit (Explicit Clock).
- FIXed (Constant Frequency).
- FLEXR (FlexRay Receiver).
- FLEXT (FlexRay Transmitter).

The EQUalized clock recovery methods are only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled.

<source> {CHANnel<N> | FUNCtion<N> | WMEMory<N>}

<N> CHANnel<N> is an integer, 1-4.

FUNCtion<N> and WMEMory<N> are:

An integer, 1-4, representing the selected function or waveform memory.

<data_rate> A real number for the base data rate in Hertz.

<damping_factor> A real number for the damping factor of the PLL in bits per second.

<loop_bandwidth> A real number for the cutoff frequency for the PLL to track.

<multiplier> An integer used as the multiplication factor.

<clock_freq> A real number used for the clock frequency of the PLL.

<track_freq> A real number used for the tracking frequency of the PLL.

<damping_fact> A real number used for the damping factor of the PLL.

<baud_rate> A real number used for the baud rate.

Example This example sets the clock recovery method to phase-locked loop.

```
myScope.WriteString ":MEASURE:CLOCK:METHOD FOPLL,2E9,1.19E6"
```

Query

```
:MEASure:CLOCk:METHod?
```

The :MEASure:CLOCk:METHod? query returns the state of the clock recovery method.

Returned Format

```
[:MEASure:CLOCk:METHod] {FOPLL,<data_rate>,<loop_bandwidth>}
    | {EQFOPLL,<data_rate>,<loop_bandwidth>}
    | {SOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>}
    | {EQSOPLL,<data_rate>,<loop_bandwidth>,<damping_factor>}
    | {PCIE,{DEEMphasis | TRANsition | BOTH}}
    | {FC,{FC1063 | FC2125 | FC425}}
    | {EXPFOPLL <source>,{RISing | FALLing | BOTH},
       <multiplier>,<clock_freq>,<track_freq>}
    | {EXPSOPLL <source>,{RISing | FALLing | BOTH},
       <multiplier>,<clock_freq>,<track_freq>,<damping_fact>}
    | {EXPlict,<source>,{RISing | FALLing | BOTH},<multiplier>}
    | {FIXed,{AUTO | {SEMI,<data_rate>} | <data_rate>}}
    | {FLEXR,<baud_rate>}
    | {FLEXT,<baud_rate>}
```

Example This example places the current setting of the clock recovery method in the variable strSetting, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTem:HEADer OFF"
myScope.WriteString ":MEASURE:CLOCK:METHOD?"
strSetting = myScope.ReadString
Debug.Print strSetting
```

:MEASure:DDPWS

(Data Dependent Pulse Width Shrinkage)

Command

NOTE This command is deprecated. In its place, use the query ":MEASure:RJDJ:ALL?" on page 624 which returns all of the RJDJ jitter measurements.

NOTE This command is only available when the E2681A Jitter Analysis Software or the N5400A/N5401A Software is installed.

```
:MEASure:DDPWS <source>
```

The :MEASure:DDPWS command measures the data dependent pulse width shrinkage for the selected source.

<source> {CHANnel<N> | FUNCtion<F> | WMEMory<N> | CLOCk | MTRend | MSPectrum | EQUalized}

MTRend and MSPectrum sources are only available if the oscilloscope has the EZJIT option installed and the feature is enabled.

The CLOCk source is only available if the oscilloscope has the High Speed Serial option installed and the feature is enabled.

The EQUalized source is only available if the oscilloscope has the High Speed Serial option and the Serial Data Equalization option installed and the features are enabled. This command uses the Feed-Forward Equalized (FFE) signal as the source.

<N> CHANnel<N> is an integer, 1- 4.

WMEMory<N> is an integer, 1-4.

<F> FUNCtion<F> is an integer, 1-16.

Example This example measures the data rate of channel 1.

```
myScope.WriteString ":MEASure:DDPWS CHANnel1"
```

Query

```
:MEASure:DDPWS? <source>
```

The :MEASure:DDPWS? query returns the measured data dependent pulse width shrinkage.

Returned Format

```
[:MEASure:DDPWS] <value><NL>
```

<value> Data dependent pulse width shrinkage in seconds for the selected source.

Example This example places the current data dependent pulse width shrinkage value of the channel 1 waveform in the numeric variable, varValue, then prints the contents of the variable to the computer's screen.

```
myScope.WriteString ":SYSTEM:HEADER OFF"    ' Response headers off.
myScope.WriteString ":MEASURE:DDPWS? CHANNEL1"
varValue = myScope.ReadNumber
Debug.Print FormatNumber(varValue, 0)
```

37 Error Messages

Error Queue 1060
Error Numbers 1061
Command Error 1062
Execution Error 1063
Device- or Oscilloscope-Specific Error 1064
Query Error 1065
List of Error Messages 1066

This chapter describes the error messages and how they are generated. The possible causes for the generation of the error messages are also listed in the following table.

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." Anytime the error queue overflows, the oldest 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).

Reading an error from the head of the queue removes that error from the queue, and opens a position at the tail of the queue for a new error. When all errors have been read from the queue, subsequent error queries return 0, "No error."

The error queue is cleared when any of the following occur:

- the instrument is powered up,
- a *CLS command is sent,
- the last item from the queue is read, or
- the instrument is switched from talk only to addressed mode on the front panel.

Error Numbers

The error numbers are grouped according to the type of error that is detected.

- +0 indicates no errors were detected.
- -100 to -199 indicates a command error was detected
- -200 to -299 indicates an execution error was detected.
- -300 to -399 indicates a device-specific error was detected.
- -400 to-499 indicates a query error was detected.
- +1 to +32767 indicates an oscilloscope specific error has been detected.

Command Error

An error number in the range -100 to -199 indicates that an IEEE 488.2 syntax error has been detected by the instrument's parser. The occurrence of any error in this class sets the command error bit (bit 5) in the event status register and indicates that one of the following events occurred:

- An IEEE 488.2 syntax error was detected by the parser. That is, a computer-to-oscilloscope message was received that is in violation of the IEEE 488.2 standard. This may be a data element that violates the oscilloscope's listening formats, or a data type that is unacceptable to the oscilloscope.
- An unrecognized header was received. Unrecognized headers include incorrect oscilloscope-specific headers and incorrect or unimplemented IEEE 488.2 common commands.
- A Group Execute Trigger (GET) was entered into the input buffer inside of an IEEE 488.2 program message.

Events that generate command errors do not generate eexecution errors, oscilloscope-specific errors, or query errors.

Execution Error

An error number in the range -200 to -299 indicates that an error was detected by the instrument's execution control block. The occurrence of any error in this class causes the execution error bit (bit 4) in the event status register to be set. It also indicates that one of the following events occurred:

- The program data following a header is outside the legal input range or is inconsistent with the oscilloscope's capabilities.
- A valid program message could not be properly executed due to some oscilloscope condition.

Execution errors are reported by the oscilloscope after expressions are evaluated and rounding operations are completed. For example, rounding a numeric data element will not be reported as an execution error. Events that generate execution errors do not generate command errors, oscilloscope specific errors, or query errors.

Device- or Oscilloscope-Specific Error

An error number in the range of -300 to -399 or +1 to +32767 indicates that the instrument has detected an error caused by an oscilloscope operation that did not properly complete. This may be due to an abnormal hardware or firmware condition. For example, this error may be generated by a self-test response error, or a full error queue. The occurrence of any error in this class causes the oscilloscope-specific error bit (bit 3) in the event status register to be set.

Query Error

An error number in the range-400 to-499 indicates that the output queue control of the instrument has detected a problem with the message exchange protocol. An occurrence of any error in this class should cause the query error bit (bit 2) in the event status register to be set. An occurrence of an error also means one of the following is true:

- An attempt is being made to read data from the output queue when no output is either present or pending.
- Data in the output queue has been lost.

List of Error Messages

Table 30 a list of the error messages that are returned by the parser on this oscilloscope.

Table 30 Error Messages

0	No error	The error queue is empty. Every error in the queue has been read (SYSTEM:ERROR? query) or the queue was cleared by power-up or *CLS.
-100	Command error	This is the generic syntax error used if the oscilloscope cannot detect more specific errors.
-101	Invalid character	A syntactic element contains a character that is invalid for that type.
-102	Syntax error	An unrecognized command or data type was encountered.
-103	Invalid separator	The parser was expecting a separator and encountered an illegal character.
-104	Data type error	The parser recognized a data element different than one allowed. For example, numeric or string data was expected but block data was received.
-105	GET not allowed	A Group Execute Trigger was received within a program message.
-108	Parameter not allowed	More parameters were received than expected for the header.
-109	Missing parameter	Fewer parameters were received than required for the header.
-112	Program mnemonic too long	The header or character data element contains more than twelve characters.
-113	Undefined header	The header is syntactically correct, but it is undefined for the oscilloscope. For example, *XYZ is not defined for the oscilloscope.
-121	Invalid character in number	An invalid character for the data type being parsed was encountered. For example, a "9" in octal data.
-123	Numeric overflow	Number is too large or too small to be represented internally.
-124	Too many digits	The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros.
-128	Numeric data not allowed	A legal numeric data element was received, but the oscilloscope does not accept one in this position for the header.
-131	Invalid suffix	The suffix does not follow the syntax described in IEEE 488.2 or the suffix is inappropriate for the oscilloscope.
-138	Suffix not allowed	A suffix was encountered after a numeric element that does not allow suffixes.
-141	Invalid character data	Either the character data element contains an invalid character or the particular element received is not valid for the header.
-144	Character data too long	

Table 30 Error Messages (continued)

-148	Character data not allowed	A legal character data element was encountered where prohibited by the oscilloscope.
-150	String data error	This error can be generated when parsing a string data element. This particular error message is used if the oscilloscope cannot detect a more specific error.
-151	Invalid string data	A string data element was expected, but was invalid for some reason. For example, an END message was received before the terminal quote character.
-158	String data not allowed	A string data element was encountered but was not allowed by the oscilloscope at this point in parsing.
-160	Block data error	This error can be generated when parsing a block data element. This particular error message is used if the oscilloscope cannot detect a more specific error.
-161	Invalid block data	
-168	Block data not allowed	A legal block data element was encountered but was not allowed by the oscilloscope at this point in parsing.
-170	Expression error	This error can be generated when parsing an expression data element. It is used if the oscilloscope cannot detect a more specific error.
-171	Invalid expression	
-178	Expression data not allowed	Expression data was encountered but was not allowed by the oscilloscope at this point in parsing.
-200	Execution error	This is a generic syntax error which is used if the oscilloscope cannot detect more specific errors.
-212	Arm ignored	
-213	Init ignored	
-214	Trigger deadlock	
-215	Arm deadlock	
-220	Parameter error	
-221	Settings conflict	
-222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value is outside the legal range defined by the oscilloscope.
-223	Too much data	Indicates that a legal program data element of block, expression, or string type was received that contained more data than the oscilloscope could handle due to memory or related oscilloscope-specific requirements.
-224	Illegal parameter value	
-230	Data corrupt or stale	

Table 30 Error Messages (continued)

-231	Data questionable	
-240	Hardware error	
-241	Hardware missing	
-250	Mass storage error	
-251	Missing mass storage	
-252	Missing media	
-253	Corrupt media	
-254	Media full	
-255	Directory full	
-256	File name not found	
-257	File name error	
-258	Media protected	
-260	Expression error	
-261	Math error in expression	
-300	Device specific error	
-310	System error	Indicates that a system error occurred.
-311	Memory error	
-312	PUD memory error	
-313	Calibration memory lost	
-314	Save/recall memory lost	
-315	Configuration memory lost	
-321	Out of memory	
-330	Self-test failed	
-350	Queue overflow	Indicates that there is no room in the error queue and an error occurred but was not recorded.
-370	No sub tests are defined for the selected self test	
-371	Self Test status is corrupt or no self test has been executed	
-372	This product configuration does not support the requested self test	
-373	This product configuration does not support the requested source	

Table 30 Error Messages (continued)

-374	The requested self test log file could not be found	
-375	Attenuator relay actuation counts can only be modified during factory service	
-400	Query error	This is the generic query error.
-410	Query INTERRUPTED	
-420	Query UNTERMINATED	
-430	Query DEADLOCKED	
-440	Query UNTERMINATED after indefinite response	

38 Sample Programs

VISA COM Examples 1072
VISA Examples 1111
SICL Examples 1162
SCPI.NET Examples 1181

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

VISA COM Examples

- "VISA COM Example in Visual Basic" on page 1072
- "VISA COM Example in C#" on page 1083
- "VISA COM Example in Visual Basic .NET" on page 1093
- "VISA COM Example in Python" on page 1103

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Reference the Agilent VISA COM library:
   a. Choose **Tools>References...** from the main menu.
   b. In the References dialog, check:
      - VISA COM 3.0 Type Library
      - Microsoft Scripting Runtime
   c. Click **OK**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the VISA address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Agilent VISA COM Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent Infiniium Series oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

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

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ----------------------------------------------------------------------
```

```
Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = _
      myMgr.Open("TCPIP0::130.29.71.191::inst0::INSTR")
  myScope.IO.Timeout = 15000   ' Set I/O communication timeout.
  myScope.IO.Clear   ' Clear the interface.

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  Exit Sub

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

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  Debug.Print "Identification string: " + strQueryResult

  ' Load the default setup.
  DoCommand "*RST"

  Exit Sub

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

End Sub

'
```

```
' Capture the waveform.
' -----------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo VisaComError

  ' Set probe attenuation factor.
  DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically set up oscilloscope.
  ' -----------------------------------------------------------------------
  Debug.Print "Autoscale."
  DoCommand ":AUToscale"

  ' Set trigger mode.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope setup.
  ' -----------------------------------------------------------------------
  varQueryResult = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

  ' Output setup string to a file:
  Dim strPath As String
  strPath = "c:\scope\config\setup.dat"
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Put hFile, , varQueryResult   ' Write data.
  Close hFile   ' Close file.
  Debug.Print "Setup bytes saved: " + CStr(LenB(varQueryResult))

  ' Change oscilloscope settings with individual commands:
  ' -----------------------------------------------------------------------

  ' Set vertical scale and offset.
  DoCommand ":CHANnel1:SCALe 0.1"
  Debug.Print "Channel 1 vertical scale: " + _
      DoQueryString(":CHANnel1:SCALe?")
```

```
  DoCommand ":CHANnel1:OFFSet 0.0"
  Debug.Print "Channel 1 vertical offset: " + _
      DoQueryString(":CHANnel1:OFFSet?")

  ' Set horizontal scale and offset.
  DoCommand ":TIMebase:SCALe 200E-6"
  Debug.Print "Timebase scale: " + _
      DoQueryString(":TIMebase:SCALe?")

  DoCommand ":TIMebase:POSition 0.0"
  Debug.Print "Timebase position: " + _
      DoQueryString(":TIMebase:POSition?")

  ' Set the acquisition mode.
  DoCommand ":ACQuire:MODE RTIMe"
  Debug.Print "Acquire mode: " + _
      DoQueryString(":ACQuire:MODE?")

  ' Or, configure by loading a previously saved setup.
  ' ------------------------------------------------------------------
  Dim varSetupString As Variant
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As hFile    ' Open file for input.
  Get hFile, , varSetupString    ' Read data.
  Close hFile    ' Close file.
  ' Write learn string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  DoCommandIEEEBlock ":SYSTem:SETup", varSetupString
  Debug.Print "Setup bytes restored: " + CStr(LenB(varSetupString))

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' ------------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

  Exit Sub

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

End Sub

'
' Analyze the captured waveform.
' --------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' Make measurements.
  ' ------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")
```

```
DoCommand ":MEASure:FREQuency"
varQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"

DoCommand ":MEASure:VAMPlitude"
varQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
MsgBox "Vertical amplitude:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"


' Download the screen image.
' ----------------------------------------------------------------------
' Get screen image.
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG")

' Save screen image to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.png"
If Len(Dir(strPath)) Then
  Kill strPath   ' Remove file if it exists.
End If

Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , byteData   ' Write data.
Close hFile   ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
    " bytes) written to " + strPath


' Download waveform data.
' ----------------------------------------------------------------------

' Get the waveform type.
Debug.Print "Waveform type: " + _
    DoQueryString(":WAVeform:TYPE?")

' Get the number of waveform points.
Debug.Print "Waveform points available: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned:
DoCommand ":WAVeform:FORMat WORD"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings from preamble:
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 intCoupling As Integer
Dim dblXDispRange As Double
Dim dblXDispOrigin As Double
Dim dblYDispRange As Double
Dim dblYDispOrigin As Double
Dim strDate As String
Dim strTime As String
Dim strFrameModel As String
Dim intAcqMode As Integer
Dim intCompletion As Integer
Dim intXUnits As Integer
Dim intYUnits As Integer
Dim dblMaxBwLimit As Double
Dim dblMinBwLimit As Double

Dim dctWavFormat As Scripting.Dictionary
Set dctWavFormat = New Scripting.Dictionary
dctWavFormat.Add 0, "ASCii"
dctWavFormat.Add 1, "BYTE"
dctWavFormat.Add 2, "WORD"
dctWavFormat.Add 3, "LONG"
dctWavFormat.Add 4, "LONGLONG"

Dim dctAcqType As Scripting.Dictionary
Set dctAcqType = New Scripting.Dictionary
dctAcqType.Add 1, "RAW"
dctAcqType.Add 2, "AVERage"
dctAcqType.Add 3, "VHIStogram"
dctAcqType.Add 4, "HHIStogram"
dctAcqType.Add 6, "INTerpolate"
dctAcqType.Add 10, "PDETect"

Dim dctAcqMode As Scripting.Dictionary
Set dctAcqMode = New Scripting.Dictionary
dctAcqMode.Add 0, "RTIMe"
dctAcqMode.Add 1, "ETIMe"
dctAcqMode.Add 3, "PDETect"

Dim dctCoupling As Scripting.Dictionary
Set dctCoupling = New Scripting.Dictionary
dctCoupling.Add 0, "AC"
dctCoupling.Add 1, "DC"
dctCoupling.Add 2, "DCFIFTY"
dctCoupling.Add 3, "LFREJECT"

Dim dctUnits As Scripting.Dictionary
Set dctUnits = New Scripting.Dictionary
```

```
dctUnits.Add 0, "UNKNOWN"
dctUnits.Add 1, "VOLT"
dctUnits.Add 2, "SECOND"
dctUnits.Add 3, "CONSTANT"
dctUnits.Add 4, "AMP"
dctUnits.Add 5, "DECIBEL"

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
intCoupling = Preamble(10)
dblXDispRange = Preamble(11)
dblXDispOrigin = Preamble(12)
dblYDispRange = Preamble(13)
dblYDispOrigin = Preamble(14)
strDate = Preamble(15)
strTime = Preamble(16)
strFrameModel = Preamble(17)
intAcqMode = Preamble(18)
intCompletion = Preamble(19)
intXUnits = Preamble(20)
intYUnits = Preamble(21)
dblMaxBwLimit = Preamble(22)
dblMinBwLimit = Preamble(23)

Debug.Print "Waveform format: " + dctWavFormat.Item(intFormat)
Debug.Print "Acquisition type: " + dctAcqType.Item(intType)

Debug.Print "Waveform points desired: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(sngYOrigin, 0)
```

```
Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

Debug.Print "Coupling: " + dctCoupling.Item(intCoupling)

Debug.Print "Waveform X display range: " + _
    Format(dblXDispRange, "Scientific")

Debug.Print "Waveform X display origin: " + _
    Format(dblXDispOrigin, "Scientific")

Debug.Print "Waveform Y display range: " + _
    Format(dblYDispRange, "Scientific")

Debug.Print "Waveform Y display origin: " + _
    Format(dblYDispOrigin, "Scientific")

Debug.Print "Date: " + strDate
Debug.Print "Time: " + strTime
Debug.Print "Frame model: " + strFrameModel
Debug.Print "Acquire mode: " + dctAcqMode.Item(intAcqMode)

Debug.Print "Completion pct: " + _
    FormatNumber(intCompletion, 0)

Debug.Print "Waveform X units: " + dctUnits.Item(intXUnits)
Debug.Print "Waveform Y units: " + dctUnits.Item(intYUnits)

Debug.Print "Max BW limit: " + _
    Format(dblMaxBwLimit, "Scientific")

Debug.Print "Min BW limit: " + _
    Format(dblMinBwLimit, "Scientific")

' Get the waveform data.
DoCommand ":WAVeform:STReaming OFF"
varQueryResult = DoQueryIEEEBlock_I2(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(UBound(varQueryResult) + 1)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
Dim lngI As Long

For lngI = 0 To UBound(varQueryResult)
  lngDataValue = varQueryResult(lngI)

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
```

```
        ", " + _
        FormatNumber((lngDataValue * sngYIncrement) + sngYOrigin)

  Next lngI

  ' Close output file.
  Close hFile    ' Close file.
  MsgBox "Waveform format WORD data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

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

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo VisaComError

  myScope.WriteString command
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteIEEEBlock command, data
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Function DoQueryString(query As String) As String

  On Error GoTo VisaComError
```

```
  myScope.WriteString query
  DoQueryString = myScope.ReadString
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumber(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryNumber = myScope.ReadNumber
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteString query
  DoQueryNumbers = myScope.ReadList
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

  On Error GoTo VisaComError
```

```
  myScope.WriteString query
  DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryIEEEBlock_I2(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryIEEEBlock_I2 = myScope.ReadIEEEBlock(BinaryType_I2)
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo VisaComError

  Dim strErrVal As String
  Dim strOut As String

  myScope.WriteString ":SYSTem:ERRor? STRing"  ' 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? STRing" ' Request error message.
    strErrVal = myScope.ReadString         ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    myScope.FlushWrite (False)
    myScope.FlushRead

  End If

  Exit Sub

VisaComError:
```

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

End Sub
```

VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 3.0 Type Library:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 3.0 Type Library**; then click **OK**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA COM Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent Infiniium Series oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace Infiniium
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
```

```
      VisaComInstrument("TCPIP0::130.29.71.191::inst0::INSTR");
    myScope.SetTimeoutSeconds(10);

    // Initialize - start from a known state.
    Initialize();

    // Capture data.
    Capture();

    // Analyze the captured waveform.
    Analyze();

    Console.WriteLine("Press any key to exit");
    Console.ReadKey();
  }
  catch (System.ApplicationException err)
  {
    Console.WriteLine("*** VISA COM Error : " + err.Message);
  }
  catch (System.SystemException err)
  {
    Console.WriteLine("*** System Error Message : " + err.Message);
  }
  catch (System.Exception err)
  {
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
  }
  finally
  {
    myScope.Close();
  }
}

/*
 * Initialize the oscilloscope to a known state.
 * -------------------------------------------------------------
 */
private static void Initialize()
{
  string strResults;

  // Clear status.
  myScope.DoCommand("*CLS");

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Load the default setup.
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * -------------------------------------------------------------
 */
```

```
private static void Capture()
{
  // Set probe attenuation factor.
  myScope.DoCommand(":CHANnel1:PROBe 1.0");
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      myScope.DoQueryString(":CHANnel1:PROBe?"));

  // Use auto-scale to automatically set up oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

  myScope.DoCommand(":TRIGger:LEVel CHANnel1,-2E-3");
  Console.WriteLine("Trigger level, channel 1: {0}",
      myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"));

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

  // Save oscilloscope setup.
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Query and read setup string.
  ResultsArray = myScope.DoQueryIEEEBlock_UI1(":SYSTem:SETup?");
  nLength = ResultsArray.Length;

  // Write setup string to file.
  strPath = "c:\\scope\\config\\setup.stp";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(ResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Setup bytes saved: {0}", nLength);

  // Change settings with individual commands:

  // Set vertical scale and offset.
  myScope.DoCommand(":CHANnel1:SCALe 0.1");
  Console.WriteLine("Channel 1 vertical scale: {0}",
      myScope.DoQueryString(":CHANnel1:SCALe?"));

  myScope.DoCommand(":CHANnel1:OFFSet 0.0");
  Console.WriteLine("Channel 1 vertical offset: {0}",
      myScope.DoQueryString(":CHANnel1:OFFSet?"));

  // Set horizontal scale and offset.
  myScope.DoCommand(":TIMebase:SCALe 0.0002");
```

```
  Console.WriteLine("Timebase scale: {0}",
      myScope.DoQueryString(":TIMebase:SCALe?"));

  myScope.DoCommand(":TIMebase:POSition 0.0");
  Console.WriteLine("Timebase position: {0}",
      myScope.DoQueryString(":TIMebase:POSition?"));

  // Set the acquisition mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe");
  Console.WriteLine("Acquire mode: {0}",
      myScope.DoQueryString(":ACQuire:MODE?"));

  // Or, configure by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);
  nBytesWritten = DataArray.Length;

  // Restore setup string.
  myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Set the desired number of waveform points,
  // and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000");
  myScope.DoCommand(":DIGitize");
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make measurements.
  // ----------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"));

  double fResult;
  myScope.DoCommand(":MEASure:FREQuency");
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  myScope.DoCommand(":MEASure:VAMPlitude");
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);

  // Download the screen image.
```

```
// ------------------------------------------------------------

// Get the screen data.
ResultsArray =
    myScope.DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG");
nLength = ResultsArray.Length;

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Get the waveform points mode.
Console.WriteLine("Waveform type: {0}",
    myScope.DoQueryString(":WAVeform:TYPE?"));

// Get the number of waveform points.
Console.WriteLine("Waveform points: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned:
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
```

```
  new Dictionary<string, string>()
{
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

strPreamble = myScope.DoQueryString(":WAVeform:PREamble?");
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
Console.WriteLine("Date: {0}", strsPreamble[15]);
Console.WriteLine("Time: {0}", strsPreamble[16]);
Console.WriteLine("Frame model: {0}", strsPreamble[17]);
Console.WriteLine("Acquire mode: {0}",
```

```
            dctAcqMode[strsPreamble[18]]);
        Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
        Console.WriteLine("Waveform X inits: {0}",
            dctUnits[strsPreamble[20]]);
        Console.WriteLine("Waveform Y units: {0}",
            dctUnits[strsPreamble[21]]);
        Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
        Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

        // Get numeric values for later calculations.
        double fXincrement;
        fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?");
        double fXorigin;
        fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?");
        double fYincrement;
        fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?");
        double fYorigin;
        fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?");

        // Get the waveform data.
        myScope.DoCommand(":WAVeform:STReaming OFF");
        short[] WordDataArray;   // Results array.
        WordDataArray = myScope.DoQueryIEEEBlock_I2(":WAVeform:DATA?");
        nLength = WordDataArray.Length;
        Console.WriteLine("Number of data values: {0}", nLength);

        // Set up output file:
        strPath = "c:\\scope\\data\\waveform_data.csv";
        if (File.Exists(strPath)) File.Delete(strPath);

        // Open file for output.
        StreamWriter writer = File.CreateText(strPath);

        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i++)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)i * fXincrement),
              (((float)WordDataArray[i])
              * fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath);
      }
    }

    class VisaComInstrument
    {
      private ResourceManagerClass m_ResourceManager;
      private FormattedIO488Class m_IoObject;
      private string m_strVisaAddress;

      // Constructor.
      public VisaComInstrument(string strVisaAddress)
      {
        // Save VISA address in member variable.
```

```
  m_strVisaAddress = strVisaAddress;

  // Open the default VISA COM IO object.
  OpenIo();

  // Clear the interface.
  m_IoObject.IO.Clear();
}

public void DoCommand(string strCommand)
{
  // Send the command.
  m_IoObject.WriteString(strCommand, true);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);
}

public void DoCommandIEEEBlock(string strCommand,
    byte[] DataArray)
{
  // Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result string.
  string strResults;
  strResults = m_IoObject.ReadString();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results string.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result number.
  double fResult;
  fResult = (double)m_IoObject.ReadNumber(
    IEEEASCIIType.ASCIIType_R8, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);
```

```
  // Return result number.
  return fResult;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result numbers.
  double[] fResultsArray;
  fResultsArray = (double[])m_IoObject.ReadList(
    IEEEASCIIType.ASCIIType_R8, ",;");

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return result numbers.
  return fResultsArray;
}

public byte[] DoQueryIEEEBlock_UI1(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  byte[] ResultsArray;
  ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_UI1, false, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}

public short[] DoQueryIEEEBlock_I2(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  short[] ResultsArray;
  ResultsArray = (short[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_I2, false, true);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}
```

```
private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  string strInstrumentError;
  bool bFirstError = true;

  do   // While not "0,No error".
  {
    m_IoObject.WriteString(":SYSTem:ERRor? STRing", true);
    strInstrumentError = m_IoObject.ReadString();

    if (!strInstrumentError.ToString().StartsWith("0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("0,"));
}

private void OpenIo()
{
  m_ResourceManager = new ResourceManagerClass();
  m_IoObject = new FormattedIO488Class();

  // Open the default VISA COM IO object.
  try
  {
    m_IoObject.IO =
      (IMessage)m_ResourceManager.Open(m_strVisaAddress,
      AccessMode.NO_LOCK, 0, "");
  }
  catch (Exception e)
  {
    Console.WriteLine("An error occurred: {0}", e.Message);
  }
}

public void SetTimeoutSeconds(int nSeconds)
{
  m_IoObject.IO.Timeout = nSeconds * 1000;
}

public void Close()
{
  try
  {
    m_IoObject.IO.Close();
  }
  catch { }

  try
  {
```

```
            Marshal.ReleaseComObject(m_IoObject);
          }
          catch { }

          try
          {
            Marshal.ReleaseComObject(m_ResourceManager);
          }
          catch { }
        }
      }
    }
```

VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 3.0 Type Library:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 3.0 Type Library**; then click **OK**.
   e. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "Infiniium.VisaComInstrumentApp" as the **Startup object**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA COM Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Agilent Infiniium Series oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports System.Collections.Generic
Imports Ivi.Visa.Interop
```

```
Imports System.Runtime.InteropServices

Namespace Infiniium
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New  _
          VisaComInstrument("TCPIP0::130.29.71.191::inst0::INSTR")
        myScope.SetTimeoutSeconds(10)

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

      Catch err As System.ApplicationException
        Console.WriteLine("*** VISA Error Message : " + err.Message)
      Catch err As System.SystemException
        Console.WriteLine("*** System Error Message : " + err.Message)
      Catch err As System.Exception
        System.Diagnostics.Debug.Fail("Unexpected Error")
        Console.WriteLine("*** Unexpected Error : " + err.Message)
      Finally
        myScope.Close()
      End Try
    End Sub

    ' Initialize the oscilloscope to a known state.
    ' ---------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As String

      ' Clear status.
      myScope.DoCommand("*CLS")

      ' Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?")
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Load the default setup.
      myScope.DoCommand("*RST")

    End Sub

    ' Capture the waveform.
    ' ---------------------------------------------------------------

    Private Shared Sub Capture()

      ' Set probe attenuation factor.
```

```
myScope.DoCommand(":CHANnel1:PROBe 1.0")
Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
    myScope.DoQueryString(":CHANnel1:PROBe?"))

' Use auto-scale to automatically configure oscilloscope.
myScope.DoCommand(":AUToscale")

' Set trigger mode.
myScope.DoCommand(":TRIGger:MODE EDGE")
Console.WriteLine("Trigger mode: {0}", _
    myScope.DoQueryString(":TRIGger:MODE?"))

' Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
Console.WriteLine("Trigger edge source: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

myScope.DoCommand(":TRIGger:LEVel CHANnel1,-2E-3")
Console.WriteLine("Trigger level, channel 1: {0}", _
    myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()    ' Results array.
Dim nLength As Integer   ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock_UI1(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.1")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet 0.0")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:SCALe 0.0002")
Console.WriteLine("Timebase scale: {0}", _
    myScope.DoQueryString(":TIMebase:SCALe?"))
```

```
  myScope.DoCommand(":TIMebase:POSition 0.0")
  Console.WriteLine("Timebase position: {0}", _
      myScope.DoQueryString(":TIMebase:POSition?"))

  ' Set the acquisition mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe")
  Console.WriteLine("Acquire mode: {0}", _
      myScope.DoQueryString(":ACQuire:MODE?"))

  ' Or, configure by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.stp"
  DataArray = File.ReadAllBytes(strPath)
  nBytesWritten = DataArray.Length

  ' Restore setup string.
  myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000")
  myScope.DoCommand(":DIGitize")

End Sub

' Analyze the captured waveform.
' ------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()    ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make measurements.
  ' ------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}", _
      myScope.DoQueryString(":MEASure:SOURce?"))

  myScope.DoCommand(":MEASure:FREQuency")
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  myScope.DoCommand(":MEASure:VAMPlitude")
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

  ' Download the screen image.
  ' ------------------------------------------------------------
```

```
' Get the screen data.
ResultsArray = myScope.DoQueryIEEEBlock_UI1(":DISPlay:DATA? PNG")
nLength = ResultsArray.Length

' Store the screen data to a file.
strPath = "c:\scope\data\screen.png"
Dim fStream As FileStream
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' ------------------------------------------------------------

' Get the waveform type.
Console.WriteLine("Waveform type: {0}", _
    myScope.DoQueryString(":WAVeform:TYPE?"))

' Get the number of waveform points.
Console.WriteLine("Waveform points: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned:
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")

Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
```

```
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

strPreamble = myScope.DoQueryString(":WAVeform:PREamble?")
strsPreamble = strPreamble.Split(",")c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))

Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
Console.WriteLine("Waveform Y display range: {0}", _
    strsPreamble(13))
Console.WriteLine("Waveform Y display origin: {0}", _
    strsPreamble(14))
Console.WriteLine("Date: {0}", strsPreamble(15))
Console.WriteLine("Time: {0}", strsPreamble(16))
Console.WriteLine("Frame model: {0}", strsPreamble(17))
Console.WriteLine("Acquire mode: {0}", _
    dctAcqMode(strsPreamble(18)))
Console.WriteLine("Completion pct: {0}", strsPreamble(19))
Console.WriteLine("Waveform X inits: {0}", _
    dctUnits(strsPreamble(20)))
Console.WriteLine("Waveform Y units: {0}", _
    dctUnits(strsPreamble(21)))
Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

' Get numeric values for later calculations.
Dim fXincrement As Double
fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?")
```

```
    Dim fXorigin As Double
    fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?")
    Dim fYincrement As Double
    fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?")
    Dim fYorigin As Double
    fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?")

    ' Get the waveform data.
    myScope.DoCommand(":WAVeform:STReaming OFF")
    Dim WordDataArray As Short()
    WordDataArray = myScope.DoQueryIEEEBlock_I2(":WAVeform:DATA?")
    nLength = WordDataArray.Length
    Console.WriteLine("Number of data values: {0}", nLength)

    ' Set up output file:
    strPath = "c:\scope\data\waveform_data.csv"
    If File.Exists(strPath) Then
      File.Delete(strPath)
    End If

    ' Open file for output.
    Dim writer As StreamWriter = File.CreateText(strPath)

    ' Output waveform data in CSV format.
    For index As Integer = 0 To nLength - 1
      ' Write time value, voltage value.
      writer.WriteLine("{0:f9}, {1:f6}", _
          fXorigin + (CSng(index) * fXincrement), _
          (CSng(WordDataArray(index)) * fYincrement) + fYorigin)
    Next

    ' Close output file.
    writer.Close()
    Console.WriteLine("Waveform format WORD data written to {0}", _
        strPath)

  End Sub

End Class

Class VisaComInstrument
  Private m_ResourceManager As ResourceManagerClass
  Private m_IoObject As FormattedIO488Class
  Private m_strVisaAddress As String

  ' Constructor.
  Public Sub New(ByVal strVisaAddress As String)

    ' Save VISA address in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA COM IO object.
    OpenIo()

    ' Clear the interface.
    m_IoObject.IO.Clear()
```

```
End Sub

Public Sub DoCommand(ByVal strCommand As String)

  ' Send the command.
  m_IoObject.WriteString(strCommand, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte())

  ' Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result string.
  Dim strResults As String
  strResults = m_IoObject.ReadString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results string.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result number.
  Dim fResult As Double
  fResult = _
      CDbl(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return result number.
  Return fResult
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) As _
    Double()
  ' Send the query.
```

```
  m_IoObject.WriteString(strQuery, True)

  ' Get the result numbers.
  Dim fResultsArray As Double()
  fResultsArray = _
      m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return result numbers.
  Return fResultsArray
End Function

Public _
    Function _
        DoQueryIEEEBlock_UI1(ByVal strQuery As String) As Byte()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the results array.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim ResultsArray As Byte()
  ResultsArray = _
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
      False, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results array.
  Return ResultsArray
End Function

Public _
    Function _
        DoQueryIEEEBlock_I2(ByVal strQuery As String) As Short()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the results array.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim ResultsArray As Short()
  ResultsArray = _
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_I2, _
      False, True)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return results array.
  Return ResultsArray
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As String
```

```
        Dim bFirstError As Boolean = True
        Do   ' While not "0,No error".
          m_IoObject.WriteString(":SYSTem:ERRor? STRing", True)
          strInstrumentError = m_IoObject.ReadString()

          If Not strInstrumentError.ToString().StartsWith("0,") Then
            If bFirstError Then
              Console.WriteLine("ERROR(s) for command '{0}': ", _
                  strCommand)
              bFirstError = False
            End If
            Console.Write(strInstrumentError)
          End If
        Loop While Not strInstrumentError.ToString().StartsWith("0,")
      End Sub

      Private Sub OpenIo()
        m_ResourceManager = New ResourceManagerClass()
        m_IoObject = New FormattedIO488Class()

        ' Open the default VISA COM IO object.
        Try
          m_IoObject.IO = _
              DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
                  AccessMode.NO_LOCK, 0, ""), IMessage)
        Catch e As Exception
          Console.WriteLine("An error occurred: {0}", e.Message)
        End Try
      End Sub

      Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
        m_IoObject.IO.Timeout = nSeconds * 1000
      End Sub

      Public Sub Close()
        Try
          m_IoObject.IO.Close()
        Catch
        End Try

        Try
          Marshal.ReleaseComObject(m_IoObject)
        Catch
        End Try

        Try
          Marshal.ReleaseComObject(m_ResourceManager)
        Catch
        End Try
      End Sub
    End Class
End Namespace
```

VISA COM Example in Python

You can use the Python programming language with the "comtypes" package to control Agilent oscilloscopes.

The Python language and "comtypes" package can be downloaded from the web at "http://www.python.org/" and "http://starship.python.net/crew/theller/comtypes/", respectively.

To run this example with Python and "comtypes":

1. Cut-and-paste the code that follows into a file named "example.py".
2. Edit the program to use the VISA address of your oscilloscope.
3. If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

   ```
   python example.py
   ```

```
#
# Agilent VISA COM Example in Python using "comtypes"
# ******************************************************
# This program illustrates a few commonly used programming
# features of your Agilent Infiniium Series oscilloscope.
# ******************************************************

# Import Python modules.
# ------------------------------------------------------------
import string
import time
import sys
import array

from comtypes.client import GetModule
from comtypes.client import CreateObject

# Run GetModule once to generate comtypes.gen.VisaComLib.
if not hasattr(sys, "frozen"):
 GetModule("C:\Program Files (x86)\IVI Foundation\VISA\VisaCom\
GlobMgr.dll")

import comtypes.gen.VisaComLib as VisaComLib


# Global variables (booleans: 0 = False, 1 = True).
# ------------------------------------------------------------


# ============================================================
# Initialize:
# ============================================================
def initialize():
 # Get and display the device's *IDN? string.
 idn_string = do_query_string("*IDN?")
 print "Identification string '%s'" % idn_string
```

```
 # Clear status and load the default setup.
 do_command("*CLS")
 do_command("*RST")


# ============================================================
# Capture:
# ============================================================
def capture():

 # Set probe attenuation factor.
 do_command(":CHANnel1:PROBe 1.0")
 qresult = do_query_string(":CHANnel1:PROBe?")
 print "Channel 1 probe attenuation factor: %s" % qresult

 # Use auto-scale to automatically set up oscilloscope.
 print "Autoscale."
 do_command(":AUToscale")

 # Set trigger mode.
 do_command(":TRIGger:MODE EDGE")
 qresult = do_query_string(":TRIGger:MODE?")
 print "Trigger mode: %s" % qresult

 # Set EDGE trigger parameters.
 do_command(":TRIGger:EDGE:SOURCe CHANnel1")
 qresult = do_query_string(":TRIGger:EDGE:SOURce?")
 print "Trigger edge source: %s" % qresult

 do_command(":TRIGger:LEVel CHANnel1,-2E-3")
 qresult = do_query_string(":TRIGger:LEVel? CHANnel1")
 print "Trigger level, channel 1: %s" % qresult

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 setup_bytes = do_query_ieee_block_UI1(":SYSTem:SETup?")
 nLength = len(setup_bytes)
 f = open("setup.stp", "wb")
 f.write(bytearray(setup_bytes))
 f.close()
 print "Setup bytes saved: %d" % nLength

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(":CHANnel1:SCALe 0.1")
 qresult = do_query_number(":CHANnel1:SCALe?")
 print "Channel 1 vertical scale: %f" % qresult

 do_command(":CHANnel1:OFFSet 0.0")
 qresult = do_query_number(":CHANnel1:OFFSet?")
 print "Channel 1 offset: %f" % qresult
```

```
 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 200e-6")
 qresult = do_query_string(":TIMebase:SCALe?")
 print "Timebase scale: %s" % qresult

 do_command(":TIMebase:POSition 0.0")
 qresult = do_query_string(":TIMebase:POSition?")
 print "Timebase position: %s" % qresult

 # Set the acquisition mode.
 do_command(":ACQuire:MODE RTIMe")
 qresult = do_query_string(":ACQuire:MODE?")
 print "Acquire mode: %s" % qresult

 # Or, configure by loading a previously saved setup.
 f = open("setup.stp", "rb")
 setup_bytes = f.read()
 f.close()
 do_command_ieee_block(":SYSTem:SETup", array.array('B', setup_bytes))
 print "Setup bytes restored: %d" % len(setup_bytes)

 # Set the desired number of waveform points,
 # and capture an acquisition.
 do_command(":ACQuire:POINts 32000")
 do_command(":DIGitize")


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
 do_command(":MEASure:SOURce CHANnel1")
 qresult = do_query_string(":MEASure:SOURce?")
 print "Measure source: %s" % qresult

 do_command(":MEASure:FREQuency")
 qresult = do_query_string(":MEASure:FREQuency?")
 print "Measured frequency on channel 1: %s" % qresult

 do_command(":MEASure:VAMPlitude")
 qresult = do_query_string(":MEASure:VAMPlitude?")
 print "Measured vertical amplitude on channel 1: %s" % qresult

 # Download the screen image.
 # ------------------------------------------------------------
 image_bytes = do_query_ieee_block_UI1(":DISPlay:DATA? PNG")
 nLength = len(image_bytes)
 f = open("screen_image.png", "wb")
 f.write(bytearray(image_bytes))
 f.close()
 print "Screen image written to 'screen_image.png'."

 # Download waveform data.
 # ------------------------------------------------------------
```

```
# Get the waveform type.
qresult = do_query_string(":WAVeform:TYPE?")
print "Waveform type: %s" % qresult

# Get the number of waveform points.
qresult = do_query_string(":WAVeform:POINts?")
print "Waveform points: %s" % qresult

# Set the waveform source.
do_command(":WAVeform:SOURce CHANnel1")
qresult = do_query_string(":WAVeform:SOURce?")
print "Waveform source: %s" % qresult

# Choose the format of the data returned:
do_command(":WAVeform:FORMat WORD")
print "Waveform format: %s" % do_query_string(":WAVeform:FORMat?")

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
 5 : "DECIBEL",
}

preamble_string = do_query_string(":WAVeform:PREamble?")
(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
```

```
 x_reference, y_increment, y_origin, y_reference, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = string.split(preamble_string, ",")

print "Waveform format: %s" % wav_form_dict[int(wav_form)]
print "Acquire type: %s" % acq_type_dict[int(acq_type)]
print "Waveform points desired: %s" % wfmpts
print "Waveform average count: %s" % avgcnt
print "Waveform X increment: %s" % x_increment
print "Waveform X origin: %s" % x_origin
print "Waveform X reference: %s" % x_reference   # Always 0.
print "Waveform Y increment: %s" % y_increment
print "Waveform Y origin: %s" % y_origin
print "Waveform Y reference: %s" % y_reference   # Always 0.
print "Coupling: %s" % coupling_dict[int(coupling)]
print "Waveform X display range: %s" % x_display_range
print "Waveform X display origin: %s" % x_display_origin
print "Waveform Y display range: %s" % y_display_range
print "Waveform Y display origin: %s" % y_display_origin
print "Date: %s" % date
print "Time: %s" % time
print "Frame model #: %s" % frame_model
print "Acquire mode: %s" % acq_mode_dict[int(acq_mode)]
print "Completion pct: %s" % completion
print "Waveform X units: %s" % units_dict[int(x_units)]
print "Waveform Y units: %s" % units_dict[int(y_units)]
print "Max BW limit: %s" % max_bw_limit
print "Min BW limit: %s" % min_bw_limit

# Get numeric values for later calculations.
x_increment = do_query_number(":WAVeform:XINCrement?")
x_origin = do_query_number(":WAVeform:XORigin?")
y_increment = do_query_number(":WAVeform:YINCrement?")
y_origin = do_query_number(":WAVeform:YORigin?")

# Get the waveform data.
do_command(":WAVeform:STReaming OFF")
data_words = do_query_ieee_block_I2(":WAVeform:DATA?")
nLength = len(data_words)
print "Number of data values: %d" % nLength

# Open file for output.
strPath = "waveform_data.csv"
f = open(strPath, "w")

# Output waveform data in CSV format.
for i in xrange(0, nLength - 1):
 time_val = x_origin + (i * x_increment)
 voltage = (data_words[i] * y_increment) + y_origin
 f.write("%E, %f\n" % (time_val, voltage))

# Close output file.
f.close()
print "Waveform format WORD data written to %s." % strPath
```

```
# ==========================================================
# Send a command and check for errors:
# ==========================================================
def do_command(command):
 myScope.WriteString("%s" % command, True)
 check_instrument_errors(command)


# ==========================================================
# Send a command and check for errors:
# ==========================================================
def do_command_ieee_block(command, data):
 myScope.WriteIEEEBlock(command, data, True)
 check_instrument_errors(command)


# ==========================================================
# Send a query, check for errors, return string:
# ==========================================================
def do_query_string(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadString()
 check_instrument_errors(query)
 return result


# ==========================================================
# Send a query, check for errors, return string:
# ==========================================================
def do_query_ieee_block_UI1(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_UI1, \
   False, True)
 check_instrument_errors(query)
 return result


# ==========================================================
# Send a query, check for errors, return string:
# ==========================================================
def do_query_ieee_block_I2(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadIEEEBlock(VisaComLib.BinaryType_I2, \
   False, True)
 check_instrument_errors(query)
 return result


# ==========================================================
# Send a query, check for errors, return values:
# ==========================================================
def do_query_number(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadNumber(VisaComLib.ASCIIType_R8, True)
 check_instrument_errors(query)
 return result
```

```
# ==========================================================
# Send a query, check for errors, return values:
# ==========================================================
def do_query_numbers(query):
 myScope.WriteString("%s" % query, True)
 result = myScope.ReadList(VisaComLib.ASCIIType_R8, ",;")
 check_instrument_errors(query)
 return result


# ==========================================================
# Check for instrument errors:
# ==========================================================
def check_instrument_errors(command):

 while True:
  myScope.WriteString(":SYSTem:ERRor? STRing", True)
  error_string = myScope.ReadString()
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".
    print "ERROR: %s, command: '%s'" % (error_string, command)
    print "Exited because of error."
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print "ERROR: :SYSTem:ERRor? STRing returned nothing, command: '%s'"
\
     % command
   print "Exited because of error."
   sys.exit(1)


# ==========================================================
# Main program:
# ==========================================================
rm = CreateObject("VISA.GlobalRM", \
 interface=VisaComLib.IResourceManager)
myScope = CreateObject("VISA.BasicFormattedIO", \
 interface=VisaComLib.IFormattedIO488)
myScope.IO = \
 rm.Open("TCPIP0::lab-qrex-lp2-10.cos.agilent.com::inst0::INSTR")

# Clear the interface.
myScope.IO.Clear
print "Interface cleared."

# Set the Timeout to 15 seconds.
myScope.IO.Timeout = 15000   # 15 seconds.
print "Timeout set to 15000 milliseconds."

# Initialize the oscilloscope, capture data, and analyze.
```

```
initialize()
capture()
analyze()

print "End of program"
```

VISA Examples

- "VISA Example in C" on page 1111
- "VISA Example in Visual Basic" on page 1120
- "VISA Example in C#" on page 1130
- "VISA Example in Visual Basic .NET" on page 1142
- "VISA Example in Python" on page 1154

VISA Example in C

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
6. Edit the program to use the VISA address of your oscilloscope.
7. Choose **Project > Properties...**. In the Property Pages dialog, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Click **OK** to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, select **VC++ Directories** under Projects and Solutions.
   c. Show directories for **Include files**, and add the include directory (for example, Program Files\IVI Foundation\VISA\WinNT\include).
   d. Show directories for **Library files**, and add the library files directory (for example, Program Files\IVI Foundation\VISA\WinNT\lib\msc).
   e. Click **OK** to close the Options dialog.
9. Build and run the program.

```
/*
 * Agilent VISA Example in C
```

```
 * -------------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Agilent Infiniium Series oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <visa.h>             /* Agilent VISA routines. */

#define VISA_ADDRESS "TCPIP0::130.29.71.191::inst0::INSTR"
#define IEEEBLOCK_SPACE 5000000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);     /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE byte block. */
int do_query_ieeeblock_words(char *query);   /* Query for word block. */
void check_instrument_errors();        /* Check for inst errors. */
void error_handler();                  /* VISA error handler. */

/* Global variables */
ViSession defaultRM, vi;        /* Device session ID. */
ViStatus err;                   /* VISA function return value. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
signed short ieeeblock_data_words[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock_words(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------------- */
void main(void)
{
  /* Open the default resource manager session. */
  err = viOpenDefaultRM(&defaultRM);
  if (err != VI_SUCCESS) error_handler();

  /* Open the session using the oscilloscope's VISA address. */
  err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL, VI_NULL, &vi);
  if (err != VI_SUCCESS) error_handler();

  /* Set the I/O timeout to fifteen seconds. */
  err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 15000);
  if (err != VI_SUCCESS) error_handler();

  /* Clear the interface. */
  err = viClear(vi);
```

```
  if (err != VI_SUCCESS) error_handler();

  /* Initialize - start from a known state. */
  initialize();

  /* Capture data. */
  capture();

  /* Analyze the captured waveform. */
  analyze();

  /* Close the vi session and the resource manager session. */
  viClose(vi);
  viClose(defaultRM);
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize (void)
{
  /* Clear status. */
  do_command("*CLS");

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Load the default setup. */
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture (void)
{
  int num_values;
  FILE *fp;

  /* Set probe attenuation factor. */
  do_command(":CHANnel1:PROBe 1.0");
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope. */
  do_command(":AUToscale");

  /* Set trigger mode. */
  do_command(":TRIGger:MODE EDGE");
  do_query_string(":TRIGger:MODE?");
  printf("Trigger mode: %s\n", str_result);

  /* Set EDGE trigger parameters. */
  do_command(":TRIGger:EDGE:SOURCe CHANnel1");
  do_query_string(":TRIGger:EDGE:SOURce?");
  printf("Trigger edge source: %s\n", str_result);

  do_command(":TRIGger:LEVel CHANnel1,-2E-3");
```

```
do_query_string(":TRIGger:LEVel? CHANnel1");
printf("Trigger level, channel 1: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope setup. */

/* Read system setup. */
num_values = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_values);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.1");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet 0.0");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and offset. */
do_command(":TIMebase:SCALe 0.0002");
do_query_string(":TIMebase:SCALe?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMebase:POSition 0.0");
do_query_string(":TIMebase:POSition?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition mode. */
do_command(":ACQuire:MODE RTIMe");
do_query_string(":ACQuire:MODE?");
printf("Acquire mode: %s\n", str_result);

/* Or, set up by loading a previously saved setup. */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_values = fread (ieeeblock_data, sizeof(unsigned char),
  IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");

/* Restore setup string. */
```

```
  num_values = do_command_ieeeblock(":SYSTem:SETup", num_values);
  printf("Restored setup string (%d bytes).\n", num_values);

  /* Set the desired number of waveform points,
   * and capture an acquisition. */
  do_command(":ACQuire:POINts 32000");
  do_command(":DIGitize");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------------ */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double y_increment;
  double y_origin;

  FILE *fp;
  int num_values;   /* Number of bytes returned from instrument. */
  int i;

  /* Make measurements.
   * ------------------------------------------------------------------ */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
  printf("Frequency: %.4f kHz\n", num_result / 1000);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  /* Download the screen image.
   * ------------------------------------------------------------------ */

  /* Read screen image. */
  num_values = do_query_ieeeblock(":DISPlay:DATA? PNG");
  printf("Screen image bytes: %d\n", num_values);

  /* Write screen image bytes to file. */
  fp = fopen ("c:\\scope\\data\\screen.png", "wb");
  num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
    fp);
  fclose (fp);
  printf("Wrote screen image (%d bytes) to ", num_values);
  printf("c:\\scope\\data\\screen.bmp.\n");

  /* Download waveform data.
   * ------------------------------------------------------------------ */
```

```
  /* Get the waveform type. */
  do_query_string(":WAVeform:TYPE?");
  printf("Waveform type: %s\n", str_result);

  /* Get the number of waveform points. */
  do_query_string(":WAVeform:POINts?");
  printf("Waveform points: %s\n", str_result);

  /* Set the waveform source. */
  do_command(":WAVeform:SOURce CHANnel1");
  do_query_string(":WAVeform:SOURce?");
  printf("Waveform source: %s\n", str_result);

  /* Choose the format of the data returned: */
  do_command(":WAVeform:FORMat WORD");
  do_query_string(":WAVeform:FORMat?");
  printf("Waveform format: %s\n", str_result);

  /* Display the waveform settings: */
  do_query_number(":WAVeform:XINCrement?");
  x_increment = num_result;
  printf("Waveform X increment: %e\n", x_increment);

  do_query_number(":WAVeform:XORigin?");
  x_origin = num_result;
  printf("Waveform X origin: %e\n", x_origin);

  do_query_number(":WAVeform:YINCrement?");
  y_increment = num_result;
  printf("Waveform Y increment: %e\n", y_increment);

  do_query_number(":WAVeform:YORigin?");
  y_origin = num_result;
  printf("Waveform Y origin: %e\n", y_origin);

  /* Read waveform data. */
  num_values = do_query_ieeeblock_words(":WAVeform:DATA?");
  printf("Number of data values: %d\n", num_values);

  /* Open file for output. */
  fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

  /* Output waveform data in CSV format. */
  for (i = 0; i < num_values - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      ((float)ieeeblock_data_words[i] * y_increment) + y_origin);
  }

  /* Close output file. */
   fclose(fp);
   printf("Waveform format WORD data written to ");
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}
```

```
/* Send a command to the instrument.
 * ---------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];

  strcpy(message, command);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ---------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  err = viPrintf(vi, message, num_bytes);
  if (err != VI_SUCCESS) error_handler();

  err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ---------------------------------------------------------------- */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%t", str_result);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}
```

```
/* Query for a number result.
 * ------------------------------------------------------------------ */
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%lf", &num_result);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------------ */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%,10lf\n", dbl_results);
  if (err != VI_SUCCESS) error_handler();

  check_instrument_errors();
}

/* Query for an IEEE definite-length byte block result.
 * ------------------------------------------------------------------ */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  data_length = IEEEBLOCK_SPACE;
  err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
  if (err != VI_SUCCESS) error_handler();

  if (data_length == IEEEBLOCK_SPACE )
  {
```

```
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Query for an IEEE definite-length word block result.
 * ------------------------------------------------------------------- */
int do_query_ieeeblock_words(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  data_length = IEEEBLOCK_SPACE;
  err = viScanf(vi, "%#hb\n", &data_length, ieeeblock_data_words);
  if (err != VI_SUCCESS) error_handler();

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  err = viQueryf(vi, ":SYSTem:ERRor? STRing\n", "%t", str_err_val);
  if (err != VI_SUCCESS) error_handler();
  while(strncmp(str_err_val, "0,", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    err = viQueryf(vi, ":SYSTem:ERRor? STRing\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
```

```
    err = viFlush(vi, VI_READ_BUF);
    if (err != VI_SUCCESS) error_handler();
    err = viFlush(vi, VI_WRITE_BUF);
    if (err != VI_SUCCESS) error_handler();
  }
}

/* Handle VISA errors.
 * ------------------------------------------------------------------- */
void error_handler()
{
  char err_msg[1024] = {0};

  viStatusDesc(vi, err, err_msg);
  printf("VISA Error: %s\n", err_msg);
  if (err < VI_SUCCESS)
  {
    exit(1);
  }
}
```

VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Add the visa32.bas file to your project:
   a. Choose **File>Import File...**.
   b. Navigate to the header file, visa32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include), select it, and click **Open**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the VISA address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Agilent VISA Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent Infiniium Series oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public err As Long   ' Error returned by VISA function calls.
Public drm As Long   ' Session to Default Resource Manager.
```

```
Public vi As Long    ' Session to instrument.

' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public Const WordArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public wordArray(WordArraySize) As Integer
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ----------------------------------------------------------------------

Sub Main()

  ' Open the default resource manager session.
  err = viOpenDefaultRM(drm)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Open the session using the oscilloscope's VISA address.
  err = viOpen(drm, _
      "TCPIP0::130.29.71.191::inst0::INSTR", 0, 15000, vi)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Set the I/O timeout to ten seconds.
  err = viSetAttribute(vi, VI_ATTR_TMO_VALUE, 10000)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Clear the interface.
  err = viClear(vi)
  If Not (err = VI_SUCCESS) Then HandleVISAError vi

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  err = viClose(vi)
  err = viClose(drm)
```

```
End Sub

'
' Initialize the oscilloscope to a known state.
' --------------------------------------------------------------------

Private Sub Initialize()

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

  ' Load the default setup.
  DoCommand "*RST"

End Sub

'
' Capture the waveform.
' --------------------------------------------------------------------

Private Sub Capture()

  ' Set probe attenuation factor.
  DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically configure oscilloscope.
  ' ------------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode (EDGE, PULSe, PATTern, etc., and input source.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' ------------------------------------------------------------------
  Dim lngSetupStringSize As Long
  lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
```

```
Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
If Len(Dir(strPath)) Then
  Kill strPath   ' Remove file if it exists.
End If

' Open file for output.
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Dim lngI As Long
For lngI = 0 To lngSetupStringSize - 1
  Put hFile, , byteArray(lngI)   ' Write data.
Next lngI
Close hFile   ' Close file.

' Change settings with individual commands:
' ------------------------------------------------------------------

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.1"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet 0.0"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and position.
DoCommand ":TIMebase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMebase:SCALe?")

DoCommand ":TIMebase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMebase:POSition?")

' Set the acquisition mode.
DoCommand ":ACQuire:MODE RTIMe"
Debug.Print "Acquire mode: " + _
    DoQueryString(":ACQuire:MODE?")

' Or, configure by loading a previously saved setup.
' ------------------------------------------------------------------
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile   ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile)  ' Length of file.
Get hFile, , byteArray   ' Read data.
Close hFile   ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
```

```
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' -----------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

End Sub

'
' Analyze the captured waveform.
' -----------------------------------------------------------------

Private Sub Analyze()

  ' Make a couple of measurements.
  ' -----------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"


  ' Download the screen image.
  ' -----------------------------------------------------------------

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' Save screen image to a file:
  Dim strPath As String
  strPath = "c:\scope\data\screen.png"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 0 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
  MsgBox "Screen image written to " + strPath
```

```
' Download waveform data.
' ------------------------------------------------------------------

' Get the waveform type.
Debug.Print "Waveform type: " + _
    DoQueryString(":WAVeform:TYPE?")

' Get the number of waveform points.
Debug.Print "Waveform points: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned:
DoCommand ":WAVeform:FORMat WORD"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double

dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment: " + _
    Format(dblYIncrement, "Scientific")

dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin: " + _
    FormatNumber(dblYOrigin, 0)

' Get the waveform data
DoCommand ":WAVeform:STReaming OFF"
Dim lngNumWords As Long
lngNumWords = DoQueryIEEEBlock_Words(":WAVeform:DATA?")
Debug.Print "Number of data values: " + CStr(lngNumWords)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
```

```
  For lngI = 0 To lngNumWords - 1

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber((wordArray(lngI) * dblYIncrement) + dblYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format WORD data written to " + _
      "c:\scope\data\waveform_data.csv."

End Sub

Private Sub DoCommand(command As String)

  err = viVPrintf(vi, command + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  CheckInstrumentErrors

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  retCount = lngBlockSize

  Dim strCommandAndLength As String
  strCommandAndLength = command + " %#" + _
      Format(lngBlockSize) + "b"

  err = viVPrintf(vi, strCommandAndLength + vbLf, paramsArray(1))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoCommandIEEEBlock = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryString(query As String) As String

  Dim strResult As String * 200

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strResult)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoQueryString = strResult

  CheckInstrumentErrors
```

```
End Function

Private Function DoQueryNumber(query As String) As Variant

  Dim dblResult As Double

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblResult))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoQueryNumber = dblResult

  CheckInstrumentErrors

End Function

Private Function DoQueryNumbers(query As String) As Long

  Dim dblResult As Double

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(dblArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = DblArraySize

  ' Read numbers.
  err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of values returned by query.
  DoQueryNumbers = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(byteArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = ByteArraySize
```

```
  ' Get unsigned integer bytes.
  err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_READ_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_WRITE_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of bytes returned by query.
  DoQueryIEEEBlock_Bytes = retCount

  CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Words(query As String) As Long

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(wordArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = WordArraySize

  ' Get signed integer words.
  err = viVScanf(vi, "%#hb" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_READ_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_WRITE_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of bytes returned by query.
  DoQueryIEEEBlock_Words = retCount

  CheckInstrumentErrors

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String

  err = viVPrintf(vi, ":SYSTem:ERRor? STRing" + vbLf, 0) ' Query any err
ors.
```

```
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strErrVal)    ' Read: Errnum,"Error String".
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal

    err = viVPrintf(vi, ":SYSTem:ERRor? STRing" + vbLf, 0) ' Request err
or.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strErrVal)    ' Read error message.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"

    err = viFlush(vi, VI_READ_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_WRITE_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub HandleVISAError(session As Long)

  Dim strVisaErr As String * 200
  Call viStatusDesc(session, err, strVisaErr)
  MsgBox "*** VISA Error : " + strVisaErr, vbExclamation

  ' If the error is not a warning, close the session.
  If err < VI_SUCCESS Then
    If session <> 0 Then Call viClose(session)
    End
  End If

End Sub
```

VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Agilent's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Click **Add** and then click **Add Existing Item...**
   c. Navigate to the header file, visa32.cs (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include directory), select it, but *do not click the Open button*.
   d. Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

      You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent Infiniium Series oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;

namespace Infiniium
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
```

```
  {
    myScope = new
      VisaInstrument("TCPIP0::130.29.71.191::inst0::INSTR");
    myScope.SetTimeoutSeconds(10);

    // Initialize - start from a known state.
    Initialize();

    // Capture data.
    Capture();

    // Analyze the captured waveform.
    Analyze();

  }
  catch (System.ApplicationException err)
  {
    Console.WriteLine("*** VISA Error Message : " + err.Message);
  }
  catch (System.SystemException err)
  {
    Console.WriteLine("*** System Error Message : " + err.Message);
  }
  catch (System.Exception err)
  {
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
  }
  finally
  {
    myScope.Close();
  }
}

/*
 * Initialize the oscilloscope to a known state.
 * ---------------------------------------------------------------
 */
private static void Initialize()
{
  StringBuilder strResults;

  // Clear status.
  myScope.DoCommand("*CLS");

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Load the default setup.
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------
 */
```

```
private static void Capture()
{
  // Set probe attenuation factor.
  myScope.DoCommand(":CHANnel1:PROBe 1.0");
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      myScope.DoQueryString(":CHANnel1:PROBe?"));

  // Use auto-scale to automatically set up oscilloscope.
  myScope.DoCommand(":AUToscale");

  // Set trigger mode.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
  Console.WriteLine("Trigger edge source: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

  myScope.DoCommand(":TRIGger:LEVel CHANnel1,-2E-3");
  Console.WriteLine("Trigger level, channel 1: {0}",
      myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"));

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  Console.WriteLine("Trigger edge slope: {0}",
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

  // Save oscilloscope configuration.
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Query and read setup string.
  nLength = myScope.DoQueryIEEEBlock_Bytes(":SYSTem:SETup?",
    out ResultsArray);

  // Write setup string to file.
  strPath = "c:\\scope\\config\\setup.stp";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(ResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Setup bytes saved: {0}", nLength);

  // Change settings with individual commands:

  // Set vertical scale and offset.
  myScope.DoCommand(":CHANnel1:SCALe 0.1");
  Console.WriteLine("Channel 1 vertical scale: {0}",
      myScope.DoQueryString(":CHANnel1:SCALe?"));

  myScope.DoCommand(":CHANnel1:OFFSet 0.0");
  Console.WriteLine("Channel 1 vertical offset: {0}",
      myScope.DoQueryString(":CHANnel1:OFFSet?"));

  // Set horizontal scale and position.
  myScope.DoCommand(":TIMebase:SCALe 0.0002");
```

```
  Console.WriteLine("Timebase scale: {0}",
      myScope.DoQueryString(":TIMebase:SCALe?"));

  myScope.DoCommand(":TIMebase:POSition 0.0");
  Console.WriteLine("Timebase position: {0}",
      myScope.DoQueryString(":TIMebase:POSition?"));

  // Set the acquisition mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe");
  Console.WriteLine("Acquire mode: {0}",
      myScope.DoQueryString(":ACQuire:MODE?"));

  // Or, set up by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);

  // Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
    DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Set the desired number of waveform points,
  // and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000");
  myScope.DoCommand(":DIGitize");
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  short[] WordResultsArray;   // Results array for WORD data.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make measurements.
  // ------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"));

  double fResult;
  myScope.DoCommand(":MEASure:FREQuency");
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  myScope.DoCommand(":MEASure:VAMPlitude");
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
  Console.WriteLine("Vertical amplitude: {0:F2} V", fResult);
```

```
// Download the screen image.
// ------------------------------------------------------------

// Get the screen data.
nLength = myScope.DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG",
    out ResultsArray);

// Store the screen data to a file.
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Get the waveform type.
Console.WriteLine("Waveform type: {0}",
    myScope.DoQueryString(":WAVeform:TYPE?"));

// Get the number of waveform points.
Console.WriteLine("Waveform points: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned:
myScope.DoCommand(":WAVeform:FORMat WORD");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
```

```
  new Dictionary<string, string>()
{
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

strPreamble =
    myScope.DoQueryString(":WAVeform:PREamble?").ToString();
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
Console.WriteLine("Date: {0}", strsPreamble[15]);
Console.WriteLine("Time: {0}", strsPreamble[16]);
Console.WriteLine("Frame model: {0}", strsPreamble[17]);
```

```
        Console.WriteLine("Acquire mode: {0}",
            dctAcqMode[strsPreamble[18]]);
        Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
        Console.WriteLine("Waveform X inits: {0}",
            dctUnits[strsPreamble[20]]);
        Console.WriteLine("Waveform Y units: {0}",
            dctUnits[strsPreamble[21]]);
        Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
        Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

        // Get numeric values for later calculations.
        double fXincrement;
        fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?");
        double fXorigin;
        fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?");
        double fYincrement;
        fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?");
        double fYorigin;
        fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?");

        // Get the waveform data.
        myScope.DoCommand(":WAVeform:STReaming OFF");
        nLength = myScope.DoQueryIEEEBlock_Words(":WAVeform:DATA?",
            out WordResultsArray);
        Console.WriteLine("Number of data values: {0}", nLength);

        // Set up output file:
        strPath = "c:\\scope\\data\\waveform_data.csv";
        if (File.Exists(strPath)) File.Delete(strPath);

        // Open file for output.
        StreamWriter writer = File.CreateText(strPath);

        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i++)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)i * fXincrement),
              ((float)WordResultsArray[i] * fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath);
      }
    }

    class VisaInstrument
    {
      private int m_nResourceManager;
      private int m_nSession;
      private string m_strVisaAddress;

      // Constructor.
      public VisaInstrument(string strVisaAddress)
      {
        // Save VISA address in member variable.
        m_strVisaAddress = strVisaAddress;
```

```
  // Open the default VISA resource manager.
  OpenResourceManager();

  // Open a VISA resource session.
  OpenSession();

  // Clear the interface.
  int nViStatus;
  nViStatus = visa32.viClear(m_nSession);
}

public void DoCommand(string strCommand)
{
  // Send the command.
  VisaSendCommandOrQuery(strCommand);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);
}

public int DoCommandIEEEBlock(string strCommand,
  byte[] DataArray)
{
  // Send the command to the device.
  string strCommandAndLength;
  int nViStatus, nLength, nBytesWritten;

  nLength = DataArray.Length;
  strCommandAndLength = String.Format("{0} #8%08d",
    strCommand);

  // Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
    nLength);
  CheckVisaStatus(nViStatus);

  // Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
    out nBytesWritten);
  CheckVisaStatus(nViStatus);

  // Check for inst errors.
  CheckInstrumentErrors(strCommand);

  return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  StringBuilder strResults = new StringBuilder(1000);
  strResults = VisaGetResultString();
```

```
  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double fResults;
  fResults = VisaGetResultNumber();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double[] fResultsArray;
  fResultsArray = VisaGetResultNumbers();

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResultsArray;
}

public int DoQueryIEEEBlock_Bytes(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Bytes(out ResultsArray);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

public int DoQueryIEEEBlock_Words(string strQuery,
```

```
  out short[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Words(out ResultsArray);

  // Check for inst errors.
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
  // Send command or query to the device.
  string strWithNewline;
  strWithNewline = String.Format("{0}\n", strCommandOrQuery);
  int nViStatus;
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
  CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
  StringBuilder strResults = new StringBuilder(1000);

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
  CheckVisaStatus(nViStatus);

  return strResults;
}

private double VisaGetResultNumber()
{
  double fResults = 0;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
  CheckVisaStatus(nViStatus);

  return fResults;
}

private double[] VisaGetResultNumbers()
{
  double[] fResultsArray;
  fResultsArray = new double[10];

  // Read return value string from the device.
  int nViStatus;
```

```
  nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
      fResultsArray);
  CheckVisaStatus(nViStatus);

  return fResultsArray;
}

private int VisaGetResultIEEEBlock_Bytes(out byte[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new byte[5000000];
  int length;   // Number of bytes returned from instrument.

  // Set the default number of bytes that will be contained in
  // the ResultsArray to 5,000,000.
  length = 5000000;

  // 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 int VisaGetResultIEEEBlock_Words(out short[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new short[5000000];
  int length;   // Number of words returned from instrument.

  // Set the default number of words that will be contained in
  // the ResultsArray to 5,000,000.
  length = 5000000;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%#hb", ref length,
    ResultsArray);
  CheckVisaStatus(nViStatus);

  // Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
  CheckVisaStatus(nViStatus);

  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
  CheckVisaStatus(nViStatus);

  return length;
```

```
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  StringBuilder strInstrumentError = new StringBuilder(1000);
  bool bFirstError = true;

  do   // While not "0,No error"
  {
    VisaSendCommandOrQuery(":SYSTem:ERRor? STRing");
    strInstrumentError = VisaGetResultString();

    if (!strInstrumentError.ToString().StartsWith("0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("0,"));
}

private void OpenResourceManager()
{
  int nViStatus;
  nViStatus =
    visa32.viOpenDefaultRM(out this.m_nResourceManager);
  if (nViStatus < visa32.VI_SUCCESS)
    throw new
      ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
  int nViStatus;
  nViStatus = visa32.viOpen(this.m_nResourceManager,
    this.m_strVisaAddress, visa32.VI_NO_LOCK,
    visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
  CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
  int nViStatus;
  nViStatus = visa32.viSetAttribute(this.m_nSession,
    visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
  CheckVisaStatus(nViStatus);
}

public void CheckVisaStatus(int nViStatus)
{
  // If VISA error, throw exception.
  if (nViStatus < visa32.VI_SUCCESS)
```

```
        {
          StringBuilder strError = new StringBuilder(256);
          visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
            strError);
          throw new ApplicationException(strError.ToString());
        }
      }

      public void Close()
      {
        if (m_nSession != 0)
          visa32.viClose(m_nSession);
        if (m_nResourceManager != 0)
          visa32.viClose(m_nResourceManager);
      }
    }
  }
```

VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual Basic, Windows, Console Application project.
3. Cut-and-paste the code that follows into the Visual Basic .NET source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add Agilent's VISA header file to your project:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add** and then choose **Add Existing Item...**
   c. Navigate to the header file, visa32.vb (installed with Agilent IO Libraries Suite and found in the Program Files\IVI Foundation\VISA\WinNT\include directory), select it, but *do not click the Open button*.
   d. Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

      You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.
   e. Right-click the project again and choose **Properties**; then, select "Infiniium.VisaInstrumentApp" as the **Startup object**.
6. Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent Infiniium Series oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace Infiniium
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
          New VisaInstrument("TCPIP0::130.29.71.191::inst0::INSTR")
        myScope.SetTimeoutSeconds(10)

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

      Catch err As System.ApplicationException
        Console.WriteLine("*** VISA Error Message : " + err.Message)
      Catch err As System.SystemException
        Console.WriteLine("*** System Error Message : " + err.Message)
      Catch err As System.Exception
        Debug.Fail("Unexpected Error")
        Console.WriteLine("*** Unexpected Error : " + err.Message)
      Finally
        myScope.Close()
      End Try
    End Sub

    '
    ' Initialize the oscilloscope to a known state.
    ' ----------------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As StringBuilder

      ' Clear status.
      myScope.DoCommand("*CLS")

      ' Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?")
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Load the default setup.
```

```
  myScope.DoCommand("*RST")

End Sub

'
' Capture the waveform.
' --------------------------------------------------------------------

Private Shared Sub Capture()

  ' Set probe attenuation factor.
  myScope.DoCommand(":CHANnel1:PROBe 1.0")
  Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
      myScope.DoQueryString(":CHANnel1:PROBe?"))

  ' Use auto-scale to automatically set up oscilloscope.
  myScope.DoCommand(":AUToscale")

  ' Set trigger mode.
  myScope.DoCommand(":TRIGger:MODE EDGE")
  Console.WriteLine("Trigger mode: {0}", _
      myScope.DoQueryString(":TRIGger:MODE?"))

  ' Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
  Console.WriteLine("Trigger edge source: {0}", _
      myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

  myScope.DoCommand(":TRIGger:LEVel CHANnel1,-2E-3")
  Console.WriteLine("Trigger edge level: {0}", _
      myScope.DoQueryString(":TRIGger:LEVel? CHANnel1"))

  myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
  Console.WriteLine("Trigger edge slope: {0}", _
      myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

  ' Save oscilloscope setup.
  Dim ResultsArray As Byte()   ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String
  Dim fStream As FileStream

  ' Query and read setup string.
  nLength = myScope.DoQueryIEEEBlock_Bytes(":SYSTem:SETup?", _
      ResultsArray)

  ' Write setup string to file.
  strPath = "c:\scope\config\setup.stp"
  fStream = File.Open(strPath, FileMode.Create)
  fStream.Write(ResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Setup bytes saved: {0}", nLength)

  ' Change settings with individual commands:

  ' Set vertical scale and offset.
  myScope.DoCommand(":CHANnel1:SCALe 0.1")
```

```
  Console.WriteLine("Channel 1 vertical scale: {0}", _
      myScope.DoQueryString(":CHANnel1:SCALe?"))

  myScope.DoCommand(":CHANnel1:OFFSet 0.0")
  Console.WriteLine("Channel 1 vertical offset: {0}", _
      myScope.DoQueryString(":CHANnel1:OFFSet?"))

  ' Set horizontal scale and position.
  myScope.DoCommand(":TIMebase:SCALe 0.0002")
  Console.WriteLine("Timebase scale: {0}", _
      myScope.DoQueryString(":TIMebase:SCALe?"))

  myScope.DoCommand(":TIMebase:POSition 0.0")
  Console.WriteLine("Timebase position: {0}", _
      myScope.DoQueryString(":TIMebase:POSition?"))

  ' Set the acquisition mode.
  myScope.DoCommand(":ACQuire:MODE RTIMe")
  Console.WriteLine("Acquire mode: {0}", _
      myScope.DoQueryString(":ACQuire:MODE?"))

  ' Or, set up by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.stp"
  DataArray = File.ReadAllBytes(strPath)

  ' Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
      DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  myScope.DoCommand(":ACQuire:POINts 32000")
  myScope.DoCommand(":DIGitize")

End Sub

'
' Analyze the captured waveform.
' ---------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()    ' Results array.
  Dim WordResultsArray As Short()    ' Results array for WORD data.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make measurements.
  ' ---------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}", _
```

```
    myScope.DoQueryString(":MEASure:SOURce?"))

myScope.DoCommand(":MEASure:FREQuency")
fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

myScope.DoCommand(":MEASure:VAMPlitude")
fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
Console.WriteLine("Vertical amplitude: {0:F2} V", fResult)

' Download the screen image.
' -------------------------------------------------------------

' Get the screen data.
nLength = myScope.DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG", _
    ResultsArray)

' Store the screen data to a file.
strPath = "c:\scope\data\screen.png"
Dim fStream As FileStream
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' -------------------------------------------------------------

' Get the waveform type.
Console.WriteLine("Waveform type: {0}", _
    myScope.DoQueryString(":WAVeform:TYPE?"))

' Get the number of waveform points.
Console.WriteLine("Waveform points: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned:
myScope.DoCommand(":WAVeform:FORMat WORD")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")

Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
```

```
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

strPreamble = _
    myScope.DoQueryString(":WAVeform:PREamble?").ToString()
strsPreamble = strPreamble.Split(","c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))

Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
Console.WriteLine("Waveform Y display range: {0}", _
    strsPreamble(13))
Console.WriteLine("Waveform Y display origin: {0}", _
    strsPreamble(14))
Console.WriteLine("Date: {0}", strsPreamble(15))
Console.WriteLine("Time: {0}", strsPreamble(16))
```

```
    Console.WriteLine("Frame model: {0}", strsPreamble(17))
    Console.WriteLine("Acquire mode: {0}", _
        dctAcqMode(strsPreamble(18)))
    Console.WriteLine("Completion pct: {0}", strsPreamble(19))
    Console.WriteLine("Waveform X inits: {0}", _
        dctUnits(strsPreamble(20)))
    Console.WriteLine("Waveform Y units: {0}", _
        dctUnits(strsPreamble(21)))
    Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
    Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

    ' Get numeric values for later calculations.
    Dim fXincrement As Double
    fXincrement = myScope.DoQueryNumber(":WAVeform:XINCrement?")
    Dim fXorigin As Double
    fXorigin = myScope.DoQueryNumber(":WAVeform:XORigin?")
    Dim fYincrement As Double
    fYincrement = myScope.DoQueryNumber(":WAVeform:YINCrement?")
    Dim fYorigin As Double
    fYorigin = myScope.DoQueryNumber(":WAVeform:YORigin?")

    ' Get the waveform data.
    myScope.DoCommand(":WAVeform:STReaming OFF")
    nLength = myScope.DoQueryIEEEBlock_Words(":WAVeform:DATA?", _
        WordResultsArray)
    Console.WriteLine("Number of data values: {0}", nLength)

    ' Set up output file:
    strPath = "c:\scope\data\waveform_data.csv"
    If File.Exists(strPath) Then
      File.Delete(strPath)
    End If

    ' Open file for output.
    Dim writer As StreamWriter = File.CreateText(strPath)

    ' Output waveform data in CSV format.
    For index As Integer = 0 To nLength - 1
      ' Write time value, voltage value.
      writer.WriteLine("{0:f9}, {1:f6}", _
          fXorigin + (CSng(index) * fXincrement), _
          (CSng(WordResultsArray(index)) * fYincrement) + _
          fYorigin)

    Next

    ' Close output file.
    writer.Close()
    Console.WriteLine("Waveform format WORD data written to {0}", _
        strPath)

  End Sub

End Class

Class VisaInstrument
  Private m_nResourceManager As Integer
```

```
Private m_nSession As Integer
Private m_strVisaAddress As String

' Constructor.
Public Sub New(ByVal strVisaAddress As String)
  ' Save VISA address in member variable.
  m_strVisaAddress = strVisaAddress

  ' Open the default VISA resource manager.
  OpenResourceManager()

  ' Open a VISA resource session.
  OpenSession()

  ' Clear the interface.
  Dim nViStatus As Integer
  nViStatus = visa32.viClear(m_nSession)
End Sub

Public Sub DoCommand(ByVal strCommand As String)
  ' Send the command.
  VisaSendCommandOrQuery(strCommand)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

End Sub

Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte()) As Integer

  ' Send the command to the device.
  Dim strCommandAndLength As String
  Dim nViStatus As Integer
  Dim nLength As Integer
  Dim nBytesWritten As Integer

  nLength = DataArray.Length
  strCommandAndLength = [String].Format("{0} #8{1:D8}", _
      strCommand, nLength)

  ' Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
  CheckVisaStatus(nViStatus)

  ' Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength, _
      nBytesWritten)
  CheckVisaStatus(nViStatus)

  ' Check for inst errors.
  CheckInstrumentErrors(strCommand)

  Return nBytesWritten
End Function

Public Function DoQueryString(ByVal strQuery As String) _
```

```
  As StringBuilder
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim strResults As New StringBuilder(1000)
  strResults = VisaGetResultString()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResults As Double
  fResults = VisaGetResultNumber()

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResultsArray As Double()
  fResultsArray = VisaGetResultNumbers()

  ' Check for instrument errors (another command and result).
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResultsArray
End Function

Public Function DoQueryIEEEBlock_Bytes(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_Bytes(ResultsArray)

  ' Check for inst errors.
```

```
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return length
End Function

Public Function DoQueryIEEEBlock_Words(ByVal strQuery As String, _
    ByRef ResultsArray As Short()) As Integer
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim length As Integer
  ' Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock_Words(ResultsArray)

  ' Check for inst errors.
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return length
End Function

Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
     As String)
  ' Send command or query to the device.
  Dim strWithNewline As String
  strWithNewline = [String].Format("{0}" & Chr(10) & "", _
      strCommandOrQuery)
  Dim nViStatus As Integer
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
  CheckVisaStatus(nViStatus)
End Sub

Private Function VisaGetResultString() As StringBuilder
  Dim strResults As New StringBuilder(1000)

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
  CheckVisaStatus(nViStatus)

  Return strResults
End Function

Private Function VisaGetResultNumber() As Double
  Dim fResults As Double = 0

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
  CheckVisaStatus(nViStatus)

  Return fResults
End Function

Private Function VisaGetResultNumbers() As Double()
```

```
  Dim fResultsArray As Double()
  fResultsArray = New Double(9) {}

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, _
      "%,10lf" & Chr(10) & "", fResultsArray)
  CheckVisaStatus(nViStatus)

  Return fResultsArray
End Function

Private Function VisaGetResultIEEEBlock_Bytes(ByRef ResultsArray _
    As Byte()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Byte(4999999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
  ' the ResultsArray to 5,000,000.
  length = 5000000

  ' 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 Function VisaGetResultIEEEBlock_Words(ByRef ResultsArray _
    As Short()) As Integer
  ' Results array, big enough to hold a PNG.
  ResultsArray = New Short(4999999) {}
  Dim length As Integer
  ' Number of bytes returned from instrument.
  ' Set the default number of bytes that will be contained in
  ' the ResultsArray to 5,000,000.
  length = 5000000

  ' Read return value string from the device.
  Dim nViStatus As Integer
  nViStatus = visa32.viScanf(m_nSession, "%#hb", length, _
      ResultsArray)
  CheckVisaStatus(nViStatus)

  ' Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
  CheckVisaStatus(nViStatus)
```

```
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
  CheckVisaStatus(nViStatus)

  Return length
End Function

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As New StringBuilder(1000)
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error"
    VisaSendCommandOrQuery(":SYSTem:ERRor? STRing")
    strInstrumentError = VisaGetResultString()

    If Not strInstrumentError.ToString().StartsWith("0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ", _
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("0,")
End Sub

Private Sub OpenResourceManager()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
  If nViStatus < visa32.VI_SUCCESS Then
    Throw New  _
        ApplicationException("Failed to open Resource Manager")
  End If
End Sub

Private Sub OpenSession()
  Dim nViStatus As Integer
  nViStatus = visa32.viOpen(Me.m_nResourceManager, _
      Me.m_strVisaAddress, visa32.VI_NO_LOCK, _
      visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
  CheckVisaStatus(nViStatus)
End Sub

Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
  Dim nViStatus As Integer
  nViStatus = visa32.viSetAttribute(Me.m_nSession, _
      visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
  CheckVisaStatus(nViStatus)
End Sub

Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
  ' If VISA error, throw exception.
  If nViStatus < visa32.VI_SUCCESS Then
    Dim strError As New StringBuilder(256)
    visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
    Throw New ApplicationException(strError.ToString())
  End If
```

```
    End Sub

    Public Sub Close()
      If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
      End If
      If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
      End If
    End Sub
  End Class
End Namespace
```

VISA Example in Python

You can use the Python programming language with the PyVISA package to control Agilent Infiniium Series oscilloscopes.

The Python language and PyVISA package can be downloaded from the web at "http://www.python.org/" and "http://pyvisa.sourceforge.net/", respectively.

To run this example with Python and PyVISA:

1 Cut-and-paste the code that follows into a file named "example.py".

2 Edit the program to use the VISA address of your oscilloscope.

3 If "python.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
python example.py
```

```
# ********************************************************
# This program illustrates a few commonly-used programming
# features of your Agilent Infiniium Series oscilloscope.
# ********************************************************

# Import modules.
# --------------------------------------------------------
import visa
import string
import struct
import sys

# Global variables (booleans: 0 = False, 1 = True).
# --------------------------------------------------------
debug = 0


# ========================================================
# Initialize:
# ========================================================
def initialize():

 # Clear status.
```

```
 do_command("*CLS")

 # Get and display the device's *IDN? string.
 idn_string = do_query_string("*IDN?")
 print "Identification string: '%s'" % idn_string

 # Load the default setup.
 do_command("*RST")


# ======================================================
# Capture:
# ======================================================
def capture():

 # Set probe attenuation factor.
 do_command(":CHANnel1:PROBe 1.0")
 qresult = do_query_string(":CHANnel1:PROBe?")
 print "Channel 1 probe attenuation factor: %s" % qresult

 # Use auto-scale to automatically set up oscilloscope.
 print "Autoscale."
 do_command(":AUToscale")

 # Set trigger mode.
 do_command(":TRIGger:MODE EDGE")
 qresult = do_query_string(":TRIGger:MODE?")
 print "Trigger mode: %s" % qresult

 # Set EDGE trigger parameters.
 do_command(":TRIGger:EDGE:SOURCe CHANnel1")
 qresult = do_query_string(":TRIGger:EDGE:SOURce?")
 print "Trigger edge source: %s" % qresult

 do_command(":TRIGger:LEVel CHANnel1,-2E-3")
 qresult = do_query_string(":TRIGger:LEVel? CHANnel1")
 print "Trigger level, channel 1: %s" % qresult

 do_command(":TRIGger:EDGE:SLOPe POSitive")
 qresult = do_query_string(":TRIGger:EDGE:SLOPe?")
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 sSetup = do_query_string(":SYSTem:SETup?")
 sSetup = get_definite_length_block_data(sSetup)

 f = open("setup.stp", "wb")
 f.write(sSetup)
 f.close()
 print "Setup bytes saved: %d" % len(sSetup)

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 do_command(":CHANnel1:SCALe 0.1")
 qresult = do_query_values(":CHANnel1:SCALe?")[0]
 print "Channel 1 vertical scale: %f" % qresult
```

```
 do_command(":CHANnel1:OFFSet 0.0")
 qresult = do_query_values(":CHANnel1:OFFSet?")[0]
 print "Channel 1 offset: %f" % qresult

 # Set horizontal scale and offset.
 do_command(":TIMebase:SCALe 200e-6")
 qresult = do_query_string(":TIMebase:SCALe?")
 print "Timebase scale: %s" % qresult

 do_command(":TIMebase:POSition 0.0")
 qresult = do_query_string(":TIMebase:POSition?")
 print "Timebase position: %s" % qresult

 # Set the acquisition mode.
 do_command(":ACQuire:MODE RTIMe")
 qresult = do_query_string(":ACQuire:MODE?")
 print "Acquire mode: %s" % qresult

 # Or, set up oscilloscope by loading a previously saved setup.
 sSetup = ""
 f = open("setup.stp", "rb")
 sSetup = f.read()
 f.close()
 do_command(":SYSTem:SETup #8%08d%s" % (len(sSetup), sSetup), hide_param
s=True)
 print "Setup bytes restored: %d" % len(sSetup)

 # Set the desired number of waveform points,
 # and capture an acquisition.
 do_command(":ACQuire:POINts 32000")
 do_command(":DIGitize")


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
 do_command(":MEASure:SOURce CHANnel1")
 qresult = do_query_string(":MEASure:SOURce?")
 print "Measure source: %s" % qresult

 do_command(":MEASure:FREQuency")
 qresult = do_query_string(":MEASure:FREQuency?")
 print "Measured frequency on channel 1: %s" % qresult

 do_command(":MEASure:VAMPlitude")
 qresult = do_query_string(":MEASure:VAMPlitude?")
 print "Measured vertical amplitude on channel 1: %s" % qresult

 # Download the screen image.
 # ------------------------------------------------------------
 sDisplay = do_query_string(":DISPlay:DATA? PNG")
 sDisplay = get_definite_length_block_data(sDisplay)
```

```
# Save display data values to file.
f = open("screen_image.png", "wb")
f.write(sDisplay)
f.close()
print "Screen image written to screen_image.png."

# Download waveform data.
# ------------------------------------------------------------

# Get the waveform type.
qresult = do_query_string(":WAVeform:TYPE?")
print "Waveform type: %s" % qresult

# Get the number of waveform points.
qresult = do_query_string(":WAVeform:POINts?")
print "Waveform points: %s" % qresult

# Set the waveform source.
do_command(":WAVeform:SOURce CHANnel1")
qresult = do_query_string(":WAVeform:SOURce?")
print "Waveform source: %s" % qresult

# Choose the format of the data returned:
do_command(":WAVeform:FORMat BYTE")
print "Waveform format: %s" % do_query_string(":WAVeform:FORMat?")

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
```

```
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
 5 : "DECIBEL",
}

preamble_string = do_query_string(":WAVeform:PREamble?")
(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
 x_reference, y_increment, y_origin, y_reference, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = string.split(preamble_string, ",")

print "Waveform format: %s" % wav_form_dict[int(wav_form)]
print "Acquire type: %s" % acq_type_dict[int(acq_type)]
print "Waveform points desired: %s" % wfmpts
print "Waveform average count: %s" % avgcnt
print "Waveform X increment: %s" % x_increment
print "Waveform X origin: %s" % x_origin
print "Waveform X reference: %s" % x_reference   # Always 0.
print "Waveform Y increment: %s" % y_increment
print "Waveform Y origin: %s" % y_origin
print "Waveform Y reference: %s" % y_reference   # Always 0.
print "Coupling: %s" % coupling_dict[int(coupling)]
print "Waveform X display range: %s" % x_display_range
print "Waveform X display origin: %s" % x_display_origin
print "Waveform Y display range: %s" % y_display_range
print "Waveform Y display origin: %s" % y_display_origin
print "Date: %s" % date
print "Time: %s" % time
print "Frame model #: %s" % frame_model
print "Acquire mode: %s" % acq_mode_dict[int(acq_mode)]
print "Completion pct: %s" % completion
print "Waveform X units: %s" % units_dict[int(x_units)]
print "Waveform Y units: %s" % units_dict[int(y_units)]
print "Max BW limit: %s" % max_bw_limit
print "Min BW limit: %s" % min_bw_limit

# Get numeric values for later calculations.
x_increment = do_query_values(":WAVeform:XINCrement?")[0]
x_origin = do_query_values(":WAVeform:XORigin?")[0]
y_increment = do_query_values(":WAVeform:YINCrement?")[0]
y_origin = do_query_values(":WAVeform:YORigin?")[0]

# Get the waveform data.
do_command(":WAVeform:STReaming OFF")
sData = do_query_string(":WAVeform:DATA?")
sData = get_definite_length_block_data(sData)

# Unpack signed byte data.
values = struct.unpack("%db" % len(sData), sData)
print "Number of data values: %d" % len(values)

# Save waveform data values to CSV file.
f = open("waveform_data.csv", "w")
```

```
 for i in xrange(0, len(values) - 1):
  time_val = x_origin + (i * x_increment)
  voltage = (values[i] * y_increment) + y_origin
  f.write("%E, %f\n" % (time_val, voltage))

 f.close()
 print "Waveform format BYTE data written to waveform_data.csv."


# ============================================================
# Send a command and check for errors:
# ============================================================
def do_command(command, hide_params=False):

 if hide_params:
  (header, data) = string.split(command, " ", 1)
  if debug:
   print "\nCmd = '%s'" % header
 else:
  if debug:
   print "\nCmd = '%s'" % command

 Infiniium.write("%s\n" % command)

 if hide_params:
  check_instrument_errors(header)
 else:
  check_instrument_errors(command)


# ============================================================
# Send a query, check for errors, return string:
# ============================================================
def do_query_string(query):
 if debug:
  print "Qys = '%s'" % query
 result = Infiniium.ask("%s\n" % query)
 check_instrument_errors(query)
 return result


# ============================================================
# Send a query, check for errors, return values:
# ============================================================
def do_query_values(query):
 if debug:
  print "Qyv = '%s'" % query
 results = Infiniium.ask_for_values("%s\n" % query)
 check_instrument_errors(query)
 return results


# ============================================================
# Check for instrument errors:
# ============================================================
def check_instrument_errors(command):
```

```
 while True:
  error_string = Infiniium.ask(":SYSTem:ERRor? STRing\n")
  if error_string:   # If there is an error string value.

   if error_string.find("0,", 0, 2) == -1:   # Not "No error".

    print "ERROR: %s, command: '%s'" % (error_string, command)
    print "Exited because of error."
    sys.exit(1)

   else:   # "No error"
    break

  else:   # :SYSTem:ERRor? STRing should always return string.
   print "ERROR: :SYSTem:ERRor? STRing returned nothing, command: '%s'"
% command
   print "Exited because of error."
   sys.exit(1)


# ============================================================
# Returns data from definite-length block.
# ============================================================
def get_definite_length_block_data(sBlock):

 # First character should be "#".
 pound = sBlock[0:1]
 if pound != "#":
  print "PROBLEM: Invalid binary block format, pound char is '%s'." % po
und
  print "Exited because of problem."
  sys.exit(1)

 # Second character is number of following digits for length value.
 digits = sBlock[1:2]

 # Get the data out of the block and return it.
 sData = sBlock[int(digits) + 2:]

 return sData


# ============================================================
# Main program:
# ============================================================

Infiniium = visa.instrument("TCPIP0::130.29.71.191::inst0::INSTR")
Infiniium.timeout = 20
Infiniium.term_chars = ""
Infiniium.clear()

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()
```

```
print "End of program."
```

SICL Examples

- "SICL Example in C" on page 1162
- "SICL Example in Visual Basic" on page 1171

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2008:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2008, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
6. Edit the program to use the SICL address of your oscilloscope.
7. Choose **Project > Properties...**. In the Property Pages dialog, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "sicl32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Click **OK** to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, select **VC++ Directories** under Projects and Solutions.
   c. Show directories for **Include files**, and add the include directory (for example, Program Files\Agilent\ IO Libraries Suite\include).
   d. Show directories for **Library files**, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
   e. Click **OK** to close the Options dialog.
9. Build and run the program.

```
/*
 * Agilent SICL Example in C
 * ----------------------------------------------------------------
 * This program illustrates a few commonly-used programming
 * features of your Agilent Infiniium Series oscilloscope.
 */
```

```
#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <sicl.h>             /* Agilent SICL routines. */

#define SICL_ADDRESS      "lan[130.29.71.191]:inst0"
#define TIMEOUT           15000
#define IEEEBLOCK_SPACE   500000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);    /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE block. */
int do_query_ieeeblock_words(char *query);   /* Query for word data. */
void check_instrument_errors();        /* Check for inst errors. */

/* Global variables */
INST id;                        /* Device session ID. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
signed short ieeeblock_data_words[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock_words(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------- */
void main(void)
{
  /* Install a default SICL error handler that logs an error message
   * and exits.  On Windows 98SE or Windows Me, view messages with
   * the SICL Message Viewer.  For Windows 2000 or XP, use the Event
   * Viewer.
   */
  ionerror(I_ERROR_EXIT);

  /* Open a device session using the SICL_ADDRESS */
  id = iopen(SICL_ADDRESS);

  if (id == 0)
  {
    printf ("Oscilloscope iopen failed!\n");
  }
  else
  {
    printf ("Oscilloscope session opened!\n");
  }
```

```
  /* Set the I/O timeout value for this session to 5 seconds. */
  itimeout(id, TIMEOUT);

  /* Clear the interface. */
  iclear(id);

  /* Initialize - start from a known state. */
  initialize();

  /* Capture data. */
  capture();

  /* Analyze the captured waveform. */
  analyze();

  /* Close the device session to the instrument. */
  iclose(id);
  printf ("Program execution is complete...\n");

  /* For WIN16 programs, call _siclcleanup before exiting to release
   * resources allocated by SICL for this application.  This call is
   * a no-op for WIN32 programs.
   */
  _siclcleanup();
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize (void)
{
  /* Clear status. */
  do_command("*CLS");

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture (void)
{
  int num_values;
  FILE *fp;

  /* Set probe attenuation factor. */
  do_command(":CHANnel1:PROBe 1.0");
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope.
   * ------------------------------------------------------------------- */
  do_command(":AUToscale");
```

```
/* Set trigger mode. */
do_command(":TRIGger:MODE EDGE");
do_query_string(":TRIGger:MODE?");
printf("Trigger mode: %s\n", str_result);

/* Set EDGE trigger parameters. */
do_command(":TRIGger:EDGE:SOURCe CHANnel1");
do_query_string(":TRIGger:EDGE:SOURce?");
printf("Trigger edge source: %s\n", str_result);

do_command(":TRIGger:LEVel CHANnel1,-2E-3");
do_query_string(":TRIGger:LEVel? CHANnel1");
printf("Trigger level, channel 1: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope configuration.
 * ------------------------------------------------------------- */

/* Read system setup. */
num_values = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_values);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_values);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:
 * ------------------------------------------------------------- */

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.1");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet 0.0");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and position. */
do_command(":TIMebase:SCALe 0.0002");
do_query_string(":TIMebase:SCALe?");
printf("Timebase scale: %s\n", str_result);

do_command(":TIMebase:POSition 0.0");
do_query_string(":TIMebase:POSition?");
printf("Timebase position: %s\n", str_result);

/* Set the acquisition mode. */
do_command(":ACQuire:MODE RTIMe");
```

```
  do_query_string(":ACQuire:MODE?");
  printf("Acquire mode: %s\n", str_result);

  /* Or, configure by loading a previously saved setup.
  * -------------------------------------------------------------- */

  /* Read setup string from file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
  num_values = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
  fclose (fp);
  printf("Read setup string (%d bytes) from file ", num_values);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Restore setup string. */
  num_values = do_command_ieeeblock(":SYSTem:SETup", num_values);
  printf("Restored setup string (%d bytes).\n", num_values);

  /* Set the desired number of waveform points,
   * and capture an acquisition. */
  do_command(":ACQuire:POINts 32000");
  do_command(":DIGitize");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------------ */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double y_increment;
  double y_origin;

  FILE *fp;
  int num_values;   /* Number of bytes returned from instrument. */
  int i;

  /* Make measurements.
   * -------------------------------------------------------------- */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
  printf("Frequency: %.4f kHz\n", num_result / 1000);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  /* Download the screen image.
   * -------------------------------------------------------------- */
```

```
/* Read screen image. */
num_values = do_query_ieeeblock(":DISPlay:DATA? PNG");
printf("Screen image bytes: %d\n", num_values);

/* Write screen image bytes to file. */
fp = fopen ("c:\\scope\\data\\screen.png", "wb");
num_values = fwrite(ieeeblock_data, sizeof(unsigned char), num_values,
  fp);
fclose (fp);
printf("Wrote screen image (%d bytes) to ", num_values);
printf("c:\\scope\\data\\screen.png.\n");

/* Download waveform data.
 * ------------------------------------------------------------- */

/* Get the waveform type. */
do_query_string(":WAVeform:TYPE?");
printf("Waveform type: %s\n", str_result);

/* Get the number of waveform points. */
do_query_string(":WAVeform:POINts?");
printf("Waveform points: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned: */
do_command(":WAVeform:FORMat WORD");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_number(":WAVeform:XINCrement?");
x_increment = num_result;
printf("Waveform X increment: %e\n", x_increment);

do_query_number(":WAVeform:XORigin?");
x_origin = num_result;
printf("Waveform X origin: %e\n", x_origin);

do_query_number(":WAVeform:YINCrement?");
y_increment = num_result;
printf("Waveform Y increment: %e\n", y_increment);

do_query_number(":WAVeform:YORigin?");
y_origin = num_result;
printf("Waveform Y origin: %e\n", y_origin);

/* Read waveform data. */
num_values = do_query_ieeeblock_words(":WAVeform:DATA?");
printf("Number of data values: %d\n", num_values);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");
```

```
  /* Output waveform data in CSV format. */
  for (i = 0; i < num_values - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      ((float)ieeeblock_data_words[i] * y_increment) + y_origin);
  }

  /* Close output file. */
   fclose(fp);
   printf("Waveform format WORD data written to ");
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ------------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];

  strcpy(message, command);
  strcat(message, "\n");
  iprintf(id, message);

  check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ------------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  iprintf(id, message, num_bytes);
  ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------------- */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
```

```
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%t\n", str_result);

  check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------------ */
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%lf", &num_result);

  check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------------ */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%,10lf\n", dbl_results);

  check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------------ */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  data_length = IEEEBLOCK_SPACE;
  iscanf(id, "%#b", &data_length, ieeeblock_data);

  if (data_length == IEEEBLOCK_SPACE )
  {
```

```
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Query for an IEEE definite-length block word data result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock_words(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  data_length = IEEEBLOCK_SPACE;
  iscanf(id, "%#wb", &data_length, ieeeblock_data_words);

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  check_instrument_errors();

  return(data_length);
}

/* Check for instrument errors.
 * ------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  ipromptf(id, ":SYSTem:ERRor? STRing\n", "%t", str_err_val);
  while(strncmp(str_err_val, "0,", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    ipromptf(id, ":SYSTem:ERRor? STRing\n", "%t", str_err_val);
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
    iflush(id, I_BUF_READ | I_BUF_WRITE);
  }
}
```

SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

1. Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).
2. Press ALT+F11 to launch the Visual Basic editor.
3. Add the sicl32.bas file to your project:
   a. Choose **File>Import File...**.
   b. Navigate to the header file, sicl32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.
4. Choose **Insert>Module**.
5. Cut-and-paste the code that follows into the editor.
6. Edit the program to use the SICL address of your oscilloscope, and save the changes.
7. Run the program.

```
'
' Agilent SICL Example in Visual Basic
' ----------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent Infiniium Series oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public id As Integer   ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/iread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ----------------------------------------------------------------------

Sub Main()

  On Error GoTo ErrorHandler
```

```
  ' Open a device session using the SICL_ADDRESS.
  id = iopen("lan[130.29.71.191]:inst0")
  Call itimeout(id, 15000)

  ' Clear the interface.
  Call iclear(id)

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  Call iclose(id)

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo ErrorHandler

  ' Clear status.
  DoCommand "*CLS"

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

  ' Load the default setup.
  DoCommand "*RST"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Capture the waveform.
```

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

Private Sub Capture()

  On Error GoTo ErrorHandler

  ' Set probe attenuation factor.
  DoCommand ":CHANnel1:PROBe 1.0"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically configure oscilloscope.
  ' ----------------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Set trigger mode.
  DoCommand ":TRIGger:MODE EDGE"
  Debug.Print "Trigger mode: " + _
      DoQueryString(":TRIGger:MODE?")

  ' Set EDGE trigger parameters.
  DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
  Debug.Print "Trigger edge source: " + _
      DoQueryString(":TRIGger:EDGE:SOURce?")

  DoCommand ":TRIGger:LEVel CHANnel1,-2E-3"
  Debug.Print "Trigger level, channel 1: " + _
      DoQueryString(":TRIGger:LEVel? CHANnel1")

  DoCommand ":TRIGger:EDGE:SLOPe POSitive"
  Debug.Print "Trigger edge slope: " + _
      DoQueryString(":TRIGger:EDGE:SLOPe?")

  ' Save oscilloscope configuration.
  ' ----------------------------------------------------------------------
  Dim lngSetupStringSize As Long
  lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
  Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

  ' Output setup string to a file:
  Dim strPath As String
  strPath = "c:\scope\config\setup.dat"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If

  ' Open file for output.
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.

  ' Change settings with individual commands:
```

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

  ' Set vertical scale and offset.
  DoCommand ":CHANnel1:SCALe 0.1"
  Debug.Print "Channel 1 vertical scale: " + _
      DoQueryString(":CHANnel1:SCALe?")

  DoCommand ":CHANnel1:OFFSet 0.0"
  Debug.Print "Channel 1 vertical offset: " + _
      DoQueryString(":CHANnel1:OFFSet?")

  ' Set horizontal scale and position.
  DoCommand ":TIMebase:SCALe 0.0002"
  Debug.Print "Timebase scale: " + _
      DoQueryString(":TIMebase:SCALe?")

  DoCommand ":TIMebase:POSition 0.0"
  Debug.Print "Timebase position: " + _
      DoQueryString(":TIMebase:POSition?")

  ' Set the acquisition mode.
  DoCommand ":ACQuire:MODE RTIMe"
  Debug.Print "Acquire mode: " + _
      DoQueryString(":ACQuire:MODE?")

  ' Or, configure by loading a previously saved setup.
  ' ------------------------------------------------------------------
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As hFile    ' Open file for input.
  Dim lngSetupFileSize As Long
  lngSetupFileSize = LOF(hFile)  ' Length of file.
  Get hFile, , byteArray    ' Read data.
  Close hFile    ' Close file.
  ' Write setup string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  Dim lngRestored As Long
  lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  ' ------------------------------------------------------------------
  DoCommand ":ACQuire:POINts 32000"
  DoCommand ":DIGitize"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Analyze the captured waveform.
' ------------------------------------------------------------------
```

```
Private Sub Analyze()

  On Error GoTo ErrorHandler

  ' Make measurements.
  ' --------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:FREQuency"
  dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

  DoCommand ":MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertical amplitude:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"

  ' Download the screen image.
  ' --------------------------------------------------------------------

  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' Save screen image to a file:
  Dim strPath As String
  strPath = "c:\scope\data\screen.png"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  ' Skip past header.
  For lngI = CInt(Chr(byteArray(1))) + 2 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
  MsgBox "Screen image written to " + strPath

  ' Download waveform data.
  ' --------------------------------------------------------------------

  ' Get the waveform type.
  Debug.Print "Waveform type: " + _
      DoQueryString(":WAVeform:TYPE?")

  ' Get the number of waveform points.
  Debug.Print "Waveform points: " + _
      DoQueryString(":WAVeform:POINts?")
```

```
' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned:
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double

dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment: " + _
    Format(dblYIncrement, "Scientific")

dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin: " + _
    FormatNumber(dblYOrigin, 0)

' Get the waveform data
DoCommand ":WAVeform:STReaming OFF"
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Number of data values: " + _
    CStr(lngNumBytes - CInt(Chr(byteArray(1))) - 2)

' Set up output file:
strPath = "c:\scope\data\waveform_data.csv"

' Open file for output.
Open strPath For Output Access Write Lock Write As hFile

' Output waveform data in CSV format.
Dim lngDataValue As Long
Dim byteUnsigned As Byte

' Skip past header.
For lngI = CInt(Chr(byteArray(1))) + 2 To lngNumBytes - 2
  byteUnsigned = byteArray(lngI)
  ' Oscilloscope BYTE format sends signed bytes. VBA Byte is
  ' interpreted as unsigned, so convert the bits to signed value.
  lngDataValue = byteUnsigned - ((byteUnsigned And &H80) * 2)

  ' Write time value, voltage value.
```

```
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
        ", " + _
        FormatNumber((lngDataValue * dblYIncrement) + dblYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo ErrorHandler

  Call ivprintf(id, command + vbLf)

  CheckInstrumentErrors

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  On Error GoTo ErrorHandler

  ' Send command part.
  Call ivprintf(id, command + " ")

  ' Write definite-length block bytes.
  Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)

  ' retCount is now actual number of bytes written.
  DoCommandIEEEBlock = retCount

  CheckInstrumentErrors

  Exit Function

ErrorHandler:
```

```
  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryString(query As String) As String

  Dim actual As Long

  On Error GoTo ErrorHandler

  Dim strResult As String * 200

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%200t", strResult)
  DoQueryString = strResult

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumber(query As String) As Double

  On Error GoTo ErrorHandler

  Dim dblResult As Double

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%lf" + vbLf, dblResult)
  DoQueryNumber = dblResult

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumbers(query As String) As Double()

  On Error GoTo ErrorHandler

  Dim dblResults(10) As Double

  Call ivprintf(id, query + vbLf)
```

```
  Call ivscanf(id, "%,10lf" + vbLf, dblResults)
  DoQueryNumbers = dblResults

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  On Error GoTo ErrorHandler

  ' Send query.
  Call ivprintf(id, query + vbLf)

  ' Read definite-length block bytes.
  Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)

  ' Get number of block length digits.
  Dim intLengthDigits As Integer
  intLengthDigits = CInt(Chr(byteArray(1)))

  ' Get block length from those digits.
  Dim strBlockLength As String
  strBlockLength = ""
  Dim i As Integer
  For i = 2 To intLengthDigits + 1
    strBlockLength = strBlockLength + Chr(byteArray(i))
  Next

  ' Return number of bytes in block plus header.
  DoQueryIEEEBlock_Bytes = CLng(strBlockLength) + intLengthDigits + 2

  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String
```

```
  Call ivprintf(id, ":SYSTem:ERRor? STRing" + vbLf) ' Query any errors d
ata.
  Call ivscanf(id, "%200t", strErrVal)    ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    Call ivprintf(id, ":SYSTem:ERRor? STRing" + vbLf) ' Request error me
ssage.
    Call ivscanf(id, "%200t", strErrVal)    ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    Call iflush(id, I_BUF_READ Or I_BUF_WRITE)

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub
```

SCPI.NET Examples

These programming examples show how to use the SCPI.NET drivers that come with Agilent's free Command Expert software.

While you can write code manually using SCPI.NET drivers (as described in this section), you can also use the Command Expert software to:

- Connect to instruments and control them interactively using SCPI command sets.
- Quickly prototype and test command sequences.
- Generate C#, VB.NET, or C/C++ code for command sequences.
- Find, download, and install SCPI command sets.
- Browse command trees, search for commands, and view command descriptions.

The Command Expert suite also comes with Add-ons for easy instrument control and measurement data retrieval in NI LabVIEW, Microsoft Excel, Agilent VEE, and Agilent SystemVue.

For more information on Agilent Command Expert, and to download the software, see: "http://www.agilent.com/find/commandexpert"

- "SCPI.NET Example in C#" on page 1181
- "SCPI.NET Example in Visual Basic .NET" on page 1189
- "SCPI.NET Example in IronPython" on page 1196

SCPI.NET Example in C#

To compile and run this example in Microsoft Visual Studio 2008:

1. Install the Agilent Command Expert software and the command set for the oscilloscope.
2. Open Visual Studio.
3. Create a new Visual C#, Windows, Console Application project.
4. Cut-and-paste the code that follows into the C# source file.
5. Edit the program to use the address of your oscilloscope.
6. Add a reference to the SCPI.NET driver:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **Browse** tab, and navigate to the ScpiNetDrivers folder.

- Windows XP: C:\Documents and Settings\All Users\Agilent\Command Expert\ScpiNetDrivers
- Windows 7: C:\ProgramData\Agilent\Command Expert\ScpiNetDrivers

d Select the .dll file for your oscilloscope, for example **AgInfiniium90000_1_10.dll**; then, click **OK**.

7 Build and run the program.

For more information, see the SCPI.NET driver help that comes with Agilent Command Expert.

```
/*
 * Agilent SCPI.NET Example in C#
 * ----------------------------------------------------------------------
 * This program illustrates a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using Agilent.CommandExpert.ScpiNet.AgInfiniium90000_3_10;

namespace Infiniium
{
  class ScpiNetInstrumentApp
  {
    private static AgInfiniium90000 myScope;

    static void Main(string[] args)
    {
      try
      {
        string strScopeAddress;
        //strScopeAddress = "labi-trex-lp11.cos.agilent.com";
        strScopeAddress =
          "TCPIP0::labi-trex-lp11.cos.agilent.com::inst0::INSTR";
        Console.WriteLine("Connecting to oscilloscope...");
        Console.WriteLine();
        myScope = new AgInfiniium90000(strScopeAddress);
        myScope.Transport.DefaultTimeout.Set(10000);

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();

        Console.WriteLine("Press any key to exit");
```

```
    Console.ReadKey();
  }
  catch (System.ApplicationException err)
  {
    Console.WriteLine("*** SCPI.NET Error : " + err.Message);
  }
  catch (System.SystemException err)
  {
    Console.WriteLine("*** System Error Message : " + err.Message);
  }
  catch (System.Exception err)
  {
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
  }
  finally
  {
    //myScope.Dispose();
  }

}

/*
 * Initialize the oscilloscope to a known state.
 * ---------------------------------------------------------------------
 */
private static void Initialize()
{
  string strResults;

  // Get and display the device's *IDN? string.
  myScope.SCPI.IDN.Query(out strResults);
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.SCPI.CLS.Command();
  myScope.SCPI.RST.Command();
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------------
 */
private static void Capture()
{
  string strResults;
  double fResult;

  // Set probe attenuation factor.
  myScope.SCPI.CHANnel.PROBe.Command(1, 1.0, null);
  myScope.SCPI.CHANnel.PROBe.Query(1, out fResult, out strResults);
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      fResult);

  // Use auto-scale to automatically configure oscilloscope.
  myScope.SCPI.AUToscale.Command();
```

```
// Set trigger mode.
myScope.SCPI.TRIGger.MODE.Command("EDGE");
myScope.SCPI.TRIGger.MODE.Query(out strResults);
Console.WriteLine("Trigger mode: {0}", strResults);

// Set EDGE trigger parameters.
myScope.SCPI.TRIGger.EDGE.SOURce.Command(null, "CHANnel1");
myScope.SCPI.TRIGger.EDGE.SOURce.Query(null, out strResults);
Console.WriteLine("Trigger edge source: {0}", strResults);

myScope.SCPI.TRIGger.LEVel.Command("CHANnel1", -0.002);
myScope.SCPI.TRIGger.LEVel.Query("CHANnel1", out fResult);
Console.WriteLine("Trigger edge level: {0:F2}", fResult);

myScope.SCPI.TRIGger.EDGE.SLOPe.Command(null, "POSitive");
myScope.SCPI.TRIGger.EDGE.SLOPe.Query(null, out strResults);
Console.WriteLine("Trigger edge slope: {0}", strResults);

// Save oscilloscope setup.
byte[] byteResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
myScope.SCPI.SYSTem.SETup.Query(out byteResultsArray);
nLength = byteResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(byteResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.SCPI.CHANnel.SCALe.Command(1, 0.1);
myScope.SCPI.CHANnel.SCALe.Query(1, out fResult);
Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult);

myScope.SCPI.CHANnel.OFFSet.Command(1, 0.0);
myScope.SCPI.CHANnel.OFFSet.Query(1, out fResult);
Console.WriteLine("Channel 1 vertical offset: {0:F4}", fResult);

// Set horizontal scale and offset.
myScope.SCPI.TIMebase.SCALe.Command(0.0002);
myScope.SCPI.TIMebase.SCALe.Query(out fResult);
Console.WriteLine("Timebase scale: {0:F4}", fResult);

myScope.SCPI.TIMebase.POSition.Command(0.0);
myScope.SCPI.TIMebase.POSition.Query(out fResult);
Console.WriteLine("Timebase position: {0:F2}", fResult);

// Set the acquisition mode.
myScope.SCPI.ACQuire.MODE.Command("RTIMe");
myScope.SCPI.ACQuire.MODE.Query(out strResults);
```

```
  Console.WriteLine("Acquire mode: {0}", strResults);

  // Or, configure by loading a previously saved setup.
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);
  nBytesWritten = DataArray.Length;

  // Restore setup string.
  myScope.SCPI.SYSTem.SETup.Command(DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Set the desired number of waveform points,
  // and capture an acquisition.
  myScope.SCPI.ACQuire.POINts.Command(32000);
  myScope.SCPI.DIGitize.Command(null);
}

/*
 * Analyze the captured waveform.
 * -------------------------------------------------------------
 */
private static void Analyze()
{
  string[] strResultsArray;   // Results array.
  string strResults;
  double fResult;

  // Make measurements.
  // ------------------------------------------------------------
  myScope.SCPI.MEASure.SOURce.Command("CHANnel1", null);
  myScope.SCPI.MEASure.SOURce.Queryx(out strResultsArray);
  Console.WriteLine("Measure source: {0}", strResultsArray[0]);

  myScope.SCPI.MEASure.FREQuency.Command("CHANnel1", null);
  myScope.SCPI.MEASure.FREQuency.QuerySendValidOff("CHANnel1",
      null, out fResult);
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  // Use direct command/query when commands not in command set.
  myScope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1");
  myScope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1",
    out strResults);
  Console.WriteLine("Vertical amplitude: {0} V", strResults);

  // Download the screen image.
  // ------------------------------------------------------------

  // Get the screen data.
  byte[] byteResultsArray;   // Results array.
  myScope.SCPI.DISPlay.DATA.Query("PNG", null, null, null,
      out byteResultsArray);
  int nLength;   // Number of bytes returned from instrument.
  nLength = byteResultsArray.Length;
```

```
// Store the screen data to a file.
string strPath;
strPath = "c:\\scope\\data\\screen.png";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(byteResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Screen image ({0} bytes) written to {1}",
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Get the waveform points mode.
myScope.SCPI.WAVeform.TYPE.Query(out strResults);
Console.WriteLine("Waveform points mode: {0}", strResults);

// Get the number of waveform points.
int nPointsAvail;
myScope.SCPI.WAVeform.POINts.Query(out nPointsAvail);
Console.WriteLine("Waveform points: {0}", nPointsAvail);

// Set the waveform source.
myScope.SCPI.WAVeform.SOURce.Command("CHANnel1");
myScope.SCPI.WAVeform.SOURce.Query(out strResults);
Console.WriteLine("Waveform source: {0}", strResults);

// Choose the format of the data returned:
myScope.SCPI.WAVeform.FORMat.Command("WORD");
myScope.SCPI.WAVeform.FORMat.Query(out strResults);
Console.WriteLine("Waveform format: {0}", strResults);

myScope.SCPI.WAVeform.BYTeorder.Command("LSBFirst");
myScope.SCPI.WAVeform.BYTeorder.Query(out strResults);
Console.WriteLine("Waveform byte order: {0}", strResults);

// Display the waveform settings from preamble:
Dictionary<string, string> dctWavFormat =
  new Dictionary<string, string>()
{
  {"0", "ASCii"},
  {"1", "BYTE"},
  {"2", "WORD"},
  {"3", "LONG"},
  {"4", "LONGLONG"},
};
Dictionary<string, string> dctAcqType =
  new Dictionary<string, string>()
{
  {"1", "RAW"},
  {"2", "AVERage"},
  {"3", "VHIStogram"},
  {"4", "HHIStogram"},
  {"6", "INTerpolate"},
  {"10", "PDETect"},
};
Dictionary<string, string> dctAcqMode =
```

```
  new Dictionary<string, string>()
{
  {"0", "RTIMe"},
  {"1", "ETIMe"},
  {"3", "PDETect"},
};
Dictionary<string, string> dctCoupling =
  new Dictionary<string, string>()
{
  {"0", "AC"},
  {"1", "DC"},
  {"2", "DCFIFTY"},
  {"3", "LFREJECT"},
};
Dictionary<string, string> dctUnits =
  new Dictionary<string, string>()
{
  {"0", "UNKNOWN"},
  {"1", "VOLT"},
  {"2", "SECOND"},
  {"3", "CONSTANT"},
  {"4", "AMP"},
  {"5", "DECIBEL"},
};
string strPreamble;
string[] strsPreamble;

myScope.SCPI.WAVeform.PREamble.Query(out strPreamble);
strsPreamble = strPreamble.Split(',');

Console.WriteLine("Waveform format: {0}",
    dctWavFormat[strsPreamble[0]]);

Console.WriteLine("Acquire type: {0}",
    dctAcqType[strsPreamble[1]]);

Console.WriteLine("Waveform points: {0}", strsPreamble[2]);
Console.WriteLine("Waveform average count: {0}", strsPreamble[3]);
Console.WriteLine("Waveform X increment: {0}", strsPreamble[4]);
Console.WriteLine("Waveform X origin: {0}", strsPreamble[5]);
Console.WriteLine("Waveform X reference: {0}", strsPreamble[6]);
Console.WriteLine("Waveform Y increment: {0}", strsPreamble[7]);
Console.WriteLine("Waveform Y origin: {0}", strsPreamble[8]);
Console.WriteLine("Waveform Y reference: {0}", strsPreamble[9]);
Console.WriteLine("Coupling: {0}", dctCoupling[strsPreamble[10]]);
Console.WriteLine("Waveform X display range: {0}",
    strsPreamble[11]);
Console.WriteLine("Waveform X display origin: {0}",
    strsPreamble[12]);
Console.WriteLine("Waveform Y display range: {0}",
    strsPreamble[13]);
Console.WriteLine("Waveform Y display origin: {0}",
    strsPreamble[14]);
Console.WriteLine("Date: {0}", strsPreamble[15]);
Console.WriteLine("Time: {0}", strsPreamble[16]);
Console.WriteLine("Frame model: {0}", strsPreamble[17]);
Console.WriteLine("Acquire mode: {0}",
```

```
            dctAcqMode[strsPreamble[18]]);
        Console.WriteLine("Completion pct: {0}", strsPreamble[19]);
        Console.WriteLine("Waveform X inits: {0}",
            dctUnits[strsPreamble[20]]);
        Console.WriteLine("Waveform Y units: {0}",
            dctUnits[strsPreamble[21]]);
        Console.WriteLine("Max BW limit: {0}", strsPreamble[22]);
        Console.WriteLine("Min BW limit: {0}", strsPreamble[23]);

        // Get numeric values for later calculations.
        double fXincrement;
        myScope.SCPI.WAVeform.XINCrement.Query(out fXincrement);
        double fXorigin;
        myScope.SCPI.WAVeform.XORigin.Query(out fXorigin);
        double fYincrement;
        myScope.SCPI.WAVeform.YINCrement.Query(out fYincrement);
        double fYorigin;
        myScope.SCPI.WAVeform.YORigin.Query(out fYorigin);

        // Get the waveform data.
        myScope.SCPI.WAVeform.STReaming.Command(false);
        short[] WordDataArray;   // Results array.
        myScope.SCPI.WAVeform.DATA.QueryBlockInt16(null, null, out WordDat
aArray);
        nLength = WordDataArray.Length;
        Console.WriteLine("Number of data values: {0}", nLength);

        // Set up output file:
        strPath = "c:\\scope\\data\\waveform_data.csv";
        if (File.Exists(strPath)) File.Delete(strPath);

        // Open file for output.
        StreamWriter writer = File.CreateText(strPath);

        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i++)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)i * fXincrement),
              (((float)WordDataArray[i])
              * fYincrement) + fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format WORD data written to {0}",
            strPath);
    }
  }
}
```

SCPI.NET Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2008:

1. Install the Agilent Command Expert software and the command set for the oscilloscope.
2. Open Visual Studio.
3. Create a new Visual Basic, Windows, Console Application project.
4. Cut-and-paste the code that follows into the Visual Basic .NET source file.
5. Edit the program to use the VISA address of your oscilloscope.
6. Add a reference to the SCPI.NET 3.0 driver:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **Browse** tab, and navigate to the ScpiNetDrivers folder.
      - Windows XP: C:\Documents and Settings\All Users\Agilent\Command Expert\ScpiNetDrivers
      - Windows 7: C:\ProgramData\Agilent\Command Expert\ScpiNetDrivers
   d. Select the .dll file for your oscilloscope, for example **AgInfiniium90000_1_10.dll**; then, click **OK**.
   e. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "Infiniium.ScpiNetInstrumentApp" as the **Startup object**.
7. Build and run the program.

For more information, see the SCPI.NET driver help that comes with Agilent Command Expert.

```
'
' Agilent SCPI.NET Example in Visual Basic .NET
' ----------------------------------------------------------------------
' This program illustrates a few commonly used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports System.Collections.Generic
Imports Agilent.CommandExpert.ScpiNet.AgInfiniium90000_3_10
```

```
Namespace Infiniium
  Class ScpiNetInstrumentApp
    Private Shared myScope As AgInfiniium90000

    Public Shared Sub Main(ByVal args As String())
      Try
        Dim strScopeAddress As String
        'strScopeAddress = "labi-trex-lp11.cos.agilent.com";
        strScopeAddress = _
            "TCPIP0::labi-trex-lp11.cos.agilent.com::inst0::INSTR"
        Console.WriteLine("Connecting to oscilloscope...")
        Console.WriteLine()
        myScope = New AgInfiniium90000(strScopeAddress)
        myScope.Transport.DefaultTimeout.[Set](10000)

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

        Console.WriteLine("Press any key to exit")
        Console.ReadKey()
      Catch err As System.ApplicationException
        Console.WriteLine("*** SCPI.NET Error : " & err.Message)
      Catch err As System.SystemException
        Console.WriteLine("*** System Error Message : " & err.Message)
      Catch err As System.Exception
        System.Diagnostics.Debug.Fail("Unexpected Error")
        Console.WriteLine("*** Unexpected Error : " & err.Message)
        'myScope.Dispose();
      Finally
      End Try

    End Sub

    ' Initialize the oscilloscope to a known state.
    ' ---------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As String

      ' Get and display the device's *IDN? string.
      myScope.SCPI.IDN.Query(strResults)
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Clear status and load the default setup.
      myScope.SCPI.CLS.Command()
      myScope.SCPI.RST.Command()
    End Sub

    ' Capture the waveform.
    ' ---------------------------------------------------------------
```

```
Private Shared Sub Capture()
  Dim strResults As String
  Dim fResult As Double

  ' Set probe attenuation factor.
  Dim nChannel As UInteger
  nChannel = 1
  myScope.SCPI.CHANnel.PROBe.Command(nChannel, 1.0, DBNull.Value)
  myScope.SCPI.CHANnel.PROBe.Query(1, fResult, strResults)
  Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
      fResult)

  ' Use auto-scale to automatically configure oscilloscope.
  myScope.SCPI.AUToscale.Command()

  ' Set trigger mode.
  myScope.SCPI.TRIGger.MODE.Command("EDGE")
  myScope.SCPI.TRIGger.MODE.Query(strResults)
  Console.WriteLine("Trigger mode: {0}", strResults)

  ' Set EDGE trigger parameters.
  myScope.SCPI.TRIGger.EDGE.SOURce.Command(DBNull.Value, "CHANnel1")
  myScope.SCPI.TRIGger.EDGE.SOURce.Query(DBNull.Value, strResults)
  Console.WriteLine("Trigger edge source: {0}", strResults)

  myScope.SCPI.TRIGger.LEVel.Command("CHANnel1", -0.002)
  myScope.SCPI.TRIGger.LEVel.Query("CHANnel1", fResult)
  Console.WriteLine("Trigger edge level: {0:F2}", fResult)

  myScope.SCPI.TRIGger.EDGE.SLOPe.Command(DBNull.Value, "POSitive")
  myScope.SCPI.TRIGger.EDGE.SLOPe.Query(DBNull.Value, strResults)
  Console.WriteLine("Trigger edge slope: {0}", strResults)

  ' Save oscilloscope setup.
  Dim byteResultsArray As Byte()
  ' Results array.
  Dim nLength As Integer
  ' Number of bytes returned from instrument.
  Dim strPath As String

  ' Query and read setup string.
  myScope.SCPI.SYSTem.SETup.Query(byteResultsArray)
  nLength = byteResultsArray.Length

  ' Write setup string to file.
  strPath = "c:\scope\config\setup.stp"
  Dim fStream As FileStream = File.Open(strPath, FileMode.Create)
  fStream.Write(byteResultsArray, 0, nLength)
  fStream.Close()
  Console.WriteLine("Setup bytes saved: {0}", nLength)

  ' Change settings with individual commands:

  ' Set vertical scale and offset.
  myScope.SCPI.CHANnel.SCALe.Command(1, 0.1)
  myScope.SCPI.CHANnel.SCALe.Query(1, fResult)
  Console.WriteLine("Channel 1 vertical scale: {0:F4}", fResult)
```

```
  myScope.SCPI.CHANnel.OFFSet.Command(1, 0.0)
  myScope.SCPI.CHANnel.OFFSet.Query(1, fResult)
  Console.WriteLine("Channel 1 vertical offset: {0:F4}", fResult)

  ' Set horizontal scale and offset.
  myScope.SCPI.TIMebase.SCALe.Command(0.0002)
  myScope.SCPI.TIMebase.SCALe.Query(fResult)
  Console.WriteLine("Timebase scale: {0:F4}", fResult)

  myScope.SCPI.TIMebase.POSition.Command(0.0)
  myScope.SCPI.TIMebase.POSition.Query(fResult)
  Console.WriteLine("Timebase position: {0:F2}", fResult)

  ' Set the acquisition mode.
  myScope.SCPI.ACQuire.MODE.Command("RTIMe")
  myScope.SCPI.ACQuire.MODE.Query(strResults)
  Console.WriteLine("Acquire mode: {0}", strResults)

  ' Or, configure by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.stp"
  DataArray = File.ReadAllBytes(strPath)
  nBytesWritten = DataArray.Length

  ' Restore setup string.
  myScope.SCPI.SYSTem.SETup.Command(DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Set the desired number of waveform points,
  ' and capture an acquisition.
  myScope.SCPI.ACQuire.POINts.Command(32000)
  myScope.SCPI.DIGitize.Command(DBNull.Value)
End Sub

' Analyze the captured waveform.
' ------------------------------------------------------------

Private Shared Sub Analyze()
  Dim strResultsArray As String()
  ' Results array.
  Dim strResults As String
  Dim fResult As Double

  ' Make measurements.
  ' ------------------------------------------------------------
  myScope.SCPI.MEASure.SOURce.Command("CHANnel1", DBNull.Value)
  myScope.SCPI.MEASure.SOURce.Queryx(strResultsArray)
  Console.WriteLine("Measure source: {0}", strResultsArray(0))

  myScope.SCPI.MEASure.FREQuency.Command("CHANnel1", DBNull.Value)
  myScope.SCPI.MEASure.FREQuency.QuerySendValidOff("CHANnel1", _
      DBNull.Value, fResult)
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)
```

```
' Use direct command/query when commands not in command set.
myScope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1")
myScope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1", _
    strResults)
Console.WriteLine("Vertical amplitude: {0} V", strResults)

' Download the screen image.
' -------------------------------------------------------------

' Get the screen data.
Dim byteResultsArray As Byte()
' Results array.
myScope.SCPI.DISPlay.DATA.Query("PNG", DBNull.Value, _
    DBNull.Value, DBNull.Value, byteResultsArray)
Dim nLength As Integer
' Number of bytes returned from instrument.
nLength = byteResultsArray.Length

' Store the screen data to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.png"
Dim fStream As FileStream = File.Open(strPath, FileMode.Create)
fStream.Write(byteResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' -------------------------------------------------------------

' Get the waveform points mode.
myScope.SCPI.WAVeform.TYPE.Query(strResults)
Console.WriteLine("Waveform points mode: {0}", strResults)

' Get the number of waveform points.
Dim nPointsAvail As Integer
myScope.SCPI.WAVeform.POINts.Query(nPointsAvail)
Console.WriteLine("Waveform points: {0}", nPointsAvail)

' Set the waveform source.
myScope.SCPI.WAVeform.SOURce.Command("CHANnel1")
myScope.SCPI.WAVeform.SOURce.Query(strResults)
Console.WriteLine("Waveform source: {0}", strResults)

' Choose the format of the data returned:
myScope.SCPI.WAVeform.FORMat.Command("WORD")
myScope.SCPI.WAVeform.FORMat.Query(strResults)
Console.WriteLine("Waveform format: {0}", strResults)

myScope.SCPI.WAVeform.BYTeorder.Command("LSBFirst")
myScope.SCPI.WAVeform.BYTeorder.Query(strResults)
Console.WriteLine("Waveform byte order: {0}", strResults)

' Display the waveform settings from preamble:
Dim dctWavFormat As New Dictionary(Of String, String)
dctWavFormat.Add("0", "ASCii")
```

```
dctWavFormat.Add("1", "BYTE")
dctWavFormat.Add("2", "WORD")
dctWavFormat.Add("3", "LONG")
dctWavFormat.Add("4", "LONGLONG")

Dim dctAcqType As New Dictionary(Of String, String)
dctAcqType.Add("1", "RAW")
dctAcqType.Add("2", "AVERage")
dctAcqType.Add("3", "VHIStogram")
dctAcqType.Add("4", "HHIStogram")
dctAcqType.Add("6", "INTerpolate")
dctAcqType.Add("10", "PDETect")

Dim dctAcqMode As New Dictionary(Of String, String)()
dctAcqMode.Add("0", "RTIMe")
dctAcqMode.Add("1", "ETIMe")
dctAcqMode.Add("3", "PDETect")

Dim dctCoupling As New Dictionary(Of String, String)()
dctCoupling.Add("0", "AC")
dctCoupling.Add("1", "DC")
dctCoupling.Add("2", "DCFIFTY")
dctCoupling.Add("3", "LFREJECT")

Dim dctUnits As New Dictionary(Of String, String)()
dctUnits.Add("0", "UNKNOWN")
dctUnits.Add("1", "VOLT")
dctUnits.Add("2", "SECOND")
dctUnits.Add("3", "CONSTANT")
dctUnits.Add("4", "AMP")
dctUnits.Add("5", "DECIBEL")

Dim strPreamble As String
Dim strsPreamble As String()

myScope.SCPI.WAVeform.PREamble.Query(strPreamble)
strsPreamble = strPreamble.Split(",")c)

Console.WriteLine("Waveform format: {0}", _
    dctWavFormat(strsPreamble(0)))

Console.WriteLine("Acquire type: {0}", _
    dctAcqType(strsPreamble(1)))

Console.WriteLine("Waveform points: {0}", strsPreamble(2))
Console.WriteLine("Waveform average count: {0}", strsPreamble(3))
Console.WriteLine("Waveform X increment: {0}", strsPreamble(4))
Console.WriteLine("Waveform X origin: {0}", strsPreamble(5))
Console.WriteLine("Waveform X reference: {0}", strsPreamble(6))
Console.WriteLine("Waveform Y increment: {0}", strsPreamble(7))
Console.WriteLine("Waveform Y origin: {0}", strsPreamble(8))
Console.WriteLine("Waveform Y reference: {0}", strsPreamble(9))
Console.WriteLine("Coupling: {0}", dctCoupling(strsPreamble(10)))
Console.WriteLine("Waveform X display range: {0}", _
    strsPreamble(11))
Console.WriteLine("Waveform X display origin: {0}", _
    strsPreamble(12))
```

```
    Console.WriteLine("Waveform Y display range: {0}", _
        strsPreamble(13))
    Console.WriteLine("Waveform Y display origin: {0}", _
        strsPreamble(14))
    Console.WriteLine("Date: {0}", strsPreamble(15))
    Console.WriteLine("Time: {0}", strsPreamble(16))
    Console.WriteLine("Frame model: {0}", strsPreamble(17))
    Console.WriteLine("Acquire mode: {0}", _
        dctAcqMode(strsPreamble(18)))
    Console.WriteLine("Completion pct: {0}", strsPreamble(19))
    Console.WriteLine("Waveform X inits: {0}", _
        dctUnits(strsPreamble(20)))
    Console.WriteLine("Waveform Y units: {0}", _
        dctUnits(strsPreamble(21)))
    Console.WriteLine("Max BW limit: {0}", strsPreamble(22))
    Console.WriteLine("Min BW limit: {0}", strsPreamble(23))

    ' Get numeric values for later calculations.
    Dim fXincrement As Double
    myScope.SCPI.WAVeform.XINCrement.Query(fXincrement)
    Dim fXorigin As Double
    myScope.SCPI.WAVeform.XORigin.Query(fXorigin)
    Dim fYincrement As Double
    myScope.SCPI.WAVeform.YINCrement.Query(fYincrement)
    Dim fYorigin As Double
    myScope.SCPI.WAVeform.YORigin.Query(fYorigin)

    ' Get the waveform data.
    myScope.SCPI.WAVeform.STReaming.Command(False)
    Dim WordDataArray As Short()
    ' Results array.
    myScope.SCPI.WAVeform.DATA.QueryBlockInt16(DBNull.Value, _
        DBNull.Value, WordDataArray)
    nLength = WordDataArray.Length
    Console.WriteLine("Number of data values: {0}", nLength)

    ' Set up output file:
    strPath = "c:\scope\data\waveform_data.csv"
    If File.Exists(strPath) Then
      File.Delete(strPath)
    End If

    ' Open file for output.
    Dim writer As StreamWriter = File.CreateText(strPath)

    ' Output waveform data in CSV format.
    For i As Integer = 0 To nLength - 2
      writer.WriteLine("{0:f9}, {1:f6}", _
          fXorigin + (CSng(i) * fXincrement), _
          (CSng(WordDataArray(i)) * fYincrement) + fYorigin)
    Next

    ' Close output file.
    writer.Close()
    Console.WriteLine("Waveform format WORD data written to {0}", _
        strPath)
End Sub
```

```
  End Class
End Namespace
```

SCPI.NET Example in IronPython

You can also control Agilent oscilloscopes using the SCPI.NET library and Python programming language on the .NET platform using:

- IronPython ("http://ironpython.codeplex.com/") which is an implementation of the Python programming language running under .NET.

To run this example with IronPython:

1. Install the Agilent Command Expert software and the command set for the oscilloscope.
2. Cut-and-paste the code that follows into a file named "example.py".
3. Edit the program to use the address of your oscilloscope.
4. If the IronPython "ipy.exe" can be found via your PATH environment variable, open a Command Prompt window; then, change to the folder that contains the "example.py" file, and enter:

```
ipy example.py
```

```
#
# Agilent SCPI.NET Example in IronPython
# ******************************************************
# This program illustrates a few commonly used programming
# features of your Agilent oscilloscope.
# ******************************************************

# Import Python modules.
# ------------------------------------------------------------
import sys
sys.path.append("C:\Python26\Lib")   # Python Standard Library.
sys.path.append("C:\ProgramData\Agilent\Command Expert\ScpiNetDrivers")
import string

# Import .NET modules.
# ------------------------------------------------------------
from System import *
from System.IO import *
from System.Text import *
from System.Runtime.InteropServices import *
import clr
clr.AddReference("AgInfiniium90000_3_10")
from Agilent.CommandExpert.ScpiNet.AgInfiniium90000_3_10 import *


# ============================================================
# Initialize:
# ============================================================
def initialize():
```

```
 # Get and display the device's *IDN? string.
 idn_string = scope.SCPI.IDN.Query()
 print "Identification string '%s'" % idn_string

 # Clear status and load the default setup.
 scope.SCPI.CLS.Command()
 scope.SCPI.RST.Command()


# ============================================================
# Capture:
# ============================================================
def capture():

 # Set probe attenuation factor.
 scope.SCPI.CHANnel.PROBe.Command(1, 1.0, None)
 (qresult, probe_units) = scope.SCPI.CHANnel.PROBe.Query(1)
 print "Channel 1 probe attenuation factor: %s" % qresult

 # Use auto-scale to automatically set up oscilloscope.
 print "Autoscale."
 scope.SCPI.AUToscale.Command()

 # Set trigger mode.
 scope.SCPI.TRIGger.MODE.Command("EDGE")
 qresult = scope.SCPI.TRIGger.MODE.Query()
 print "Trigger mode: %s" % qresult

 # Set EDGE trigger parameters.
 scope.SCPI.TRIGger.EDGE.SOURce.Command(None, "CHANnel1")
 qresult = scope.SCPI.TRIGger.EDGE.SOURce.Query(None,)
 print "Trigger edge source: %s" % qresult

 scope.SCPI.TRIGger.LEVel.Command("CHANnel1", -0.002)
 qresult = scope.SCPI.TRIGger.LEVel.Query("CHANnel1")
 print "Trigger edge level: %s" % qresult

 scope.SCPI.TRIGger.EDGE.SLOPe.Command(None, "POSitive")
 qresult = scope.SCPI.TRIGger.EDGE.SLOPe.Query(None,)
 print "Trigger edge slope: %s" % qresult

 # Save oscilloscope setup.
 setup_bytes = scope.SCPI.SYSTem.SETup.Query()
 nLength = len(setup_bytes)
 File.WriteAllBytes("setup.stp", setup_bytes)
 print "Setup lines saved: %d" % nLength

 # Change oscilloscope settings with individual commands:

 # Set vertical scale and offset.
 scope.SCPI.CHANnel.SCALe.Command(1, 0.1)
 qresult = scope.SCPI.CHANnel.SCALe.Query(1)
 print "Channel 1 vertical scale: %f" % qresult

 scope.SCPI.CHANnel.OFFSet.Command(1, 0.0)
 qresult = scope.SCPI.CHANnel.OFFSet.Query(1)
 print "Channel 1 offset: %f" % qresult
```

```
 # Set horizontal scale and offset.
 scope.SCPI.TIMebase.SCALe.Command(0.0002)
 qresult = scope.SCPI.TIMebase.SCALe.Query()
 print "Timebase scale: %f" % qresult

 scope.SCPI.TIMebase.POSition.Command(0.0)
 qresult = scope.SCPI.TIMebase.POSition.Query()
 print "Timebase position: %f" % qresult

 # Set the acquisition mode.
 scope.SCPI.ACQuire.MODE.Command("RTIMe")
 qresult = scope.SCPI.ACQuire.MODE.Query()
 print "Acquire mode: %s" % qresult

 # Or, set up oscilloscope by loading a previously saved setup.
 setup_lines = File.ReadAllBytes("setup.stp")
 scope.SCPI.SYSTem.SETup.Command(setup_lines)
 print "Setup lines restored: %d" % len(setup_lines)

 # Set the desired number of waveform points,
 # and capture an acquisition.
 scope.SCPI.ACQuire.POINts.Command(32000)
 scope.SCPI.DIGitize.Command(None)


# ============================================================
# Analyze:
# ============================================================
def analyze():

 # Make measurements.
 # ------------------------------------------------------------
 scope.SCPI.MEASure.SOURce.Command("CHANnel1", None)
 source_list = scope.SCPI.MEASure.SOURce.Queryx()
 print "Measure source: %s" % source_list[0]

 scope.SCPI.MEASure.FREQuency.Command("CHANnel1", None)
 qresult = scope.SCPI.MEASure.FREQuency.QuerySendValidOff("CHANnel1", No
ne)
 print "Measured frequency on channel 1: %f" % qresult

 # Use direct command/query when commands not in command set.
 scope.Transport.Command.Invoke(":MEASure:VAMPlitude CHANnel1")
 qresult = scope.Transport.Query.Invoke(":MEASure:VAMPlitude? CHANnel1")
 print "Measured vertical amplitude on channel 1: %s" % qresult

 # Download the screen image.
 # ------------------------------------------------------------
 image_bytes = scope.SCPI.DISPlay.DATA.Query("PNG", None, None, None)
 nLength = len(image_bytes)
 fStream = File.Open("screen_image.png", FileMode.Create)
 fStream.Write(image_bytes, 0, nLength)
 fStream.Close()
 print "Screen image written to screen_image.png."

 # Download waveform data.
```

```
# ------------------------------------------------------

# Get the waveform points mode.
qresult = scope.SCPI.WAVeform.TYPE.Query()
print "Waveform points mode: %s" % qresult

# Get the number of waveform points.
qresult = scope.SCPI.WAVeform.POINts.Query()
print "Waveform points: %s" % qresult

# Set the waveform source.
scope.SCPI.WAVeform.SOURce.Command("CHANnel1")
qresult = scope.SCPI.WAVeform.SOURce.Query()
print "Waveform source: %s" % qresult

# Choose the format of the data returned:
scope.SCPI.WAVeform.FORMat.Command("WORD")
qresult = scope.SCPI.WAVeform.FORMat.Query()
print "Waveform format: %s" % qresult

scope.SCPI.WAVeform.BYTeorder.Command("LSBFirst")
qresult = scope.SCPI.WAVeform.BYTeorder.Query()
print "Waveform byte order: %s" % qresult

# Display the waveform settings from preamble:
wav_form_dict = {
 0 : "ASCii",
 1 : "BYTE",
 2 : "WORD",
 3 : "LONG",
 4 : "LONGLONG",
}
acq_type_dict = {
 1 : "RAW",
 2 : "AVERage",
 3 : "VHIStogram",
 4 : "HHIStogram",
 6 : "INTerpolate",
 10 : "PDETect",
}
acq_mode_dict = {
 0 : "RTIMe",
 1 : "ETIMe",
 3 : "PDETect",
}
coupling_dict = {
 0 : "AC",
 1 : "DC",
 2 : "DCFIFTY",
 3 : "LFREJECT",
}
units_dict = {
 0 : "UNKNOWN",
 1 : "VOLT",
 2 : "SECOND",
 3 : "CONSTANT",
 4 : "AMP",
```

```
 5 : "DECIBEL",
}

preamble_string = scope.SCPI.WAVeform.PREamble.Query()
(
 wav_form, acq_type, wfmpts, avgcnt, x_increment, x_origin,
 x_reference, y_increment, y_origin, y_reference, coupling,
 x_display_range, x_display_origin, y_display_range,
 y_display_origin, date, time, frame_model, acq_mode,
 completion, x_units, y_units, max_bw_limit, min_bw_limit
) = string.split(preamble_string, ",")

print "Waveform format: %s" % wav_form_dict[int(wav_form)]
print "Acquire type: %s" % acq_type_dict[int(acq_type)]
print "Waveform points desired: %s" % wfmpts
print "Waveform average count: %s" % avgcnt
print "Waveform X increment: %s" % x_increment
print "Waveform X origin: %s" % x_origin
print "Waveform X reference: %s" % x_reference   # Always 0.
print "Waveform Y increment: %s" % y_increment
print "Waveform Y origin: %s" % y_origin
print "Waveform Y reference: %s" % y_reference   # Always 0.
print "Coupling: %s" % coupling_dict[int(coupling)]
print "Waveform X display range: %s" % x_display_range
print "Waveform X display origin: %s" % x_display_origin
print "Waveform Y display range: %s" % y_display_range
print "Waveform Y display origin: %s" % y_display_origin
print "Date: %s" % date
print "Time: %s" % time
print "Frame model #: %s" % frame_model
print "Acquire mode: %s" % acq_mode_dict[int(acq_mode)]
print "Completion pct: %s" % completion
print "Waveform X units: %s" % units_dict[int(x_units)]
print "Waveform Y units: %s" % units_dict[int(y_units)]
print "Max BW limit: %s" % max_bw_limit
print "Min BW limit: %s" % min_bw_limit

# Get numeric values for later calculations.
x_increment = scope.SCPI.WAVeform.XINCrement.Query()
x_origin = scope.SCPI.WAVeform.XORigin.Query()
y_increment = scope.SCPI.WAVeform.YINCrement.Query()
y_origin = scope.SCPI.WAVeform.YORigin.Query()

# Get the waveform data.
scope.SCPI.WAVeform.STReaming.Command(False)
data_words = scope.SCPI.WAVeform.DATA.QueryBlockInt16(None, None)
nLength = len(data_words)
print "Number of data values: %d" % nLength

# Open file for output.
strPath = "waveform_data.csv"
writer = File.CreateText(strPath)

# Output waveform data in CSV format.
for i in xrange(0, nLength - 1):
 time_val = x_origin + i * x_increment
 voltage = data_words[i] * y_increment + y_origin
```

```
  writer.WriteLine("%E, %f" % (time_val, voltage))

 # Close output file.
 writer.Close()
 print "Waveform format WORD data written to %s." % strPath


# ============================================================
# Main program:
# ============================================================
#addr = "labi-trex-lp11.cos.agilent.com"
addr = "TCPIP0::labi-trex-lp11.cos.agilent.com::inst0::INSTR"
scope = AgInfiniium90000(addr)
scope.Transport.DefaultTimeout.Set(10000)

# Initialize the oscilloscope, capture data, and analyze.
initialize()
capture()
analyze()

print "End of program."

# Wait for a key press before exiting.
print "Press any key to exit..."
Console.ReadKey(True)
```

39 Reference

HDF5 Example 1204
CSV and TSV Header Format 1205
BIN Header Format 1207

HDF5 Example

Here is an example of a captured HDF5 file.

```
Channel 1(6576)
Group Size = 1
Number of Attributes = 17
Waveform Type = 1
Start = 1
NumPoints = 1000000
NumSegments = 0
Count = 1
XDispRange = 1.0E-6
XDispOrigin = -5.0E-7
XInc = 5.0E-11
XOrg = -2.4999999E-5
XUnits = Second
YDispRange = 8.0
YDispOrigin = 0.0
YInc = 1.327218738E-4
YOrg = 0.11645629362732
YUnits = Volt
MinBandwidth = 0.0
MaxBandwidth = 6.0E9
```

CSV and TSV Header Format

Revision Always 0 (zero).

Type How the waveform was acquired: normal, raw, interpolate, average, or versus. When this field is read back into the scope, all modes, except versus, are converted to raw. The default value is raw.

Start Starting point in the waveform of the first data point in the file. This is usually zero.

Points The number of points in the waveform record. The number of points is set by the Memory Depth control. The default value is 1.

Count or Segments For count, it is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. Count is ignored when it is read back into the scope. The default value is 0.

Segments is used instead of Count when the data is acquired using the Segmented acquisition mode. This number is the total number of segments that were acquired.

XDispRange The number of X display range columns (n) depends on the number of sources being stored. The X display range is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.

XDispOrg The number of X display origin columns (n) depends on the number of sources being stored. The X display origin is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

XInc The number of X increment columns (n) depends on the number of sources being store. The X increment is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.

XOrg The number of X origin columns (n) depends on the number of sources being store. The X origin is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

XUnits The number of X units columns (n) depends on the number of sources being store. The X units is the unit of measure for each time value of the acquired data.

YDispRange The number of Y display range columns (n) depends on the number of sources being store. The Y display range is the Y-axis duration of the waveform which is displayed. For voltage waveforms, it is the amount of voltage across the display. If the value is zero then no data has been acquired.

YDispOrg The number of Y display origin columns (n) depends on the number of sources being store. The Y-display origin is the Y-axis value at the center of the display. For voltage waveforms, it is the voltage at the center of the display. If the value is zero then no data has been acquired.

YInc The number of Y increment columns (n) depends on the number of sources being store. The Y increment is the duration between Y-axis levels. For voltage waveforms, it is the voltage corresponding to one level. If the value is zero then no data has been acquired.

YOrg The number of Y origin columns (n) depends on the number of sources being store. The Y origin is the Y-axis value at level zero. For voltage waveforms, it is the voltage at level zero. If the value is zero then no data has been acquired.

YUnits The number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform.

Frame A string containing the model number and serial number of the scope in the format of MODEL#:SERIAL#.

Date The date when the waveform was acquired. The default value is 27 DEC 1996.

Time The time when the waveform was acquired. The default value is 01:00:00:00.

Max bandwidth An estimation of the maximum bandwidth of the waveform. The default value is 0.

Min bandwidth An estimation of the minimum bandwidth of the waveform. The default value is 0.

Time Tags The Time Tags only occur when the data was acquired using the Segmented acquisition mode with time tags enabled and the file format is YValues. The number of columns depends on the number of Segments being saved.

Data The data values follow this header entry.

BIN Header Format

- "File Header" on page 1207
- "Waveform Header" on page 1207
- "Waveform Data Header" on page 1209
- "Example Program for Reading Binary Data" on page 1210

File Header

There is only one file header in a binary file. The file header consists of the following information.

Cookie Two byte characters, AG, which indicates that the file is in the Agilent Binary Data file format.

Version Two bytes which represent the file version.

File Size An integer (4 byte signed) which is the number of bytes that are in the file.

Number of Waveforms An integer (4 byte signed) which is the number of waveforms that are stored in the file.

Waveform Header

The waveform header contains information about the type of waveform data that is stored following the waveform data header which is located after each waveform header. Because it is possible to store more than one waveform in the file, there will be a waveform header and a waveform data header for each waveform.

Header Size An integer (4 byte signed) which is the number of bytes in the header.

Waveform Type An integer (4 byte signed) which is the type of waveform that is stored in the file. The follow shows what each value means.

0 = Unknown

1 = Normal

2 = Peak Detect

3 = Average

4 = Horizontal Histogram

5 = Vertical Histogram

6 = Logic

Number of Waveform Buffers An integer (4 byte signed) which is the number of waveform buffers required to read the data. This value is one except for peak detect data and digital data.

Count An integer (4 byte signed) which is the number of hits at each time bucket in the waveform record when the waveform was created using an acquisition mode like averaging. For example, when averaging, a count of four would mean every waveform data point in the waveform record has been averaged at least four times. The default value is 0.

X Display Range A float (4 bytes) which is the X-axis duration of the waveform that is displayed. For time domain waveforms, it is the duration of time across the display. If the value is zero then no data has been acquired.

X Display Origin A double (8 bytes) which is the X-axis value at the left edge of the display. For time domain waveforms, it is the time at the start of the display. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

X Increment A double (8 bytes) which is the duration between data points on the X axis. For time domain waveforms, this is the time between points. If the value is zero then no data has been acquired.

X Origin A double (8 bytes) which is the X-axis value of the first data point in the data record. For time domain waveforms, it is the time of the first point. This value is treated as a double precision 64-bit floating point number. If the value is zero then no data has been acquired.

X Units An integer (4 byte signed) which is the number of X units columns (n) depends on the number of sources being stored. The X units is the unit of measure for each time value of the acquired data. X unit definitions are:

0 = Unknown

1 = Volt

2 = Second

3 = Constant

4 = Amp

5 = Decibel

Y Units An integer (4 byte signed) which is the number of Y units columns (n) depends on the number of sources being stored. The Y units is the unit of measure of each voltage value of the acquired waveform. Y units definitions are:

0 = Unknown

1 = Volt

2 = Second

3 = Constant

4 = Amp

5 = Decibel

Date A 16 character array which is the date when the waveform was acquired. The default value is 27 DEC 1996.

Time A 16 character array which is the time when the waveform was acquired. The default value is 01:00:00:00.

Frame A 24 character array which is the model number and serial number of the scope in the format of MODEL#:SERIAL#.

Waveform Label A 16 character array which is the waveform label.

Time Tags A double (8 bytes) which is the time tag value of the segment being saved.

Segment Index An unsigned integer (4 byte signed) which is the segment index of the data that follows the waveform data header.

Waveform Data Header

The waveform data header consists of information about the waveform data points that are stored immediately after the waveform data header.

Waveform Data Header Size An integer (4 byte signed) which is the size of the waveform data header.

Buffer Type A short (2 byte signed) which is the type of waveform data that is stored in the file. The following shows what each value means.

0 = Unknown data

1 = Normal 32 bit float data

2 = Maximum float data

3 = Minimum float data

4 = Time float data

5 = Counts 32 bit float data

6 = Digital unsigned 8 bit char data

Bytes Per Point A short (2 byte signed) which is the number of bytes per data point.

Buffer Size An integer (4 byte signed) which is the size of the buffer required to hold the data bytes.

Example Program for Reading Binary Data

The following is a programming example of reading a Binary Data (.bin) file and converting it to a CSV (.csv) file without a file header.

```
/* bintoascii.c */

/* Reads the binary file format.
   This program demonstrates how to import the Infiniium
   oscilloscope binary file format and how to export it to an
   ascii comma separated file format.
*/
#pragma pack(4)

#include <stdio.h>               /* location of: printf() */
#include <stdlib.h>      /* location of: atof(), atoi() */
#include <string.h>     /* location of: strlen() */
#include "sicl.h"

/* Defines */
#define MAX_LENGTH 10000000
#define INTERFACE "lan[130.29.70.247]:inst0" /* Change the IP address
                                              * to the one found in
                                              * the Remote Setup
                                              * dialog box.
                                              */
#define TRUE 1
#define FALSE 0
#define IO_TIMEOUT 20000

/* Type definitions */
typedef unsigned _int64 UINT64; /* This defines a 64-bit unsigned
                                 * integer for Microsoft platforms.
                                 */

/* Structure and Union definitions */
union DATATYPE
{
   char buffer[MAX_LENGTH];  /* Buffer for reading word format data */
   char byte[MAX_LENGTH];
   unsigned short word[MAX_LENGTH/2];
   UINT64 longlong[MAX_LENGTH/4];
};


typedef struct
{
   char Cookie[2];
   char Version[2];
   int  FileSize;
   int  NumberOfWaveforms;
} FileHeader;

const char COOKIE[2] = {'A', 'G'};
const char VERSION[2] = {'1', '0'};
```

```
#define DATE_TIME_STRING_LENGTH 16
#define FRAME_STRING_LENGTH  24
#define SIGNAL_STRING_LENGTH 16

typedef struct
{
   int    HeaderSize;
   int    WaveformType;
   int    NWaveformBuffers;
   int    Points;
   int    Count;
   float  XDisplayRange;
   double XDisplayOrigin;
   double XIncrement;
   double XOrigin;
   int    XUnits;
   int    YUnits;
   char   Date[DATE_TIME_STRING_LENGTH];
   char   Time[DATE_TIME_STRING_LENGTH];
   char   Frame[FRAME_STRING_LENGTH];
   char   WaveformLabel[SIGNAL_STRING_LENGTH];
   double TimeTag;
   unsigned int SegmentIndex;
} WaveformHeader;

typedef struct
{
   int   HeaderSize;
   short BufferType;
   short BytesPerPoint;
   int   BufferSize;
} WaveformDataHeader;

typedef enum
{
   PB_UNKNOWN,
   PB_NORMAL,
   PB_PEAK_DETECT,
   PB_AVERAGE,
   PB_HORZ_HISTOGRAM,
   PB_VERT_HISTOGRAM,
   PB_LOGIC
} WaveformType;

typedef enum
{
   PB_DATA_UNKNOWN,
   PB_DATA_NORMAL,
   PB_DATA_MAX,
   PB_DATA_MIN,
   PB_DATA_TIME,
   PB_DATA_COUNTS,
   PB_DATA_LOGIC
} DataType;

/* Prototypes */
void GetTimeConversionFactors( WaveformHeader waveformHeader,
```

```
                                double *xInc, double *xOrg );
void OutputNormalWaveform( WaveformHeader waveformHeader );
void OutputPeakDetectWaveform( WaveformHeader waveformHeader );
void OutputHistogramWaveform( WaveformHeader waveformHeader );
void OutputData( FILE *PeakFile,
                 WaveformDataHeader waveformDataHeader );

/* Globals */
double xOrg=0L, xInc=0L;     /* Values necessary to create time data */
union DATATYPE WaveFormData; /* Used to input and output data */
FILE *InputFile = NULL;
FILE *OutputFile;
errno_t err;
char *buffer;
float Volts[MAX_LENGTH];
float MaxVolts[MAX_LENGTH];
float MinVolts[MAX_LENGTH];
UINT64 HistogramData[MAX_LENGTH];

int main( int argc, char **argv )
{
   FileHeader fileHeader;
   WaveformHeader waveformHeader;

   if( argc > 1 )
   {
      InputFile = fopen( argv[1], "rb" );

      if( InputFile )
      {
         OutputFile = fopen( argv[2], "wb" );

         if( OutputFile )
         {
            /* Read the File Header */
            fread( &fileHeader, 1, sizeof( FileHeader ), InputFile );

            /* Make sure that this is an Agilent Binary File */
            if( (fileHeader.Cookie[0] == COOKIE[0]) &&
                (fileHeader.Cookie[1] == COOKIE[1]) )
            {
               fread( &waveformHeader, 1,
                      sizeof( WaveformHeader ), InputFile );

               switch( waveformHeader.WaveformType )
               {
                  case PB_NORMAL:
                  case PB_AVERAGE:
                     OutputNormalWaveform( waveformHeader );
                     break;
                  case PB_PEAK_DETECT:
                     OutputPeakDetectWaveform( waveformHeader );
                     break;
                  case PB_HORZ_HISTOGRAM:
                  case PB_VERT_HISTOGRAM:
                     OutputHistogramWaveform( waveformHeader );
                     break;
```

```
                default:
                case PB_UNKNOWN:
                   printf( "Unknown waveform type: %d\n" );
                   break;
             }
          }
       }
       else
       {
          printf( "Unable to open output file %s\n", OutputFile);
       }
    }
    else
    {
       printf( "Unable to open input file %s\n", argv[1] );
    }

    fclose( InputFile );
    fclose( OutputFile );
 }
 else
 {
    printf( "Usage: bintoascii inputfile outputfile\n" );
 }

}

/*****************************************************************
 * Function name:   GetTimeConversionFactors
 *  Parameters:     double xInc which is the time between consecutive
 *                     sample points.
 *                  double xOrg which is the time value of the first
 *                     data point.
 *   Return value: none
 *   Description: This routine transfers the waveform conversion
 *                factors for the time values.
*****************************************************************/
void GetTimeConversionFactors( WaveformHeader waveformHeader,
                               double *xInc, double *xOrg )
{

   /* Read values which are used to create time values */

   *xInc = waveformHeader.XIncrement;
   *xOrg = waveformHeader.XOrigin;

}

/*****************************************************************
 * Function name:  OutputNormalWaveform
 *  Parameters:    WaveformHeader *waveformHeader which is a structure
 *                 that contains the waveform header information.
 *   Return value: none
 *   Description: This routine stores the time and voltage information
 *                about the waveform as time and voltage separated by
 *                commas to a file.
*****************************************************************/
```

```
void OutputNormalWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;
   double Time;

   BytesRead = fread( &waveformDataHeader, 1,
                      sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, MAX_LENGTH, InputFile );
      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
      {
         Time = (j * xInc) + xOrg; /* calculate time */
         j = j + 1;
         fprintf( OutputFile, "%e,%f\n", Time, Volts[i] );
      }
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
   }
}

/*********************************************************************
 * Function name: OutputHistogramWaveform
 *  Parameters:   WaveformHeader *waveformHeader which is a structure
 *                   that contains the waveform header information.
 *  Return value: none
 *  Description:  This routine stores the time and hits information
 *                as time and hits separated by commas to a file.
*********************************************************************/
void OutputHistogramWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
   size_t BytesRead = 0L;

   fread( &waveformDataHeader, 1,
          sizeof( WaveformDataHeader ), InputFile );
   GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );
   while( !done )
   {
      BytesRead = fread( (char *) HistogramData, 1, MAX_LENGTH,
                         InputFile );

      for( i = 0; i < (BytesRead/waveformDataHeader.BytesPerPoint); i++)
       {
         fprintf( OutputFile, "%d,%u64l\n", j, HistogramData[i] );
         j = j + 1;
       }
```

```
      if( BytesRead < MAX_LENGTH )
      {
         done = TRUE;
      }
   }
}

/*********************************************************************
 * Function name: OutputData
 *  Parameters:   FILE *PeakFile which is the pointer to the file
 *                   to be written.
 *                WaveformDataHeader waveformDataHeader
 *                   which is a structure that contains the waveform
 *                   header information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputData( FILE *PeakFile, WaveformDataHeader waveformDataHeader )
{
   int done = FALSE;
   size_t BytesRead = 0L;
   int NumberToRead;

   NumberToRead = waveformDataHeader.BufferSize;

   while( !done )
   {
      BytesRead = fread( (char *) Volts, 1, NumberToRead, InputFile ) +
                  BytesRead;

      fwrite( Volts, 1, BytesRead, PeakFile );

      if( BytesRead <= NumberToRead )
      {
         done = TRUE;
      }
   }
}

/*********************************************************************
 * Function name: OutputPeakDetectWaveform
 *  Parameters:   WaveformHeader waveformHeader which is a
 *                structure that contains the waveform header
 *                information.
 *  Return value: none
 *  Description: This routine stores the time, minimum voltage, and
 *               maximum voltage for the peak detect waveform as comma
 *               separated values to a file.
*********************************************************************/
void OutputPeakDetectWaveform( WaveformHeader waveformHeader )
{
   WaveformDataHeader waveformDataHeader;
   int done = FALSE;
   unsigned long i;
   unsigned long j = 0;
```

```
    size_t BytesRead = 0L;
    double Time;
    FILE *MaxFile;
    FILE *MinFile;

    fread( &waveformDataHeader, 1,
           sizeof( WaveformDataHeader ), InputFile );
    GetTimeConversionFactors( waveformHeader, &xInc, &xOrg );

    MaxFile = fopen( "maxdata.bin", "wb" );
    MinFile = fopen( "mindata.bin", "wb" );

    if( MaxFile && MinFile )
    {
       if( waveformDataHeader.BufferType == PB_DATA_MAX )
       {
          OutputData( MaxFile, waveformDataHeader );
          OutputData( MinFile, waveformDataHeader );
       }
       else
       {
          OutputData( MinFile, waveformDataHeader );
          OutputData( MaxFile, waveformDataHeader );
       }

       fclose( MaxFile );
       fclose( MinFile );

       MaxFile = fopen( "maxdata.bin", "rb" );
       MinFile = fopen( "mindata.bin", "rb" );

       while( !done )
       {
          BytesRead = fread( (char *) MaxVolts, 1, MAX_LENGTH, MaxFile );
          fread( (char *) MinVolts, 1, MAX_LENGTH, MinFile );

          for( i = 0; i < BytesRead/4; i++)
          {
             Time = (j * xInc) + xOrg; /* calculate time */
             j = j + 1;
             fprintf( OutputFile, "%9.5e,%f,%f\n", Time, MinVolts[i],
                      MaxVolts[i] );
          }

          if( BytesRead < MAX_LENGTH )
          {
             done = TRUE;
          }
       }

       fclose( MaxFile );
       fclose( MinFile );
    }
}
```

Index

Symbols

:ACQuire:AVERage command/query, 150
:ACQuire:AVERage:COUNt command/query, 151
:ACQuire:BANDwidth, command/query, 152
:ACQuire:COMPlete command/query, 154
:ACQuire:COMPlete:STATe command/query, 156
:ACQuire:HRESolution command/query, 157
:ACQuire:INTerpolate command/query, 158
:ACQuire:MODE command/query, 159
:ACQuire:POINts:ANALog command/query, 161
:ACQuire:POINts:AUTO command/query, 165
:ACQuire:POINts:DIGital? query, 166
:ACQuire:REDGe command/query, 167
:ACQuire:RESPonse command/query, 168
:ACQuire:SEGMented:COUNt command/query, 169
:ACQuire:SEGMented:INDex command/query, 170
:ACQuire:SEGMented:TTAGs command/query, 171
:ACQuire:SRATe:ANALog command/query, 172
:ACQuire:SRATe:ANALog:AUTO command/query, 174
:ACQuire:SRATe:DIGital command/query, 175
:ACQuire:SRATe:DIGital:AUTO command/query, 176
:ADER? query, 749
:AER? query, 750
:ASTate? query, 751
:ATER? query, 752
:AUToscale command, 753
:AUToscale:CHANnels command, 754
:AUToscale:PLACement command/query, 755
:AUToscale:VERTical command, 756
:BEEP command, 757
:BLANk command, 758
:BUS:B<N>:TYPE command/query, 178
:BUS<N>:BIT<M> command/query, 179
:BUS<N>:BITS command/query, 180
:BUS<N>:CLEar command/query, 181
:BUS<N>:CLOCk command/query, 182
:BUS<N>:CLOCk:SLOPe command/query, 183
:BUS<N>:DISPlay command/query, 184
:BUS<N>:LABel command/query, 185
:BUS<N>:READout command/query, 186
:CALibrate:DATE? query, 189
:CALibrate:OUTPut command/query, 190
:CALibrate:SKEW command/query, 191
:CALibrate:STATus? query, 192
:CALibrate:TEMP? query, 193
:CDISplay command, 759
:CHANnel<N>:COMMonmode command/query, 197
:CHANnel<N>:DIFFerential command/query, 198
:CHANnel<N>:DIFFerential:SKEW command/query, 199
:CHANnel<N>:DISPlay command/query, 200
:CHANnel<N>:DISPlay:AUTO command/query, 201
:CHANnel<N>:DISPlay:OFFSet command/query, 202
:CHANnel<N>:DISPlay:RANGe command/query, 203
:CHANnel<N>:DISPlay:SCALe command/query, 204
:CHANnel<N>:INPut command/query, 205
:CHANnel<N>:ISIM:APPLy command/query, 206
:CHANnel<N>:ISIM:BANDwidth command/query, 207
:CHANnel<N>:ISIM:BWLimit command/query, 208
:CHANnel<N>:ISIM:CONVolve command/query, 209
:CHANnel<N>:ISIM:CORRection command/query, 210
:CHANnel<N>:ISIM:DEConvolve command/query, 212
:CHANnel<N>:ISIM:DELay command/query, 213
:CHANnel<N>:ISIM:NORMalize command/query, 215
:CHANnel<N>:ISIM:PEXTraction command/query, 216
:CHANnel<N>:ISIM:SPAN command/query, 218
:CHANnel<N>:ISIM:STATe command/query, 219
:CHANnel<N>:LABel command/query, 220
:CHANnel<N>:OFFSet command/query, 221
:CHANnel<N>:PROBe command/query, 222
:CHANnel<N>:PROBe:ACCAL command/query, 223
:CHANnel<N>:PROBe:ATTenuation command/query, 224
:CHANnel<N>:PROBe:AUTozero command/query, 225
:CHANnel<N>:PROBe:COUPling command/query, 226
:CHANnel<N>:PROBe:EADapter command/query, 227
:CHANnel<N>:PROBe:ECOupling command/query, 229
:CHANnel<N>:PROBe:EXTernal command/query, 230
:CHANnel<N>:PROBe:EXTernal:GAIN command/query, 231
:CHANnel<N>:PROBe:EXTernal:OFFSet command/query, 232
:CHANnel<N>:PROBe:EXTernal:UNITs command/query, 233
:CHANnel<N>:PROBe:GAIN command/query, 234
:CHANnel<N>:PROBe:HEAD:ADD command, 235
:CHANnel<N>:PROBe:HEAD:DELete command, 236
:CHANnel<N>:PROBe:HEAD:SELect command/query, 237
:CHANnel<N>:PROBe:HEAD:VTERm command/query, 238
:CHANnel<N>:PROBe:ID? query, 239
:CHANnel<N>:PROBe:MODE command/query, 240
:CHANnel<N>:PROBe:PRECprobe:BANDwidth command, 241
:CHANnel<N>:PROBe:PRECprobe:CALibration command, 242
:CHANnel<N>:PROBe:PRECprobe:MODE command, 243
:CHANnel<N>:PROBe:PRECprobe:ZSRC command, 244
:CHANnel<N>:PROBe:SKEW command/query, 246
:CHANnel<N>:PROBe:STYPe command/query, 247
:CHANnel<N>:RANGe command/query, 248
:CHANnel<N>:SCALe command/query, 249
:CHANnel<N>:UNITs command/query, 250
:DIGital<N>:DISPlay command/query, 276
:DIGital<N>:LABel command/query, 277
:DIGital<N>:SIZE command/query, 278
:DIGital<N>:THReshold command/query, 279
:DIGitize command, 760
:DISable DIGital command, 761
:DISK:CDIRectory command, 282
:DISK:COPY command, 283
:DISK:DELete command, 284
:DISK:DIRectory? query, 285
:DISK:LOAD command, 286
:DISK:MDIRectory command, 287
:DISK:PWD? query, 288
:DISK:SAVE:COMPosite command, 289
:DISK:SAVE:IMAGe command, 290
:DISK:SAVE:JITTer command, 291

:DISK:SAVE:LISTing command, 292
:DISK:SAVE:MEASurements command, 293
:DISK:SAVE:PRECprobe command, 294
:DISK:SAVE:SETup command, 295
:DISK:SAVE:WAVeform command, 296
:DISK:SEGMented command/query, 298
:DISK:STORe command, 1053
:DISPlay: ROW command query, 316
:DISPlay:CGRade command/query, 300
:DISPlay:CGRade:LEVels? query, 301
:DISPlay:CGRade:SCHeme command/query, 303
:DISPlay:COLumn command/query, 305
:DISPlay:CONNect command/query, 306
:DISPlay:DATA? query, 307
:DISPlay:GRATicule command/query, 308
:DISPlay:GRATicule:INTensity command/query, 309
:DISPlay:GRATicule:NUMBer command/query, 310
:DISPlay:GRATicule:SETGrat command, 311
:DISPlay:GRATicule:SIZE command/query, 312
:DISPlay:LABel command/query, 313
:DISPlay:LINE command, 314
:DISPlay:PERSistence command/query, 315
:DISPlay:SCOLor command/query, 317
:DISPlay:STATus:COL command query, 319
:DISPlay:STATus:ROW command query, 320
:DISPlay:STRing command, 321
:DISPlay:TAB command/query, 322
:DISPlay:TEXT command, 323
:ENABle DIGital command, 762
:FUNCtion<F>:ABSolute command, 329
:FUNCtion<F>:ADD command, 330
:FUNCtion<F>:ADEMod command, 331
:FUNCtion<F>:AVERage command, 332
:FUNCtion<F>:COMMonmode command, 333
:FUNCtion<F>:DELay command, 334
:FUNCtion<F>:DIFF command, 335
:FUNCtion<F>:DISPlay command/query, 336
:FUNCtion<F>:DIVide command, 337
:FUNCtion<F>:FFT:FREQuency command/query, 338
:FUNCtion<F>:FFT:REFerence command/query, 339
:FUNCtion<F>:FFT:RESolution? query, 340
:FUNCtion<F>:FFT:TDELay command/query, 341
:FUNCtion<F>:FFT:WINDow command/query, 342
:FUNCtion<F>:FFTMagnitude command, 344
:FUNCtion<F>:FFTPhase command, 345
:FUNCtion<F>:GATing command, 346
:FUNCtion<F>:HIGHpass command, 347
:FUNCtion<F>:HORizontal command/query, 348
:FUNCtion<F>:HORizontal:POSition command/query, 349
:FUNCtion<F>:HORizontal:RANGe command/query, 350
:FUNCtion<F>:INTegrate command, 351
:FUNCtion<F>:INVert command, 352
:FUNCtion<F>:LOWPass command, 353
:FUNCtion<F>:MAGNify command, 354
:FUNCtion<F>:MAXimum command, 355
:FUNCtion<F>:MHISTogram command, 356
:FUNCtion<F>:MINimum command, 357
:FUNCtion<F>:MTRend command, 358
:FUNCtion<F>:MULTiply command, 359
:FUNCtion<F>:OFFSet command/query, 360
:FUNCtion<F>:RANGe command/query, 361
:FUNCtion<F>:SMOoth command, 362
:FUNCtion<F>:SQRT command, 363
:FUNCtion<F>:SQUare command, 364
:FUNCtion<F>:SUBTract command, 365
:FUNCtion<F>:VERSus command, 366
:FUNCtion<F>:VERTical command/query, 367
:FUNCtion<F>:VERTical:OFFSet command/query, 368
:FUNCtion<F>:VERTical:RANGe command/query, 369
:FUNCtion<F>? query, 328
:HARDcopy:AREA command/query, 372
:HARDcopy:DPRinter command/query, 373
:HARDcopy:FACTors command/query, 374
:HARDcopy:IMAGe command/query, 375
:HARDcopy:PRINters? query, 376
:HISTogram:AXIS command/query, 379
:HISTogram:MODE command/query, 380
:HISTogram:SCALe:SIZE command/query, 381
:HISTogram:WINDow:BLIMit command/query, 386
:HISTogram:WINDow:DEFault command, 382
:HISTogram:WINDow:LLIMit command/query, 384
:HISTogram:WINDow:RLIMit command/query, 385
:HISTogram:WINDow:SOURce command/query, 383
:HISTogram:WINDow:TLIMit command/query, 387
:ISCan:DELay command/query, 390
:ISCan:MEASurement command/query, 393
:ISCan:MEASurement:FAIL command/query, 391
:ISCan:MEASurement:LLIMit command/query, 392
:ISCan:MEASurement:ULIMit command/query, 394
:ISCan:MODE command/query, 395
:ISCan:NONMonotonic:EDGE command/query, 396
:ISCan:NONMonotonic:HYSTeresis command/query, 397
:ISCan:NONMonotonic:SOURce command/query, 398
:ISCan:RUNT:HYSTeresis command/query, 399
:ISCan:RUNT:LLEVel command/query, 400
:ISCan:RUNT:SOURce command/query, 401
:ISCan:RUNT:ULEVel command/query, 402
:ISCan:SERial:PATTern command/query, 403
:ISCan:SERial:SOURce command/query, 404
:ISCan:ZONE:HIDE command/query, 405
:ISCan:ZONE:SOURce command/query, 406
:ISCan:ZONE<N>:MODE command/query, 407
:ISCan:ZONE<N>:PLACement command/query, 408
:ISCan:ZONE<N>:STATe command/query, 409
:LISTer:DATA? query, 420
:LISTer:DISPlay command/query, 421
:LTESt:FAIL command/query, 412
:LTESt:LLIMit command/query, 413
:LTESt:MEASurement command/query, 414
:LTESt:RESults? query, 415
:LTESt:TEST command/query, 416
:LTESt:ULIMit command/query, 417
:MARKer:CURSor? query, 424
:MARKer:MEASurement:MEASurement command, 425
:MARKer:MODE command/query, 426
:MARKer:X1Position command/query, 427
:MARKer:X1Y1source command/query, 429
:MARKer:X2Position command/query, 428
:MARKer:X2Y2source command/query, 430
:MARKer:XDELta? query, 431
:MARKer:Y1Position command/query, 432
:MARKer:Y2Position command/query, 433
:MARKer:YDELta? query, 434
:MEASure:AREA command/query, 488
:MEASure:BINTerval command/query, 489
:MEASure:BPERiod command/query, 490
:MEASure:BWIDth command/query, 491
:MEASure:CDRRATE command, 492
:MEASure:CGRade:CROSsing command/query, 493
:MEASure:CGRade:DCDistortion command/query, 494
:MEASure:CGRade:EHEight command/query, 495
:MEASure:CGRade:EWIDth command/query, 496
:MEASure:CGRade:EWINdow command/query, 497
:MEASure:CGRade:JITTer command/query, 498
:MEASure:CGRade:QFACtor command/query, 499
:MEASure:CLEar command, 500
:MEASure:CLOCk command/query, 501
:MEASure:CLOCk:METHod command/query, 502, 1054
:MEASure:CLOCk:METHod:ALIGn command/query, 504
:MEASure:CLOCk:METHod:DEEMphasis command/query, 505
:MEASure:CLOCk:METHod:EDGE command/query, 506
:MEASure:CLOCk:METHod:JTF command/query, 508
:MEASure:CLOCk:METHod:OJTF command/query, 510
:MEASure:CLOCk:VERTical command/query, 512
:MEASure:CLOCk:VERTical:OFFSet command/query, 513

:MEASure:CLOCk:VERTical:RANGe
command/query, 514
:MEASure:CROSsing command/query, 515
:MEASure:CTCDutycycle command/query, 516
:MEASure:CTCJitter command/query, 518
:MEASure:CTCNwidth command/query, 520
:MEASure:CTCPwidth command/query, 522
:MEASure:DATarate command/query, 524
:MEASure:DDPWS command/query, 35, 1057
:MEASure:DEEMphasis command/query, 526
:MEASure:DELTatime command/query, 528
:MEASure:DELTatime:DEFine
command/query, 530
:MEASure:DUTYcycle command/query, 532
:MEASure:EDGE command/query, 534
:MEASure:ETOedge command, 536
:MEASure:FALLtime command/query, 537
:MEASure:FFT:DFRequency
command/query, 539
:MEASure:FFT:DMAGnitude
command/query, 541
:MEASure:FFT:FREQuency
command/query, 543
:MEASure:FFT:MAGNitude
command/query, 544
:MEASure:FFT:PEAK1 command/query, 545
:MEASure:FFT:PEAK2 command/query, 546
:MEASure:FFT:THReshold
command/query, 547
:MEASure:FREQuency command/query, 548
:MEASure:HISTogram:HITS
command/query, 550
:MEASure:HISTogram:M1S
command/query, 551
:MEASure:HISTogram:M2S
command/query, 552
:MEASure:HISTogram:M3S
command/query, 553
:MEASure:HISTogram:MAX
command/query, 554
:MEASure:HISTogram:MEAN
command/query, 555
:MEASure:HISTogram:MEDian
command/query, 556
:MEASure:HISTogram:MIN
command/query, 557
:MEASure:HISTogram:MODE
command/query, 558
:MEASure:HISTogram:PEAK
command/query, 559
:MEASure:HISTogram:PP command/query, 560
:MEASure:HISTogram:RESolution
command/query, 561
:MEASure:HISTogram:STDDev
command/query, 562
:MEASure:HOLDtime command/query, 563
:MEASure:JITTer:HISTogram
command/query, 565
:MEASure:JITTer:MEASurement
command/query, 566
:MEASure:JITTer:SPECtrum
command/query, 567
:MEASure:JITTer:SPECtrum:HORizontal
command/query, 568
:MEASure:JITTer:SPECtrum:HORizontal:POSition
command/query, 569
:MEASure:JITTer:SPECtrum:HORizontal:RANGe
command/query, 570
:MEASure:JITTer:SPECtrum:VERTical
command/query, 571
:MEASure:JITTer:SPECtrum:VERTical:OFFSet
command/query, 572
:MEASure:JITTer:SPECtrum:VERTical:RANGe
command/query, 573
:MEASure:JITTer:SPECtrum:VERTical:TYPE
command/query, 574
:MEASure:JITTer:SPECtrum:WINDow
command/query, 575
:MEASure:JITTer:STATistics
command/query, 576
:MEASure:JITTer:TRENd command/query, 577
:MEASure:JITTer:TRENd:SMOoth
command/query, 578
:MEASure:JITTer:TRENd:SMOoth:POINts
command/query, 579
:MEASure:JITTer:TRENd:VERTical
command/query, 580
:MEASure:JITTer:TRENd:VERTical:OFFSet
command/query, 581
:MEASure:JITTer:TRENd:VERTical:RANGe
command/query, 582
:MEASure:NAME command/query, 583
:MEASure:NCJitter command/query, 584
:MEASure:NOISe command/query, 586
:MEASure:NOISe:ALL? query, 587
:MEASure:NOISe:BANDwidth
command/query, 589
:MEASure:NOISe:LOCation
command/query, 590
:MEASure:NOISe:METHod
command/query, 591
:MEASure:NOISe:REPort command/query, 592
:MEASure:NOISe:RN command/query, 593
:MEASure:NOISe:SCOPe:RN
command/query, 594
:MEASure:NOISe:STATe command/query, 595
:MEASure:NOISe:UNITs command/query, 596
:MEASure:NPERiod command/query, 597
:MEASure:NPULses command/query, 598
:MEASure:NUI command/query, 599
:MEASure:NWIDth command/query, 600
:MEASure:OVERshoot command/query, 602
:MEASure:PAMPlitude command/query, 604
:MEASure:PBASe command/query, 605
:MEASure:PERiod command/query, 606
:MEASure:PHASe command/query, 608
:MEASure:PPULses command/query, 610
:MEASure:PREShoot command/query, 611
:MEASure:PTOP command/query, 613
:MEASure:PWIDth command/query, 614
:MEASure:QUALifier<M>:CONDition
command/query, 616
:MEASure:QUALifier<M>:SOURce
command/query, 617
:MEASure:QUALifier<M>:STATe
command/query, 618
:MEASure:RESults? query, 619
:MEASure:RISetime command/query, 622
:MEASure:RJDJ:ALL? query, 624
:MEASure:RJDJ:APLength? query, 626
:MEASure:RJDJ:BANDwidth
command/query, 627
:MEASure:RJDJ:BER command/query, 628
:MEASure:RJDJ:CLOCk command/query, 630
:MEASure:RJDJ:EDGE command/query, 631
:MEASure:RJDJ:INTerpolate
command/query, 632
:MEASure:RJDJ:METHod
command/query, 633
:MEASure:RJDJ:MODe command/query, 634
:MEASure:RJDJ:PLENgth
command/query, 635
:MEASure:RJDJ:REPort command/query, 636
:MEASure:RJDJ:RJ command/query, 637
:MEASure:RJDJ:SCOPe:RJ
command/query, 638
:MEASure:RJDJ:SOURce
command/query, 639
:MEASure:RJDJ:STATe command/query, 640
:MEASure:RJDJ:TJRJDJ? query, 641
:MEASure:RJDJ:UNITs command/query, 642
:MEASure:SCRatch command, 643
:MEASure:SENDvalid command/query, 644
:MEASure:SETuptime command/query, 645
:MEASure:SLEWrate command/query, 647
:MEASure:SOURce command/query, 648
:MEASure:STATistics command/query, 649
:MEASure:TEDGe command/query, 650
:MEASure:THResholds:ABSolute
command/query, 652
:MEASure:THResholds:GENeral:ABSolute
command/query, 654
:MEASure:THResholds:GENeral:HYSTeresis
command/query, 656
:MEASure:THResholds:GENeral:METHod
command/query, 658
:MEASure:THResholds:GENeral:PERCent
command/query, 660
:MEASure:THResholds:GENeral:TOPBase:ABSol
ute command/query, 662
:MEASure:THResholds:GENeral:TOPBase:METH
od command/query, 664
:MEASure:THResholds:HYSTeresis
command/query, 666
:MEASure:THResholds:METHod
command/query, 668
:MEASure:THResholds:PERCent
command/query, 670
:MEASure:THResholds:RFALl:ABSolute
command/query, 672
:MEASure:THResholds:RFALl:HYSTeresis
command/query, 674
:MEASure:THResholds:RFALl:METHod
command/query, 676
:MEASure:THResholds:RFALl:PERCent
command/query, 678

:MEASure:THResholds:RFALI:TOPBase:ABSolute command/query, 680
:MEASure:THResholds:RFALI:TOPBase:METHod command/query, 682
:MEASure:THResholds:SERial:ABSolute command/query, 684
:MEASure:THResholds:SERial:HYSTeresis command/query, 686
:MEASure:THResholds:SERial:METHod command/query, 688
:MEASure:THResholds:SERial:PERCent command/query, 690
:MEASure:THResholds:SERial:TOPBase:ABSolute command/query, 692
:MEASure:THResholds:SERial:TOPBase:METHod command/query, 694
:MEASure:THResholds:TOPBase:ABSolute command/query, 696
:MEASure:THResholds:TOPBase:METHod command/query, 698
:MEASure:TIEClock2 command/query, 700
:MEASure:TIEData command/query, 702
:MEASure:TIEFilter:SHAPe command/query, 704
:MEASure:TIEFilter:STARt command/query, 705
:MEASure:TIEFilter:STATe command/query, 706
:MEASure:TIEFilter:STOP command/query, 707
:MEASure:TIEFilter:TYPE command/query, 708
:MEASure:TMAX command/query, 709
:MEASure:TMIN command/query, 710
:MEASure:TVOLt command/query, 711
:MEASure:UITouijitter command/query, 713
:MEASure:UNITinterval command/query, 714
:MEASure:VAMPlitude command/query, 716
:MEASure:VAVerage command/query, 717
:MEASure:VBASe command/query, 719
:MEASure:VLOWer command/query, 720
:MEASure:VMAX command/query, 721
:MEASure:VMIDdle command/query, 723
:MEASure:VMIN command/query, 724
:MEASure:VOVershoot command/query, 726
:MEASure:VPP command/query, 727
:MEASure:VPReshoot command/query, 729
:MEASure:VRMS command/query, 730
:MEASure:VTIMe command/query, 732
:MEASure:VTOP command/query, 734
:MEASure:VUPPer command/query, 735
:MEASure:WINDow command/query, 737
:MEASurement<N>:NAME command/query, 738
:MEASurement<N>:SOURce command/query, 739
:MODel? query, 763
:MTEE command/query, 764
:MTER? query, 765
:MTESt:ALIGn command, 437
:MTESt:AlignFIT command/query, 438
:MTESt:AMASk:CREate command, 440
:MTESt:AMASk:SAVE command, 442
:MTESt:AMASk:SOURce command/query, 441
:MTESt:AMASk:UNITs command/query, 443
:MTESt:AMASk:XDELta command/query, 444
:MTESt:AMASk:YDELta command/query, 445
:MTESt:AUTO command/query, 446
:MTESt:AVERage command/query, 447
:MTESt:AVERage:COUNt command/query, 448
:MTESt:COUNt:FAILures? query, 449
:MTESt:COUNt:FUI? query, 450
:MTESt:COUNt:FWAVeforms? query, 451
:MTESt:COUNt:UI? query, 452
:MTESt:COUNt:WAVeforms? query, 453
:MTESt:DELete command, 454
:MTESt:ENABle command/query, 455
:MTESt:FOLDing command/query, 456
:MTESt:FOLDing:BITS command/query, 457
:MTESt:FOLDing:FAST command/query, 458
:MTESt:HAMPlitude command/query, 459
:MTESt:IMPedance command/query, 460
:MTESt:INVert command/query, 461
:MTESt:LAMPlitude command/query, 462
:MTESt:LOAD command, 463
:MTESt:NREGions? query, 464
:MTESt:PROBe:IMPedance? query, 465
:MTESt:RUMode command/query, 466
:MTESt:RUMode:SOFailure command/query, 467
:MTESt:SCALe:BIND command/query, 468
:MTESt:SCALe:X1 command/query, 469
:MTESt:SCALe:XDELta command/query, 470
:MTESt:SCALe:Y1 command/query, 471
:MTESt:SCALe:Y2 command/query, 472
:MTESt:SOURce command/query, 473
:MTESt:STARt command, 474
:MTESt:STIMe command/query, 476
:MTESt:STOP command, 475
:MTESt:TITLe? query, 477
:MTESt:TRIGger:SOURce command/query, 478
:OPEE command/query, 766
:OPER? query, 767
:OVLRegister? query, 768
:PDER? query, 769
:POD<N>:DISPlay command/query, 742
:POD<N>:PSKew command/query, 743
:POD<N>:THReshold command/query, 744
:PRINt command, 770
:RECall:SETup command, 771
:RSTate? query, 772
:RUN command, 773
:SBUS<N>:CAN:SAMPlepoint command/query, 788
:SBUS<N>:CAN:SIGNal:BAUDrate command/query, 789
:SBUS<N>:CAN:SIGNal:DEFinition command/query, 790
:SBUS<N>:CAN:SOURce command/query, 791
:SBUS<N>:IIC:ASIZe command/query, 793
:SBUS<N>:IIC:SOURce:CLOCk command/query, 794
:SBUS<N>:IIC:SOURce:DATA command/query, 795
:SBUS<N>:SPI:BITorder command/query, 797
:SBUS<N>:SPI:CLOCk:SLOPe command/query, 798
:SBUS<N>:SPI:CLOCk:TIMeout command/query, 799
:SBUS<N>:SPI:FRAMe:STATe command/query, 800
:SBUS<N>:SPI:SOURce:CLOCk command/query, 801
:SBUS<N>:SPI:SOURce:DATA command/query, 802
:SBUS<N>:SPI:SOURce:FRAMe command/query, 803
:SBUS<N>:SPI:SOURce:MISO command/query, 804
:SBUS<N>:SPI:SOURce:MOSI command/query, 805
:SBUS<N>:SPI:TYPE command/query, 806
:SBUS<N>:SPI:WIDTh command/query, 807
:SELFtest:CANCel command, 810
:SELFtest:SCOPETEST command/query, 811
:SERial command/query, 774
:SINGle command, 775
:SPRocessing:CTLequalizer:DCGain command/query, 817
:SPRocessing:CTLequalizer:DISPlay command/query, 815
:SPRocessing:CTLequalizer:NUMPoles command/query, 818
:SPRocessing:CTLequalizer:P1 command/query, 819
:SPRocessing:CTLequalizer:P2 command/query, 820
:SPRocessing:CTLequalizer:P3 command/query, 821
:SPRocessing:CTLequalizer:RATe command/query, 822
:SPRocessing:CTLequalizer:SOURce command/query, 816
:SPRocessing:CTLequalizer:VERTical command/query, 823
:SPRocessing:CTLequalizer:VERTical:OFFSet command/query, 824
:SPRocessing:CTLequalizer:VERTical:RANGe command/query, 825
:SPRocessing:CTLequalizer:ZERo command/query, 826
:SPRocessing:DFEQualizer:NTAPs command/query, 829
:SPRocessing:DFEQualizer:SOURce command/query, 828
:SPRocessing:DFEQualizer:STATe command/query, 827
:SPRocessing:DFEQualizer:TAP command/query, 830
:SPRocessing:DFEQualizer:TAP:AUTomatic command, 838
:SPRocessing:DFEQualizer:TAP:DELay command/query, 832
:SPRocessing:DFEQualizer:TAP:GAIN command/query, 835
:SPRocessing:DFEQualizer:TAP:LTARget command/query, 837

:SPRocessing:DFEQualizer:TAP:MAX command/query, 833
:SPRocessing:DFEQualizer:TAP:MIN command/query, 834
:SPRocessing:DFEQualizer:TAP:UTARget command/query, 836
:SPRocessing:DFEQualizer:TAP:WIDTh command/query, 831
:SPRocessing:FFEequalizer:RATe command/query, 843
:SPRocessing:FFEQualizer:DISPlay command/query, 839
:SPRocessing:FFEQualizer:NPRecursor command/query, 841
:SPRocessing:FFEQualizer:NTAPs command/query, 842
:SPRocessing:FFEQualizer:SOURce command/query, 840
:SPRocessing:FFEQualizer:TAP command/query, 844
:SPRocessing:FFEQualizer:TAP:AUTomatic command, 848
:SPRocessing:FFEQualizer:TAP:BANDwidth command/query, 849
:SPRocessing:FFEQualizer:TAP:BWMode command/query, 850
:SPRocessing:FFEQualizer:TAP:DELay command/query, 847
:SPRocessing:FFEQualizer:TAP:PLENgth command/query, 845
:SPRocessing:FFEQualizer:TAP:TDELay command/query, 851
:SPRocessing:FFEQualizer:TAP:TDMode command/query, 852
:SPRocessing:FFEQualizer:TAP:WIDTh command/query, 846
:SPRocessing:FFEQualizer:VERTical command/query, 853
:SPRocessing:FFEQualizer:VERTical:OFFSet command/query, 854
:SPRocessing:FFEQualizer:VERTical:RANGe command/query, 855
:STATus? query, 776
:STOP command, 777
:STORe:JITTer command, 778
:STORe:SETup command, 779
:STORe:WAVeform command, 780
:SYSTem:DATE command/query, 858
:SYSTem:DEBug command/query, 859
:SYSTem:DSP command/query, 861
:SYSTem:ERRor? query, 862
:SYSTem:HEADer command/query, 863
:SYSTem:LOCK command/query, 864
:SYSTem:LONGform command/query, 865
:SYSTem:PRESet command, 866
:SYSTem:SETup command block data, 56
:SYSTem:SETup command/query, 867
:SYSTem:TIME command/query, 869
:TER? query, 781
:TIMebase:POSition command/query, 872
:TIMebase:RANGe command/query, 873
:TIMebase:REFClock command/query, 874
:TIMebase:REFerence command/query, 875
:TIMebase:SCALe command/query, 876
:TIMebase:VIEW command/query, 877
:TIMebase:WINDow:DELay command/query, 878
:TIMebase:WINDow:POSition command/query, 879
:TIMebase:WINDow:RANGe command/query, 880
:TIMebase:WINDow:SCALe command/query, 881
:TRIGger:AND:ENABle command/query, 886
:TRIGger:AND:SOURce command/query, 887
:TRIGger:COMM:BWIDth command/query, 900
:TRIGger:COMM:ENCode command/query, 901
:TRIGger:COMM:PATTern command/query, 902
:TRIGger:COMM:POLarity command/query, 903
:TRIGger:COMM:SOURce command/query, 904
:TRIGger:DELay:ARM:SLOPe command/query, 907
:TRIGger:DELay:ARM:SOURce command/query, 906
:TRIGger:DELay:EDELay:COUNt command/query, 908
:TRIGger:DELay:EDELay:SLOPe command/query, 910
:TRIGger:DELay:EDELay:SOURce command/query, 909
:TRIGger:DELay:MODE command/query, 911
:TRIGger:DELay:TDELay:TIME command/query, 912
:TRIGger:DELay:TRIGger:SLOPe command/query, 914
:TRIGger:DELay:TRIGger:SOURce command/query, 913
:TRIGger:EDGE:SLOPe command/query, 916
:TRIGger:EDGE:SOURce command/query, 917
:TRIGger:GLITch:POLarity command/query, 919
:TRIGger:GLITch:SOURce command/query, 920
:TRIGger:GLITch:WIDTh command/query, 921
:TRIGger:HOLDoff command/query, 888
:TRIGger:HOLDoff:MAX command/query, 889
:TRIGger:HOLDoff:MIN command/query, 890
:TRIGger:HOLDoff:MODE command/query, 891
:TRIGger:HTHReshold command/query, 892
:TRIGger:HYSTeresis command/query, 893
:TRIGger:LEVel command/query, 894
:TRIGger:LTHReshold command/query, 895
:TRIGger:MODE command/query, 896
:TRIGger:PATTern:CONDition command/query, 923
:TRIGger:PATTern:LOGic command/query, 924
:TRIGger:PWIDth:DIRection command/query, 926
:TRIGger:PWIDth:POLarity command/query, 927
:TRIGger:PWIDth:SOURce command/query, 928
:TRIGger:PWIDth:TPOint command/query, 929
:TRIGger:PWIDth:WIDTh command/query, 930
:TRIGger:RUNT:POLarity command/query, 932
:TRIGger:RUNT:QUALified command/query, 933
:TRIGger:RUNT:SOURce command/query, 934
:TRIGger:RUNT:TIME command/query, 935
:TRIGger:SEQuence:RESet:ENABle command/query, 939
:TRIGger:SEQuence:RESet:EVENt command, 941
:TRIGger:SEQuence:RESet:TIME command/query, 942
:TRIGger:SEQuence:RESet:TYPE command/query, 940
:TRIGger:SEQuence:TERM1 command/query, 937
:TRIGger:SEQuence:TERM2 command/query, 938
:TRIGger:SEQuence:WAIT:ENABle command/query, 943
:TRIGger:SEQuence:WAIT:TIME command/query, 944
:TRIGger:SHOLd:CSOurce command/query, 946
:TRIGger:SHOLd:CSOurce:EDGE command/query, 947
:TRIGger:SHOLd:DSOurce command/query, 948
:TRIGger:SHOLd:HoldTIMe (HTIMe) command/query, 949
:TRIGger:SHOLd:MODE command/query, 950
:TRIGger:SHOLd:SetupTIMe command/query, 951
:TRIGger:STATe:CLOCk command/query, 953
:TRIGger:STATe:LOGic command/query, 954
:TRIGger:STATe:LTYPe command/query, 955
:TRIGger:STATe:SLOPe command/query, 956
:TRIGger:SWEep command/query, 898
:TRIGger:TIMeout:CONDition command/query, 958
:TRIGger:TIMeout:SOURce command/query, 959
:TRIGger:TIMeout:TIME command/query, 960
:TRIGger:TRANsition:DIRection command/query, 962
:TRIGger:TRANsition:SOURce command/query, 963
:TRIGger:TRANsition:TIME command/query, 964
:TRIGger:TRANsition:TYPE command/query, 965
:TRIGger:TV:LINE command/query, 967
:TRIGger:TV:MODE command/query, 968
:TRIGger:TV:POLarity command/query, 969
:TRIGger:TV:SOURce command/query, 970
:TRIGger:TV:STANdard command/query, 971
:TRIGger:TV:UDTV:ENUMber command/query, 972

:TRIGger:TV:UDTV:HSYNc
command/query, 973
:TRIGger:TV:UDTV:HTIMe
command/query, 974
:TRIGger:TV:UDTV:PGTHan
command/query, 975
:TRIGger:TV:UDTV:POLarity
command/query, 976
:TRIGger:WINDow:CONDition
command/query, 978
:TRIGger:WINDow:SOURce
command/query, 979
:TRIGger:WINDow:TIME command/query, 980
:TRIGger:WINDow:TPOint
command/query, 981
:VIEW command, 782
:WAVeform:BANDpass? query, 986
:WAVeform:BYTeorder command/query, 987
:WAVeform:COMPlete? query, 988
:WAVeform:COUNt? query, 989
:WAVeform:COUPling? query, 990
:WAVeform:DATA? query, 991
:WAVeform:FORMat command/query, 1013
:WAVeform:POINts? query, 1016
:WAVeform:PREamble? query, 1017
:WAVeform:SEGMented:ALL
command/query, 1022
:WAVeform:SEGMented:COUNt? query, 1023
:WAVeform:SEGMented:TTAG? query, 1024
:WAVeform:SEGMented:XLISt? query, 1025
:WAVeform:SOURce command/query, 1026
:WAVeform:STReaming command/query, 1027
:WAVeform:TYPE? query, 1028
:WAVeform:VIEW command/query, 1029
:WAVeform:XDISplay? query, 1031
:WAVeform:XINCrement? query, 1032
:WAVeform:XORigin? query, 1033
:WAVeform:XRANge? query, 1034
:WAVeform:XREFerence? query, 1035
:WAVeform:XUNits? query, 1036
:WAVeform:YDISplay? query, 1037
:WAVeform:YINCrement? query, 1038
:WAVeform:YORigin? query, 1039
:WAVeform:YRANge? query, 1040
:WAVeform:YREFerence? query, 1041
:WAVeform:YUNits? query, 1042
:WMEMory<N>:CLEar command, 1044
:WMEMory<N>:DISPlay
command/query, 1045
:WMEMory<N>:LOAD command, 1046
:WMEMory<N>:SAVE command, 1047
:WMEMory<N>:XOFFset
command/query, 1048
:WMEMory<N>:XRANge
command/query, 1049
:WMEMory<N>:YOFFset
command/query, 1050
:WMEMory<N>:YRANge
command/query, 1051
..., Ellipsis, 60
(Event Status Enable (*ESE)
command/query, 254
*CLS (Clear Status) command, 253
*ESE (Event Status Enable)
command/query, 254
*ESR? (Event Status Register) query, 256
*IDN? (Identification Number) query, 257
*LRN? (Learn) query, 258
*LRN?, and SYSTem SETup?[LRN], 868
*OP?T (Option) query, 261
*OPC (Operation Complete)
command/query, 260
*PSC (Power-on Status Clear)
command/query, 264
*RCL (Recall) command, 265
*RST (Reset) command, 266
*SAV (Save) command, 267
*SRE (Service Request Enable)
command/query, 268
*STB? (Status Byte) query, 270
*TRG (Trigger) command, 272
*TST? (Test) query, 273
*WAI (Wait-to-Continue) command, 274

Numerics

82350A GPIB interface, 4
9.99999E+37, Infinity Representation, 144

A

Aborting a digitize operation, 100
aborting a digitize operation, 81
absolute voltage, and VMAX, 721
absolute voltage, and VMIN, 724
ABSolute, :FUNCtion<F>:ABSolute
command, 329
ABSolute, :MEASure:THResholds:ABSolute
command/query, 652
ABSolute,
:MEASure:THResholds:GENeral:ABSolute
command/query, 654
ABSolute,
:MEASure:THResholds:GENeral:TOPBase:A
BSolute command/query, 662
ABSolute,
:MEASure:THResholds:RFALl:ABSolute
command/query, 672
ABSolute,
:MEASure:THResholds:RFALl:TOPBase:ABS
olute command/query, 680
ABSolute,
:MEASure:THResholds:SERial:ABSolute
command/query, 684
ABSolute,
:MEASure:THResholds:SERial:TOPBase:AB
Solute command/query, 692
ABSolute,
:MEASure:THResholds:TOPBase:ABSolute
command/query, 696
ACCAL, :CHANnel<N>:PROBe:ACCAL
command/query, 223
accuracy and probe calibration, 188
Acquire Commands, 149
Acquire Commands, SRATe, 175
acquisition state, 751
acquisition, ACQuire AVER and completion, 154
acquisition, points, 161
acquisition, record length, 161
acquisition, sample rate, 172, 175
ADD, :CHANnel<N>:PROBe:HEAD:ADD
command, 235
ADD, :FUNCtion<F>:ADD command, 330
address field size, IIC serial decode, 793
address, GPIB default, 95
ADEMod, :FUNCtion<F>:ADEMod
command, 331
advisory line, reading and writing to, 857
Agilent Connection Expert, 48
Agilent Interactive IO application, 51
Agilent IO Control icon, 48
Agilent IO Libraries Suite, 4, 45, 75, 77
Agilent IO Libraries Suite, installing, 46
algebraic sum of functions, 330
ALIGn, :MEASure:CLOCk:METHod:ALIGn
command/query, 504
ALIGn, :MTESt:ALIGn command, 437
AlignFIT, :MTESt:AlignFIT command/query, 438
all edges, measure, 532, 534, 548, 576, 606,
608
ALL, :WAVeform:SEGMented:ALL
command/query, 1022
ALL?, :MEASure:NOISe:ALL? query, 587
ALL?, :MEASure:RJDJ:ALL? query, 624
ALL?, :MEASure:RJDJ:APLength? query, 626
alphanumeric, characters in embedded
string, 70
alphanumeric, strings, 68
AMASk, :MTESt:AMASk:CREate
command, 440
AMASk, :MTESt:AMASk:SAVE command, 442
AMASk, :MTESt:AMASk:SOURce
command/query, 441
AMASk, :MTESt:AMASk:UNITs
command/query, 443
AMASk, :MTESt:AMASk:XDELta
command/query, 444
AMASk, :MTESt:AMASk:YDELta
command/query, 445
AMPS as vertical units, 233, 250
ANALog, :ACQuire:POINts:ANALog
command/query, 161
ANALog, :ACQuire:SRATe:ANALog
command/query, 172
ANALog, :ACQuire:SRATe:ANALog:AUTO
command/query, 174
AND, :TRIGger:AND:ENABle
command/query, 886
AND, :TRIGger:AND:SOURce
command/query, 887
APPLy, :CHANnel<N>:ISIM:APPLy
command/query, 206
AREA, :HARDcopy:AREA command/query, 372
AREA, :MEASure:AREA command/query, 488

ARM, :TRIGger:DELay:ARM:SLOPe command/query, 907
ARM, :TRIGger:DELay:ARM:SOURce command/query, 906
Arming the trigger, 100
ASCII, and FORMat, 1013
ASCII, character 32, 58
ASCII, linefeed, 71
ASIZe, :SBUS<N>:IIC:ASIZe command/query, 793
attenuation factor for probe, 188, 222
ATTenuation, :CHANnel<N>:PROBe:ATTenuation command/query, 224
AUTO, :ACQuire:POINts:AUTO command/query, 165
AUTO, :ACQuire:SRATe:ANALog:AUTO command/query, 174
AUTO, :ACQuire:SRATe:DIGital:AUTO command/query, 176
AUTO, :CHANnel<N>:DISPlay:AUTO command/query, 201
AUTO, :MTESt:AUTO command/query, 446
AUTomatic, :SPRocessing:DFEQualizer:TAP:AUTomatic command, 838
AUTomatic, :SPRocessing:FFEQualizer:TAP:AUTomatic command, 848
AUToscale, during initialization, 77
AUTozero, :CHANnel<N>:PROBe:AUTozero command/query, 225
Aux Out connector, 190
availability of measured data, 105
AVERage, :ACQuire:AVERage command/query, 150
AVERage, :ACQuire:AVERage:COUNt command/query, 151
AVERage, :FUNCtion<F>:AVERage command, 332
AVERage, :MTESt:AVERage command/query, 447
AVERage, :MTESt:AVERage:COUNt command/query, 448
AVERage, and acquisition completion, 154
AVERage, and count, 448
AXIS, :HISTogram:AXIS command/query, 379

B

B<N>, :BUS:B<N>:TYPE command/query, 178
BANDpass?, :WAVeform:BANDpass? query, 986
bandwidth limit, 986
BANdwidth, :ACQuire:BANDwidth, command/query, 152
BANDwidth, :CHANnel<N>:ISIM:BANDwidth command/query, 207
BANDwidth, :CHANnel<N>:PROBe:PRECprobe:BANDwidth command, 241
BANDwidth, :MEASure:NOISe:BANDwidth command/query, 589
BANDwidth, :MEASure:RJDJ:BANDwidth command/query, 627
BANDwidth, :SPRocessing:FFEQualizer:TAP:BANDwidth command/query, 849
basic command structure, 79
basic operations, 54
baud rate, 789
BAUDrate, :SBUS<N>:CAN:SIGNal:BAUDrate command/query, 789
BER, :MEASure:RJDJ:BER command/query, 628
BIND, :MTESt:SCALe:BIND command/query, 468
BINTerval, :MEASure:BINTerval command/query, 489
Bit Definitions in Status Reporting, 106
bit order, SPI decode, 797
BIT<M>, :BUS<N>:BIT<M> command/query, 179
BITS, :BUS<N>:BITS command/query, 180
BITS, :MTESt:FOLDing:BITS command/query, 457
blanking the user text area, 323
BLIMit, :HISTogram:WINDow:BLIMit command/query, 386
block data, 56, 85
block data, in :SYSTem:SETup command, 56
Block Diagram, Status Reporting Overview, 106
BPERiod, :MEASure:BPERiod command/query, 490
Braces, 59
Brackets, Square, 61
brickwall filter, TIE, 704
buffer, output, 66, 82
buffered responses, 146
Bus Activity, Halting, 100
Bus Commands, 100
BWIDth, :MEASure:BWIDth command/query, 491
BWIDth, :TRIGger:COMM:BWIDth command/query, 900
BWLimit, :CHANnel<N>:ISIM:BWLimit command/query, 208
BWMode, :SPRocessing:FFEQualizer:TAP:BWMode command/query, 850
BYTE, and FORMat, 1014
BYTE, Understanding the format, 1008
BYTeorder, :WAVeform:BYTeorder command/query, 987
BYTeorder, and DATA, 993

C

C Program, DATA? Analog Channels, 993
C Program, DATA? Digital Channels, 999
C, SICL library example, 1162
C, VISA library example, 1111
C#, SCPI.NET example, 1181
C#, VISA COM example, 1083
C#, VISA example, 1130
Calibration Commands, 187
calibration status, 192
CALibration, :CHANnel<N>:PROBe:PRECprobe:CALibration command, 242
CAN acknowledge, 788
CAN baud rate, 789
CAN serial bus commands, 787
CAN signal definition, 790
CAN source, 791
CAN, :SBUS<N>:CAN:SAMPlepoint command/query, 788
CAN, :SBUS<N>:CAN:SIGNal:BAUDrate command/query, 789
CAN, :SBUS<N>:CAN:SIGNal:DEFinition command/query, 790
CAN, :SBUS<N>:CAN:SOURce command/query, 791
CANCel, :SELFtest:CANCel command, 810
CDIRectory, :DISK:CDIRectory command, 282
CDRRATE, :MEASure:CDRRATE command, 492
center screen voltage, 221, 232
CGRade, :DISPlay:CGRade command/query, 300
CGRade, :DISPlay:CGRade:LEVels? query, 301
CGRade, :DISPlay:CGRade:SCHeme command/query, 303
CGRade, :MEASure:CGRade:CROSsing command/query, 493
CGRade, :MEASure:CGRade:DCDistortion command/query, 494
CGRade, :MEASure:CGRade:EHEight command/query, 495
CGRade, :MEASure:CGRade:EWIDth command/query, 496
CGRade, :MEASure:CGRade:EWINdow command/query, 497
CGRade, :MEASure:CGRade:JITTer command/query, 498
CGRade, :MEASure:CGRade:QFACtor command/query, 499
Channel Commands, 195
CHANnels, :AUToscale:CHANnels command, 754
channels, and VIEW, 1029
channel-to-channel skew factor, 191
character program data, 68
CLASsic color grade scheme, 303
Clear method, 77
Clear Status (*CLS) command, 253
CLEar, :BUS<N>:CLEar command/query, 181
CLEar, :MEASure:CLEar command, 500
CLEar, :WMEMory<N>:CLEar command, 1044
Clearing, Buffers, 100
clearing, DONE bit, 121
clearing, error queue, 125, 1060
Clearing, Pending Commands, 100
clearing, registers and queues, 128

clearing, Standard Event Status Register, 115, 256
clearing, status data structures, 253
clearing, TRG bit, 114, 123
clipped waveforms, and measurement error, 487
clock timeout, SPI, 799
CLOCk, :BUS<N>:CLOCk command/query, 182
CLOCk, :BUS<N>:CLOCk:SLOPe command/query, 183
CLOCk, :MEASure:CLOCk command/query, 501
CLOCk, :MEASure:CLOCk:METHod command/query, 502, 1054
CLOCk, :MEASure:CLOCk:METHod:ALIGn command/query, 504
CLOCk, :MEASure:CLOCk:METHod:DEEMphasis command/query, 505
CLOCk, :MEASure:CLOCk:METHod:EDGE command/query, 506
CLOCk, :MEASure:CLOCk:METHod:JTF command/query, 508
CLOCk, :MEASure:CLOCk:METHod:OJTF command/query, 510
CLOCk, :MEASure:CLOCk:VERTical command/query, 512
CLOCk, :MEASure:CLOCk:VERTical:OFFSet command/query, 513
CLOCk, :MEASure:CLOCk:VERTical:RANGe command/query, 514
CLOCk, :MEASure:RJDJ:CLOCk command/query, 630
CLOCk, :SBUS<N>:IIC:SOURce:CLOCk command/query, 794
CLOCk, :SBUS<N>:SPI:BITorder command/query, 797
CLOCk, :SBUS<N>:SPI:CLOCk:SLOPe command/query, 798
CLOCk, :SBUS<N>:SPI:CLOCk:TIMeout command/query, 799
CLOCk, :SBUS<N>:SPI:SOURce:CLOCk command/query, 801
CLOCk, :TRIGger:STATe:CLOCk command/query, 953
CME bit, 254, 256
code, SCPI.NET library example in C#, 1181
code, SCPI.NET library example in IronPython, 1196
code, SCPI.NET library example in Visual Basic .NET, 1189
code, SICL library example in C, 1162
code, SICL library example in Visual Basic, 1171
code, VISA COM library example in C#, 1083
code, VISA COM library example in Python, 1103
code, VISA COM library example in Visual Basic, 1072
code, VISA COM library example in Visual Basic .NET, 1093
code, VISA library example in C, 1111
code, VISA library example in C#, 1130
code, VISA library example in Python, 1154
code, VISA library example in Visual Basic, 1120
code, VISA library example in Visual Basic .NET, 1142
COL, :DISPlay:STATus:COL command query, 319
COLumn, :DISPlay:COLumn command/query, 305
combining compound and simple commands, 73
combining, commands in same subsystem, 65
combining, long- and short-form headers, 67
COMM, :TRIGger:COMM:BWIDth command/query, 900
COMM, :TRIGger:COMM:ENCode command/query, 901
COMM, :TRIGger:COMM:PATTern command/query, 902
COMM, :TRIGger:COMM:POLarity command/query, 903
COMM, :TRIGger:COMM:SOURce command/query, 904
Command and Data Concepts, GPIB, 94
Command Error, 1062
Command Error, Status Bit, 106
Command Expert, 1181
Command tree, 141
Command Types, 141
Command, DIGitize, 80
command, execution and order, 103
Command, GPIB Mode, 94
command, structure, 79
Command, TRIGger MODE, 884
commands embedded in program messages, 72
commands, obsolete and discontinued, 1053
commas and spaces, 63
Common Command Header, 65
Common Commands, 251
Common Commands, Reset (*RST), 266
Common Commands, within a program message, 251
commonmode voltage of operands, 333
COMMonmode, :CHANnel<N>:COMMonmode command/query, 197
COMMonmode, :FUNCtion<F>:COMMonmode command, 333
Communicating Over the GPIB Interface, 95
Communicating Over the LAN Interface, 96
COMPlete, :ACQuire:COMPlete command/query, 154
COMPlete, :ACQuire:COMPlete:STATe command/query, 156
COMPlete?, :WAVeform:COMPlete? query, 988
COMPosite, :DISK:SAVE:COMPosite command, 289
compound command header, 64
compound queries, 103
Computer Code and Capability, 93
computer control examples, 1071
concurrent commands, 145
CONDition, :MEASure:QUALifier<M>:CONDition command/query, 616
CONDition, :TRIGger:PATTern:CONDition command/query, 923
CONDition, :TRIGger:TIMeout:CONDition command/query, 958
CONDition, :TRIGger:WINDow:CONDition command/query, 978
connect oscilloscope, 47
CONNect, :DISPlay:CONNect command/query, 306
conventions of programming, 139
converting waveform data, from data value to Y-axis units, 984
CONVolve, :CHANnel<N>:ISIM:CONVolve command/query, 209
COPY, :DISK:COPY command, 283
copying files, 283
CORRection, :CHANnel<N>:ISIM:CORRection command/query, 210
COUNt, :ACQuire:AVERage:COUNt command/query, 151
COUNt, :ACQuire:SEGMented:COUNt command/query, 169
COUNt, :MTESt:AVERage:COUNt command/query, 448
COUNt, :MTESt:COUNt:FAILures? query, 449
COUNt, :MTESt:COUNt:FUI? query, 450
COUNt, :MTESt:COUNt:FWAVeforms? query, 451
COUNt, :MTESt:COUNt:UI? query, 452
COUNt, :MTESt:COUNt:WAVeforms? query, 453
COUNt, :TRIGger:DELay:EDELay:COUNt command/query, 908
COUNt?, :WAVeform:COUNt? query, 989
COUNt?, :WAVeform:SEGMented:COUNt? query, 1023
COUPling, :CHANnel<N>:PROBe:COUPling command/query, 226
coupling, input, 205
COUPling?, :WAVeform:COUPling? query, 990
CREate, :MTESt:AMASk:CREate command, 440
CROSsing, :MEASure:CGRade:CROSsing command/query, 493
CROSsing, :MEASure:CROSsing command/query, 515
CSOurce, :TRIGger:SHOLd:CSOurce command/query, 946
CSOurce, :TRIGger:SHOLd:CSOurce:EDGE command/query, 947
CTCDutycycle, :MEASure:CTCDutycycle command/query, 516
CTCJitter, :MEASure:CTCJitter command/query, 518
CTCNwidth, :MEASure:CTCNwidth command/query, 520
CTCPwidth, :MEASure:CTCPwidth command/query, 522

CTLequalizer, :SPRocessing:CTLequalizer:DCGain command/query, 817
CTLequalizer, :SPRocessing:CTLequalizer:DISPlay command/query, 815
CTLequalizer, :SPRocessing:CTLequalizer:NUMPoles command/query, 818
CTLequalizer, :SPRocessing:CTLequalizer:P1 command/query, 819
CTLequalizer, :SPRocessing:CTLequalizer:P2 command/query, 820
CTLequalizer, :SPRocessing:CTLequalizer:P3 command/query, 821
CTLequalizer, :SPRocessing:CTLequalizer:RATe command/query, 822
CTLequalizer, :SPRocessing:CTLequalizer:SOURce command/query, 816
CTLequalizer, :SPRocessing:CTLequalizer:VERTical command/query, 823
CTLequalizer, :SPRocessing:CTLequalizer:VERTical:OFFSet command/query, 824
CTLequalizer, :SPRocessing:CTLequalizer:VERTical:RANGe command/query, 825
CTLequalizer, :SPRocessing:CTLequalizer:ZERo command/query, 826
CURSor?, :MARKer:CURSor? query, 424

D

data in a :SYSTem:SETup command, 56
data in a program, 63
Data Mode, GPIB, 94
data source, SPI trigger, 802
Data Structures, and Status Reporting, 108
data transmission mode, and FORMat, 1013
DATA, :DISPlay:DATA? query, 307
DATA, :SBUS<N>:IIC:SOURce:DATA command/query, 795
DATA, :SBUS<N>:SPI:SOURce:DATA command/query, 802
data, acquisition, 984
data, conversion, 984
DATA?, :LISTer:DATA? query, 420
DATA?, :WAVeform:DATA? query, 991
DATA?, Analog Channels C Program, 993
DATA?, Digital Channels C Program, 999
DATarate, :MEASure:DATarate command/query, 524
DATE, :CALibrate:DATE? query, 189
DATE, :SYSTem:DATE command/query, 858
DCDistortion, :MEASure:CGRade:DCDistortion command/query, 494
DCGain, :SPRocessing:CTLequalizer:DCGain command/query, 817
DDE bit, 255, 256
DDPWS, :MEASure:DDPWS command/query, 35, 1057
DEBug, :SYSTem:DEBug command/query, 859
decimal 32 (ASCII space), 58
Decision Chart for Status Reporting, 128
decode type, SPI, 806
DEConvolve, :CHANnel<N>:ISIM:DEConvolve command/query, 212
DEEMphasis, :MEASure:CLOCk:METHod:DEEMphasis command/query, 505
DEEMphasis, :MEASure:DEEMphasis command/query, 526
default setup, 866
Default Startup Conditions, 92
DEFault, :HISTogram:WINDow:DEFault command, 382
Default, GPIB Address, 95
Default, Startup Conditions, 92
DEFine, :MEASure:DELTatime:DEFine command/query, 530
defining functions, 326
DEFinition, :SBUS<N>:CAN:SIGNal:DEFinition command/query, 790
def-length block response data, 85
DELay, :CHANnel<N>:ISIM:DELay command/query, 213
DELay, :FUNCtion<F>:DELay command, 334
DELay, :ISCan:DELay command/query, 390
DELay, :SPRocessing:DFEQualizer:TAP:DELay command/query, 832
DELay, :SPRocessing:FFEQualizer:TAP:DELay command/query, 847
DELay, :TIMebase:WINDow:DELay command/query, 878
DELay, :TRIGger:DELay:ARM:SLOPe command/query, 907
DELay, :TRIGger:DELay:ARM:SOURce command/query, 906
DELay, :TRIGger:DELay:EDELay:COUNt command/query, 908
DELay, :TRIGger:DELay:EDELay:SLOPe command/query, 910
DELay, :TRIGger:DELay:EDELay:SOURce command/query, 909
DELay, :TRIGger:DELay:MODE command/query, 911
DELay, :TRIGger:DELay:TDELay:TIME command/query, 912
DELay, :TRIGger:DELay:TRIGger:SLOPe command/query, 914
DELay, :TRIGger:DELay:TRIGger:SOURce command/query, 913
delay, and WINDow DELay, 878
DELete, :CHANnel<N>:PROBe:HEAD:DELete command, 236
DELete, :DISK:DELete command, 284
DELete, :MTESt:DELete command, 454
deleting files, 284
DELTatime, :MEASure:DELTatime command/query, 528
DELTatime, :MEASure:DELTatime:DEFine command/query, 530
derivative function, 335
Device Address, GPIB, 95
Device Address, LAN, 96
Device Clear (DCL), 100
Device Clear Code and Capability, 93
Device Dependent Error (DDE), Status Bit, 107
Device- or Oscilloscope-Specific Error, 1064
Device Trigger Code and Capability, 93
device-dependent data, 85
DFEQualizer, :SPRocessing:DFEQualizer:NTAPs command/query, 829
DFEQualizer, :SPRocessing:DFEQualizer:SOURce command/query, 828
DFEQualizer, :SPRocessing:DFEQualizer:STATe command/query, 827
DFEQualizer, :SPRocessing:DFEQualizer:TAP command/query, 830
DFEQualizer, :SPRocessing:DFEQualizer:TAP:AUTomatic command, 838
DFEQualizer, :SPRocessing:DFEQualizer:TAP:DELay command/query, 832
DFEQualizer, :SPRocessing:DFEQualizer:TAP:GAIN command/query, 835
DFEQualizer, :SPRocessing:DFEQualizer:TAP:LTARget command/query, 837
DFEQualizer, :SPRocessing:DFEQualizer:TAP:MAX command/query, 833
DFEQualizer, :SPRocessing:DFEQualizer:TAP:MIN command/query, 834
DFEQualizer, :SPRocessing:DFEQualizer:TAP:UTARget command/query, 836
DFEQualizer, :SPRocessing:DFEQualizer:TAP:WIDTh command/query, 831
DFRequency, :MEASure:FFT:DFRequency command/query, 539
DIFF, :FUNCtion<F>:DIFF command, 335
DIFFerential, :CHANnel<N>:DIFFerential command/query, 198
DIFFerential, :CHANnel<N>:DIFFerential:SKEW command/query, 199
Digital Commands, 275
DIGital, :ACQuire:POINts:DIGital? query, 166
DIGital, :ACQuire:SRATe:DIGital command/query, 175
DIGital, :ACQuire:SRATe:DIGital:AUTO command/query, 176
Digitize, Aborting, 100
DIGitize, setting up for execution, 149
DIRection, :TRIGger:PWIDth:DIRection command/query, 926
DIRection, :TRIGger:TRANsition:DIRection command/query, 962

DIRectory, :DISK:DIRectory? query, 285
Disabling Serial Poll, 100
discontinued and obsolete commands, 1053
discrete derivative function, 335
Disk Commands, 281
Disk Commands, SEGMented, 298
Display Commands, 299
Display Commands, DATA?, 307
Display Commands, TEXT, 323
display persistence, 315
DISPlay, :BUS<N>:DISPlay command/query, 184
DISPlay, :CHANnel<N>:DISPlay command/query, 200
DISPlay, :CHANnel<N>:DISPlay:AUTO command/query, 201
DISPlay, :CHANnel<N>:DISPlay:OFFSet command/query, 202
DISPlay, :CHANnel<N>:DISPlay:RANGe command/query, 203
DISPlay, :CHANnel<N>:DISPlay:SCALe command/query, 204
DISPlay, :DIGital<N>:DISPlay command/query, 276
DISPlay, :FUNCtion<F>:DISPlay command/query, 336
DISPlay, :LISTer:DISPlay command/query, 421
DISPlay, :POD<N>:DISPlay command/query, 742
DISPlay, :SPRocessing:CTLequalizer:DISPlay command/query, 815
DISPlay, :SPRocessing:FFEQualizer:DISPlay command/query, 839
DISPlay, :WMEMory<N>:DISPlay command/query, 1045
display, serial decode bus, 785
DIVide, :FUNCtion<F>:DIVide command, 337
dividing functions, 337
DMAGnitude, :MEASure:FFT:DMAGnitude command/query, 541
DPRinter, :HARDcopy:DPRinter command/query, 373
Driver Electronics Code and Capability, 93
DSOurce, :TRIGger:SHOLd:DSOurce command/query, 948
DSP, :SYSTem:DSP command/query, 861
duplicate mnemonics, 65
DUTYcycle, :MEASure:DUTYcycle command/query, 532

E

EADapter, :CHANnel<N>:PROBe:EADapter command/query, 227
ECOupling, :CHANnel<N>:PROBe:ECOupling command/query, 229
EDELay, :TRIGger:DELay:EDELay:COUNt command/query, 908
EDELay, :TRIGger:DELay:EDELay:SLOPe command/query, 910
EDELay, :TRIGger:DELay:EDELay:SOURce command/query, 909
EDGE, :ISCan:NONMonotonic:EDGE command/query, 396
EDGE, :MEASure:CLOCk:METHod:EDGE command/query, 506
EDGE, :MEASure:EDGE command/query, 534
EDGE, :MEASure:RJDJ:EDGE command/query, 631
EDGE, :TRIGger:EDGE:SLOPe command/query, 916
EDGE, :TRIGger:EDGE:SOURce command/query, 917
EDGE, :TRIGger:SHOLd:CSOurce:EDGE command/query, 947
edges, measure all, 532, 534, 548, 576, 606, 608
EHEight, :MEASure:CGRade:EHEight command/query, 495
Ellipsis, ..., 60
embedded, commands, 72
embedded, strings, 55, 56, 70
Enable Register, 252
ENABle, :MTESt:ENABle command/query, 455
ENABle, :TRIGger:AND:ENABle command/query, 886
ENABle, :TRIGger:SEQuence:RESet:ENABle command/query, 939
ENABle, :TRIGger:SEQuence:WAIT:ENABle command/query, 943
ENCode, :TRIGger:COMM:ENCode command/query, 901
End Of String (EOS), 71
End Of Text (EOT), 71
End-Or-Identify (EOI), 71
Enhanced Bandwidth, 152
ENUMber, :TRIGger:TV:UDTV:ENUMber command/query, 972
EOI and IEEE 488.2, 147
error messages, 1059
Error Messages table, 1066
error queue, 1060
error queue, and status reporting, 125
error queue, overflow, 1060
error, in measurements, 486
error, numbers, 1061
error, query interrupt, 66, 82
ERRor?, :SYSTem:ERRor? query, 862
errors, exceptions to protocol, 103
ESB (Event Status Bit), 107, 268, 270
ESB (Event Summary Bit), 254
ESR (Standard Event Status Register), 115
ETIMe acquisition mode, 159
ETOedge, :MEASure:ETOedge command, 536
event monitoring, 105
Event Registers Default, 92
Event Status Bit (ESB), 107
Event Status Enable (*ESE), Status Reporting, 116
Event Status Register (*ESR?) query, 256
Event Summary Bit (ESB), 254
EVENt, :TRIGger:SEQuence:RESet:EVENt command, 941
EWIDth, :MEASure:CGRade:EWIDth command/query, 496
EWINdow, :MEASure:CGRade:EWINdow command/query, 497
Example Program, 79
Example Program, in initialization, 79
example programs, 1071
exceptions to protocol, 103
EXE bit, 255, 256
executing DIGITIZE, 149
Execution Error, 1063
Execution Error (EXE), Status Bit, 106
execution, errors, and command errors, 1062
execution, of commands and order, 103
exponential notation, 69
exponents, 69
EXTernal, :CHANnel<N>:PROBe:EXTernal command/query, 230
EXTernal, :CHANnel<N>:PROBe:EXTernal:GAIN command/query, 231
EXTernal, :CHANnel<N>:PROBe:EXTernal:OFFSet command/query, 232
EXTernal, :CHANnel<N>:PROBe:EXTernal:UNITs command/query, 233

F

FACTors, :HARDcopy:FACTors command/query, 374
FAIL, :ISCan:MEASurement:FAIL command/query, 391
FAIL, :LTESt:FAIL command/query, 412
FAILures?, :MTESt:COUNt:FAILures? query, 449
fall time measurement setup, 486
FALLtime, :MEASure:FALLtime command/query, 537
FAST, :MTESt:FOLDing:FAST command/query, 458
Fast, Worst Case Only option, 31, 458
FFEequalizer, :SPRocessing:FFEequalizer:RATe command/query, 843
FFEQualizer, :SPRocessing:FFEQualizer:DISPlay command/query, 839
FFEQualizer, :SPRocessing:FFEQualizer:NPRecursor command/query, 841
FFEQualizer, :SPRocessing:FFEQualizer:NTAPs command/query, 842
FFEQualizer, :SPRocessing:FFEQualizer:SOURce command/query, 840
FFEQualizer, :SPRocessing:FFEQualizer:TAP command/query, 844
FFEQualizer, :SPRocessing:FFEQualizer:TAP:AUTomatic command, 848
FFEQualizer, :SPRocessing:FFEQualizer:TAP:BANDwidth command/query, 849

FFEQualizer,
:SPRocessing:FFEQualizer:TAP:BWMode
command/query, 850
FFEQualizer,
:SPRocessing:FFEQualizer:TAP:DELay
command/query, 847
FFEQualizer,
:SPRocessing:FFEQualizer:TAP:PLENgth
command/query, 845
FFEQualizer,
:SPRocessing:FFEQualizer:TAP:TDELay
command/query, 851
FFEQualizer,
:SPRocessing:FFEQualizer:TAP:TDMode
command/query, 852
FFEQualizer,
:SPRocessing:FFEQualizer:TAP:WIDTh
command/query, 846
FFEQualizer, :SPRocessing:FFEQualizer:VERTical
command/query, 853
FFEQualizer,
:SPRocessing:FFEQualizer:VERTical:OFFSet
command/query, 854
FFEQualizer,
:SPRocessing:FFEQualizer:VERTical:RANGe
command/query, 855
FFT Commands, 486
FFT, :FUNCtion<F>:FFT:FREQuency
command/query, 338
FFT, :FUNCtion<F>:FFT:REFerence
command/query, 339
FFT, :FUNCtion<F>:FFT:RESolution?
query, 340
FFT, :FUNCtion<F>:FFT:TDELay
command/query, 341
FFT, :FUNCtion<F>:FFT:WINDow
command/query, 342
FFT, :MEASure:FFT:DFRequency
command/query, 539
FFT, :MEASure:FFT:DMAGnitude
command/query, 541
FFT, :MEASure:FFT:FREQuency
command/query, 543
FFT, :MEASure:FFT:MAGNitude
command/query, 544
FFT, :MEASure:FFT:PEAK1
command/query, 545
FFT, :MEASure:FFT:PEAK2
command/query, 546
FFT, :MEASure:FFT:THReshold
command/query, 547
FFTMagnitude, :FUNCtion<F>:FFTMagnitude
command, 344
FFTPhase, :FUNCtion<F>:FFTPhase
command, 345
FOLDing, :MTESt:FOLDing
command/query, 456
FOLDing, :MTESt:FOLDing:BITS
command/query, 457
FOLDing, :MTESt:FOLDing:FAST
command/query, 458
FORMat, :WAVeform:FORMat
command/query, 1013
FORMat, and DATA, 993
FormattedIO488 object, 76
formatting query responses, 857
fractional values, 69
FRAMe, :SBUS<N>:SPI:FRAMe:STATe
command/query, 800
FRAMe, :SBUS<N>:SPI:SOURce:FRAMe
command/query, 803
frequency measurement setup, 486
FREQuency, :FUNCtion<F>:FFT:FREQuency
command/query, 338
FREQuency, :MEASure:FFT:FREQuency
command/query, 543
FREQuency, :MEASure:FREQuency
command/query, 548
FUI?, :MTESt:COUNt:FUI? query, 450
full-scale vertical axis, 248
Function Commands, 325
function, and vertical scaling, 361
function, time scale, 326
functional elements of protocol, 102
functions, and VIEW, 1029
functions, combining in instructions, 65
FWAVeforms?, :MTESt:COUNt:FWAVeforms?
query, 451

G

gain and offset of a probe, 187
gain factor for user-defined probe, 231
GAIN, :CHANnel<N>:PROBe:EXTernal:GAIN
command/query, 231
GAIN, :CHANnel<N>:PROBe:GAIN
command/query, 234
GAIN, :SPRocessing:DFEQualizer:TAP:GAIN
command/query, 835
GATing, :FUNCtion<F>:GATing command, 346
general SBUS<N> commands, 784
GENeral,
:MEASure:THResholds:GENeral:ABSolute
command/query, 654
GENeral,
:MEASure:THResholds:GENeral:HYSTeresis
command/query, 656
GENeral,
:MEASure:THResholds:GENeral:METHod
command/query, 658
GENeral,
:MEASure:THResholds:GENeral:PERCent
command/query, 660
GENeral,
:MEASure:THResholds:GENeral:TOPBase:A
BSolute command/query, 662
GENeral,
:MEASure:THResholds:GENeral:TOPBase:
METHod command/query, 664
GLITch, :TRIGger:GLITch:POLarity
command/query, 919
GLITch, :TRIGger:GLITch:SOURce
command/query, 920
GLITch, :TRIGger:GLITch:WIDTh
command/query, 921
GPIB interface, 47
GPIB, Interface Connector, 91
GRATicule, :DISPlay:GRATicule command, 311
GRATicule, :DISPlay:GRATicule
command/query, 308
GRATicule, :DISPlay:GRATicule:INTensity
command/query, 309
GRATicule, :DISPlay:GRATicule:NUMBer
command/query, 310
GRATicule, :DISPlay:GRATicule:SIZE
command/query, 312
GRATicule, HARDcopy AREA, 372
Group Execute Trigger (GET), 100

H

Halting bus activity, 100
HAMPlitude, :MTESt:HAMPlitude
command/query, 459
Hardcopy Commands, 371
Hardcopy Commands, AREA, 372
hardcopy of the screen, 371
hardcopy output and message termination, 103
HEAD, :CHANnel<N>:PROBe:HEAD:ADD
command, 235
HEAD, :CHANnel<N>:PROBe:HEAD:DELete
command, 236
HEAD, :CHANnel<N>:PROBe:HEAD:SELect
command/query, 237
HEAD, :CHANnel<N>:PROBe:HEAD:VTERm
command/query, 238
HEADer, :SYSTem:HEADer
command/query, 863
header, within instruction, 56
headers, 57
headers, types, 64
HIDE, :ISCan:ZONE:HIDE command/query, 405
HIGHpass, :FUNCtion<F>:HIGHpass
command, 347
Histogram Commands, 377
HISTogram, :MEASure:HISTogram:HITS
command/query, 550
HISTogram, :MEASure:HISTogram:M1S
command/query, 551
HISTogram, :MEASure:HISTogram:M2S
command/query, 552
HISTogram, :MEASure:HISTogram:M3S
command/query, 553
HISTogram, :MEASure:HISTogram:MAX
command/query, 554
HISTogram, :MEASure:HISTogram:MEAN
command/query, 555
HISTogram, :MEASure:HISTogram:MEDian
command/query, 556
HISTogram, :MEASure:HISTogram:MIN
command/query, 557
HISTogram, :MEASure:HISTogram:MODE
command/query, 558
HISTogram, :MEASure:HISTogram:PEAK
command/query, 559

HISTogram, :MEASure:HISTogram:PP command/query, 560
HISTogram, :MEASure:HISTogram:RESolution command/query, 561
HISTogram, :MEASure:HISTogram:STDDev command/query, 562
HISTogram, :MEASure:JITTer:HISTogram command/query, 565
HITS, :MEASure:HISTogram:HITS command/query, 550
HOLDoff, :TRIGger:HOLDoff command/query, 888
HOLDoff, :TRIGger:HOLDoff:MAX command/query, 889
HOLDoff, :TRIGger:HOLDoff:MIN command/query, 890
HOLDoff, :TRIGger:HOLDoff:MODE command/query, 891
HoldTIMe (HTIMe), :TRIGger:SHOLd:HoldTIMe (HTIMe) command/query, 949
HOLDtime, :MEASure:HOLDtime command/query, 563
horizontal scaling, functions, 326
HORizontal, :FUNCtion<F>:HORizontal command/query, 348
HORizontal, :FUNCtion<F>:HORizontal:POSition command/query, 349
HORizontal, :FUNCtion<F>:HORizontal:RANGe command/query, 350
HORizontal, :MEASure:JITTer:SPECtrum:HORizontal command/query, 568
HORizontal, :MEASure:JITTer:SPECtrum:HORizontal:POSition command/query, 569
HORizontal, :MEASure:JITTer:SPECtrum:HORizontal:RANGe command/query, 570
horizontal, functions, controlling, 871
horizontal, offset, and XOFFset, 1048
horizontal, range, and XRANge, 1049
Host language, 56
HRESolution acquisition mode, 159
HRESolution, :ACQuire:HRESolution command/query, 157
HSYNc, :TRIGger:TV:UDTV:HSYNc command/query, 973
HTHReshold, :TRIGger:HTHReshold command/query, 892
HTIMe, :TRIGger:TV:UDTV:HTIMe command/query, 974
hue, 318
HYSTeresis, :ISCan:NONMonotonic:HYSTeresis command/query, 397
HYSTeresis, :ISCan:RUNT:HYSTeresis command/query, 399
HYSTeresis, :MEASure:THResholds:GENeral:HYSTeresis command/query, 656
HYSTeresis, :MEASure:THResholds:HYSTeresis command/query, 666
HYSTeresis, :MEASure:THResholds:RFALl:HYSTeresis command/query, 674
HYSTeresis, :MEASure:THResholds:SERial:HYSTeresis command/query, 686
HYSTeresis, :TRIGger:HYSTeresis command/query, 893

I

ID, :CHANnel<N>:PROBe:ID? query, 239
Identification Number (*IDN?) query, 257
IEEE 488.1, 101
IEEE 488.1, and IEEE 488.2 relationship, 101
IEEE 488.2, 101
IEEE 488.2, compliance, 101
IEEE 488.2, conformity, 54
IEEE 488.2, Standard, 54
IEEE 488.2, Standard Status Data Structure Model, 106
IIC clock source, 794
IIC data source, 795
IIC serial decode address field size, 793
IIC trigger commands, 792
IIC, :SBUS<N>:IIC:ASIZe command/query, 793
IIC, :SBUS<N>:IIC:SOURce:CLOCk command/query, 794
IIC, :SBUS<N>:IIC:SOURce:DATA command/query, 795
IMAGe, :DISK:SAVE:IMAGe command, 290
IMAGe, :HARDcopy:IMAGe command/query, 375
IMPedance, :MTESt:IMPedance command/query, 460
impedance, input, 205
IMPedance?, :MTESt:PROBe:IMPedance? query, 465
INDex, :ACQuire:SEGMented:INDex command/query, 170
individual commands language, 54
InfiniiScan Commands, 389
Infinity Representation, 144
initialization, 77
initialization, event status, 105
input buffer, 102
Input Buffer, Clearing, 100
input buffer, default condition, 103
input coupling, and COUPling?, 990
INPut, :CHANnel<N>:INPut command/query, 205
instruction headers, 57
Instrument Address, GPIB, 95
instrument status, 87
integer definition, 69
INTegrate, :FUNCtion<F>:INTegrate command, 351
intensity, 309
INTensity, :DISPlay:GRATicule:INTensity command/query, 309
Interface, Capabilities, 93
Interface, Clear (IFC), 100
interface, functions, 89
Interface, GPIB Select Code, 95
INTerpolate, :ACQuire:INTerpolate command/query, 158
INTerpolate, :MEASure:RJDJ:INTerpolate command/query, 632
interpreting commands, parser, 102
interrupted query, 66, 82
Introduction to Programming, 53
INVert, :FUNCtion<F>:INVert command, 352
INVert, :MTESt:INVert command/query, 461
inverting functions, 352
IO library, referencing, 75
IronPython, SCPI.NET example, 1196
ISIM, :CHANnel<N>:ISIM:APPLy command/query, 206
ISIM, :CHANnel<N>:ISIM:BANDwidth command/query, 207
ISIM, :CHANnel<N>:ISIM:BWLimit command/query, 208
ISIM, :CHANnel<N>:ISIM:CONVolve command/query, 209
ISIM, :CHANnel<N>:ISIM:CORRection command/query, 210
ISIM, :CHANnel<N>:ISIM:DECONvolve command/query, 212
ISIM, :CHANnel<N>:ISIM:DELay command/query, 213
ISIM, :CHANnel<N>:ISIM:NORMalize command/query, 215
ISIM, :CHANnel<N>:ISIM:PEXTraction command/query, 216
ISIM, :CHANnel<N>:ISIM:SPAN command/query, 218
ISIM, :CHANnel<N>:ISIM:STATe command/query, 219

J

JITTer, :DISK:SAVE:JITTer command, 291
JITTer, :DISK:SAVE:LISTing command, 292
JITTer, :MEASure:CGRade:JITTer command/query, 498
JITTer, :MEASure:JITTer:HISTogram command/query, 565
JITTer, :MEASure:JITTer:MEASurement command/query, 566
JITTer, :MEASure:JITTer:SPECtrum command/query, 567
JITTer, :MEASure:JITTer:SPECtrum:HORizontal command/query, 568
JITTer, :MEASure:JITTer:SPECtrum:HORizontal:POSition command/query, 569
JITTer, :MEASure:JITTer:SPECtrum:HORizontal:RANGe command/query, 570
JITTer, :MEASure:JITTer:SPECtrum:VERTical command/query, 571
JITTer, :MEASure:JITTer:SPECtrum:VERTical:OFFSet command/query, 572

JITTer, :MEASure:JITTer:SPECtrum:VERTical:RANGe command/query, 573
JITTer, :MEASure:JITTer:SPECtrum:VERTical:TYPE command/query, 574
JITTer, :MEASure:JITTer:SPECtrum:WINDow command/query, 575
JITTer, :MEASure:JITTer:STATistics command/query, 576
JITTer, :MEASure:JITTer:TRENd command/query, 577
JITTer, :MEASure:JITTer:TRENd:SMOoth command/query, 578
JITTer, :MEASure:JITTer:TRENd:SMOoth:POINts command/query, 579
JITTer, :MEASure:JITTer:TRENd:VERTical command/query, 580
JITTer, :MEASure:JITTer:TRENd:VERTical:OFFSet command/query, 581
JITTer, :MEASure:JITTer:TRENd:VERTical:RANGe command/query, 582
JITTer, :STORe:JITTer command, 778
JTF, :MEASure:CLOCk:METHod:JTF command/query, 508

L

LABel, :BUS<N>:LABel command/query, 185
LABel, :CHANnel<N>:LABel command/query, 220
LABel, :DIGital<N>:LABel command/query, 277
LABel, :DISPlay:LABel command/query, 313
LAMPlitude, :MTESt:LAMPlitude command/query, 462
LAN interface, 47, 49
language for program examples, 54
Learn (*LRN?) query, 258
LEVel, :TRIGger:LEVel command/query, 894
LEVels, :DISPlay:CGRade:LEVels? query, 301
Limit Test Commands, 411
LINE, :DISPlay:LINE command, 314
LINE, :TRIGger:TV:LINE command/query, 967
linefeed, 71
List of Error Messages, 1066
Listener Code and Capability, 93
Listeners, Unaddressing All, 100
Lister Commands, 419
LLEVel, :ISCan:RUNT:LLEVel command/query, 400
LLIMit, :HISTogram:WINDow:LLIMit command/query, 384
LLIMit, :ISCan:MEASurement:LLIMit command/query, 392
LLIMit, :LTESt:LLIMit command/query, 413
LOAD, :DISK:LOAD command, 286
LOAD, :MTESt:LOAD command, 463
LOAD, :WMEMory<N>:LOAD command, 1046
loading and saving, 281
LOCation, :MEASure:NOISe:LOCation command/query, 590
LOCK, :SYSTem:LOCK command/query, 864
LOGic, :TRIGger:PATTern:LOGic command/query, 924
LOGic, :TRIGger:STATe:LOGic command/query, 954
long-form headers, 67
LONGform, :SYSTem:LONGform command/query, 865
lowercase, 67
lowercase, headers, 67
LOWPass, :FUNCtion<F>:LOWPass command, 353
LTARget, :SPRocessing:DFEQualizer:TAP:LTARget command/query, 837
LTHReshold, :TRIGger:LTHReshold command/query, 895
LTYPe, :TRIGger:STATe:LTYPe command/query, 955
luminosity, 318

M

M1S, :MEASure:HISTogram:M1S command/query, 551
M2S, :MEASure:HISTogram:M2S command/query, 552
M3S, :MEASure:HISTogram:M3S command/query, 553
MAGNify, :FUNCtion<F>:MAGNify command, 354
MAGNitude, :MEASure:FFT:MAGNitude command/query, 544
making measurements, 486
Marker Commands, 423
Mask Test Commands, 435
Mask Test Commands, DELete, 454
mask, Service Request Enable Register, 268
Master Summary Status (MSS), and *STB, 270
Master Summary Status (MSS), Status Bit, 107
MAV (Message Available), 107
MAV (Message Available), bit, 268, 270
MAX, :MEASure:HISTogram:MAX command/query, 554
MAX, :SPRocessing:DFEQualizer:TAP:MAX command/query, 833
MAX, :TRIGger:HOLDoff:MAX command/query, 889
MAXimum, :FUNCtion<F>:MAXimum command, 355
MDIRectory, :DISK:MDIRectory command, 287
MEAN, :MEASure:HISTogram:MEAN command/query, 555
measure all edges, 532, 534, 548, 576, 606, 608
Measure Commands, 479
Measure Commands, TMAX, 709
Measure Commands, TMIN, 710
Measure Commands, TVOLt, 711
Measure Commands, VMIDdle, 723
MEASure, RESults and statistics, 649
MEASurement, :ISCan:MEASurement command/query, 393
MEASurement, :ISCan:MEASurement:FAIL command/query, 391
MEASurement, :ISCan:MEASurement:LLIMit command/query, 392
MEASurement, :ISCan:MEASurement:ULIMit command/query, 394
MEASurement, :LTESt:MEASurement command/query, 414
MEASurement, :MARKer:MEASurement:MEASurement command, 425
MEASurement, :MEASure:JITTer:MEASurement command/query, 566
measurement, error, 486
measurement, setup, 486
Measurements tab, sizing, 312
MEASurements, :DISK:SAVE:MEASurements command, 293
MEDian, :MEASure:HISTogram:MEDian command/query, 556
memories, and VIEW, 1029
Message (MSG), Status Bit, 107
Message Available (MAV), and *OPC, 260
Message Available (MAV), Status Bit, 107
Message Communications and System Functions, 101
Message Event Register, 113
message exchange protocols, of IEEE 488.2, 102
message, queue, 127
message, termination with hardcopy, 103
METHod, :MEASure:CLOCk:METHod command/query, 502, 1054
METHod, :MEASure:CLOCk:METHod:ALIGn command/query, 504
METHod, :MEASure:CLOCk:METHod:DEEMphasis command/query, 505
METHod, :MEASure:CLOCk:METHod:EDGE command/query, 506
METHod, :MEASure:CLOCk:METHod:JTF command/query, 508
METHod, :MEASure:CLOCk:METHod:OJTF command/query, 510
METHod, :MEASure:NOISe:METHod command/query, 591
METHod, :MEASure:RJDJ:METHod command/query, 633
METHod, :MEASure:THResholds:GENeral:METHod command/query, 658
METHod, :MEASure:THResholds:GENeral:TOPBase:METHod command/query, 664
METHod, :MEASure:THResholds:METHod command/query, 668
METHod, :MEASure:THResholds:RFALl:METHod command/query, 676

METHod,
:MEASure:THResholds:RFALl:TOPBase:METHod command/query, 682
METHod,
:MEASure:THResholds:SERial:METHod command/query, 688
METHod,
:MEASure:THResholds:SERial:TOPBase:METHod command/query, 694
METHod,
:MEASure:THResholds:TOPBase:METHod command/query, 698
MHIStogram, :FUNCtion<F>:MHIStogram command, 356
MIN, :MEASure:HISTogram:MIN command/query, 557
MIN, :SPRocessing:DFEQualizer:TAP:MIN command/query, 834
MIN, :TRIGger:HOLDoff:MIN command/query, 890
MINimum, :FUNCtion<F>:MINimum command, 357
MISO data source, SPI, 804
MISO, :SBUS<N>:SPI:SOURce:MISO command/query, 804
Mnemonic Truncation, 140
MODE, :ACQuire:MODE command/query, 159
MODE, :CHANnel<N>:PROBe:MODE command/query, 240
MODE,
:CHANnel<N>:PROBe:PRECprobe:MODE command, 243
MODE, :HISTogram:MODE command/query, 380
MODE, :ISCan:MODE command/query, 395
MODE, :ISCan:ZONE<N>:MODE command/query, 407
MODE, :MARKer:MODE command/query, 426
MODE, :MEASure:HISTogram:MODE command/query, 558
MODe, :MEASure:RJDJ:MODe command/query, 634
MODE, :TRIGger:DELay:MODE command/query, 911
MODE, :TRIGger:HOLDoff:MODE command/query, 891
MODE, :TRIGger:MODE command/query, 896
MODE, :TRIGger:SHOLd:MODE command/query, 950
MODE, :TRIGger:TV:MODE command/query, 968
mode, serial decode, 786
monitoring events, 105
MOSI data source, SPI, 805
MOSI, :SBUS<N>:SPI:SOURce:MOSI command/query, 805
MSG bit, 269, 271
MSG, bit in the status register, 113
MSS bit and *STB, 270
MTRend, :FUNCtion<F>:MTRend command, 358
Multiple numeric variables, 86
multiple, program commands, 73
multiple, queries, 86
multiple, subsystems, 73
MULTiply, :FUNCtion<F>:MULTiply command, 359

N

N2750A probe, 240
N2893A probe, 225
N5444A probe head, 238
NAME, :MEASure:NAME command/query, 583
NAME, :MEASurement<N>:NAME command/query, 738
NCJitter, :MEASure:NCJitter command/query, 584
NL (New Line), 71
NOISe, :MEASure:NOISe command/query, 586
NOISe, :MEASure:NOISe:ALL? query, 587
NOISe, :MEASure:NOISe:BANDwidth command/query, 589
NOISe, :MEASure:NOISe:LOCation command/query, 590
NOISe, :MEASure:NOISe:METHod command/query, 591
NOISe, :MEASure:NOISe:REPort command/query, 592
NOISe, :MEASure:NOISe:RN command/query, 593
NOISe, :MEASure:NOISe:SCOPe:RN command/query, 594
NOISe, :MEASure:NOISe:STATe command/query, 595
NOISe, :MEASure:NOISe:UNITs command/query, 596
NONMonotonic, :ISCan:NONMonotonic:EDGE command/query, 396
NONMonotonic,
:ISCan:NONMonotonic:HYSTeresis command/query, 397
NONMonotonic,
:ISCan:NONMonotonic:SOURce command/query, 398
NORMalize, :CHANnel<N>:ISIM:NORMalize command/query, 215
notices, 2
NPERiod, :MEASure:NPERiod command/query, 597
NPRecursor,
:SPRocessing:FFEQualizer:NPRecursor command/query, 841
NPULses, :MEASure:NPULses command/query, 598
NREGions?, :MTESt:NREGions? query, 464
NTAPs, :SPRocessing:DFEQualizer:NTAPs command/query, 829
NTAPs, :SPRocessing:FFEQualizer:NTAPs command/query, 842
NUI, :MEASure:NUI command/query, 599
NUMBer, :DISPlay:GRATicule:NUMBer command/query, 310
numeric, program data, 69
numeric, variable example, 84
numeric, variables, 84
NUMPoles,
:SPRocessing:CTLequalizer:NUMPoles command/query, 818
NWIDth, :MEASure:NWIDth command/query, 600

O

obsolete and discontinued commands, 1053
offset and gain of a probe, 187
OFFSet, :CHANnel<N>:DISPlay:OFFSet command/query, 202
OFFSet, :CHANnel<N>:OFFSet command/query, 221
OFFSet, :CHANnel<N>:PROBe:EXTernal:OFFSet command/query, 232
OFFSet, :FUNCtion<F>:OFFSet command/query, 360
OFFSet, :FUNCtion<F>:VERTical:OFFSet command/query, 368
OFFSet, :MEASure:CLOCk:VERTical:OFFSet command/query, 513
OFFSet,
:MEASure:JITTer:SPECtrum:VERTical:OFFSet command/query, 572
OFFSet,
:MEASure:JITTer:TRENd:VERTical:OFFSet command/query, 581
OFFSet,
:SPRocessing:CTLequalizer:VERTical:OFFSet command/query, 824
OFFSet,
:SPRocessing:FFEQualizer:VERTical:OFFSet command/query, 854
OJTF, :MEASure:CLOCk:METHod:OJTF command/query, 510
OPC bit, 255, 256
Open method, 76
OPER bit, 268, 270
operands and time scale, 326
operating the disk, 281
Operation Complete (*OPC) command/query, 260
Operation Complete (*OPC), Status Bit, 107
operation status, 105
Option (*OPT?) query, 261
Options, Program Headers, 67
order of commands and execution, 103
oscilloscope connection, opening, 76
oscilloscope connection, verifying, 48
Oscilloscope Default GPIB Address, 95
oscilloscope, connecting, 47
oscilloscope, operation, 4
oscilloscope, setting up, 47
oscilloscope, trigger modes and commands, 884
output buffer, 66, 82
output queue, 66, 126
Output Queue, Clearing, 100

output queue, default condition, 103
output queue, definition, 102
OUTPut, :CALibrate:OUTPut command/query, 190
overlapped and sequential commands, 145
OVERshoot, :MEASure:OVERshoot command/query, 602

P

P1, :SPRocessing:CTLequalizer:P1 command/query, 819
P2, :SPRocessing:CTLequalizer:P2 command/query, 820
P3, :SPRocessing:CTLequalizer:P3 command/query, 821
PAMPlitude, :MEASure:PAMPlitude command/query, 604
Parallel Poll Code and Capability, 93
parametric measurements, 483
parser, 102
parser, default condition, 103
Parser, Resetting, 100
passing values across the bus, 66
PATTern, :ISCan:SERial:PATTern command/query, 403
PATTern, :TRIGger:COMM:PATTern command/query, 902
PATTern, :TRIGger:PATTern:CONDition command/query, 923
PATTern, :TRIGger:PATTern:LOGic command/query, 924
PBASe, :MEASure:PBASe command/query, 605
PDETect acquisition mode, 159
PEAK, :MEASure:HISTogram:PEAK command/query, 559
PEAK1, :MEASure:FFT:PEAK1 command/query, 545
PEAK2, :MEASure:FFT:PEAK2 command/query, 546
peak-to-peak voltage, and VPP, 727
Pending Commands, Clearing, 100
PERCent, :MEASure:THResholds:GENeral:PERCent command/query, 660
PERCent, :MEASure:THResholds:PERCent command/query, 670
PERCent, :MEASure:THResholds:RFALl:PERCent command/query, 678
PERCent, :MEASure:THResholds:SERial:PERCent command/query, 690
period measurement setup, 486
PERiod, :MEASure:PERiod command/query, 606
PERSistence, :DISPlay:PERSistence command/query, 315
PEXTraction, :CHANnel<N>:ISIM:PEXTraction command/query, 216
PGTHan, :TRIGger:TV:UDTV:PGTHan command/query, 975
PHASe, :MEASure:PHASe command/query, 608
PLACement, :AUToscale:PLACement command/query, 755
PLACement, :ISCan:ZONE<N>:PLACement command/query, 408
PLENgth, :MEASure:RJDJ:PLENgth command/query, 635
PLENgth, :SPRocessing:FFEQualizer:TAP:PLENgth command/query, 845
Pod Commands, 741
POINts, :ACQuire:POINts:ANALog command/query, 161
POINts, :ACQuire:POINts:AUTO command/query, 165
POINts, :ACQuire:POINts:DIGital? query, 166
POINts, :MEASure:JITTer:TRENd:SMOoth:POINts command/query, 579
POINts?, :WAVeform:POINts? query, 1016
POLarity, :TRIGger:COMM:POLarity command/query, 903
POLarity, :TRIGger:GLITch:POLarity command/query, 919
POLarity, :TRIGger:PWIDth:POLarity command/query, 927
POLarity, :TRIGger:RUNT:POLarity command/query, 932
POLarity, :TRIGger:TV:POLarity command/query, 969
POLarity, :TRIGger:TV:UDTV:POLarity command/query, 976
PON bit, 256
POSition, :FUNCtion<F>:HORizontal:POSition command/query, 349
POSition, :MEASure:JITTer:SPECtrum:HORizontal:POSition command/query, 569
POSition, :TIMebase:POSition command/query, 872
POSition, :TIMebase:WINDow:POSition command/query, 879
position, and WINDow POSition, 879
pound sign (#) and block data, 85
Power On (PON) status bit, 106, 254
Power-on Status Clear (*PSC) command/query, 264
Power-up Condition, 92
PP, :MEASure:HISTogram:PP command/query, 560
PPULses, :MEASure:PPULses command/query, 610
PREamble, :WAVeform:PREamble? query, 1017
PRECprobe, :CHANnel<N>:PROBe:PRECprobe:BANDwidth command, 241
PRECprobe, :CHANnel<N>:PROBe:PRECprobe:CALibration command, 242
PRECprobe, :CHANnel<N>:PROBe:PRECprobe:MODE command, 243
PRECprobe, :CHANnel<N>:PROBe:PRECprobe:ZSRC command, 244
PRECprobe, :DISK:SAVE:PRECprobe command, 294
PRESet, :SYSTem:PRESet command, 866
PREShoot, :MEASure:PREShoot command/query, 611
PRINters?, :HARDcopy:PRINters? query, 376
printing, specific screen data, 372
printing, the screen, 371
probe attenuation factor, 188
Probe Calibration, 187
PROBe, :CHANnel<N>:PROBe command/query, 222
PROBe, :CHANnel<N>:PROBe:ACCAL command/query, 223
PROBe, :CHANnel<N>:PROBe:ATTenuation command/query, 224
PROBe, :CHANnel<N>:PROBe:AUTozero command/query, 225
PROBe, :CHANnel<N>:PROBe:COUPling command/query, 226
PROBe, :CHANnel<N>:PROBe:EADapter command/query, 227
PROBe, :CHANnel<N>:PROBe:ECOupling command/query, 229
PROBe, :CHANnel<N>:PROBe:EXTernal command/query, 230
PROBe, :CHANnel<N>:PROBe:EXTernal:GAIN command/query, 231
PROBe, :CHANnel<N>:PROBe:EXTernal:OFFSet command/query, 232
PROBe, :CHANnel<N>:PROBe:EXTernal:UNITs command/query, 233
PROBe, :CHANnel<N>:PROBe:GAIN command/query, 234
PROBe, :CHANnel<N>:PROBe:HEAD:ADD command, 235
PROBe, :CHANnel<N>:PROBe:HEAD:DELete command, 236
PROBe, :CHANnel<N>:PROBe:HEAD:SELect command/query, 237
PROBe, :CHANnel<N>:PROBe:HEAD:VTERm command/query, 238
PROBe, :CHANnel<N>:PROBe:ID? query, 239
PROBe, :CHANnel<N>:PROBe:MODE command/query, 240
PROBe, :CHANnel<N>:PROBe:PRECprobe:BANDwidth command, 241
PROBe, :CHANnel<N>:PROBe:PRECprobe:CALibration command, 242
PROBe, :CHANnel<N>:PROBe:PRECprobe:MODE command, 243

PROBe,
:CHANnel<N>:PROBe:PRECprobe:ZSRC
command, 244
PROBe, :CHANnel<N>:PROBe:SKEW
command/query, 246
PROBe, :CHANnel<N>:PROBe:STYPe
command/query, 247
PROBe, :MTESt:PROBe:IMPedance?
query, 465
program data, 63
Program example, 79
Program Header Options, 67
program message, 76
program message terminator, 71
program overview, initialization example, 79
programming basics, 54
Programming Conventions, 139
programming examples, 1071
programming examples language, 54
Programming Getting Started, 74
protocol, exceptions and operation, 102
PSKew, :POD<N>:PSKew
command/query, 743
PTOP, :MEASure:PTOP command/query, 613
pulse width measurement setup, 486
PWD, :DISK:PWD? query, 288
PWIDth, :MEASure:PWIDth
command/query, 614
PWIDth, :TRIGger:PWIDth:DIRection
command/query, 926
PWIDth, :TRIGger:PWIDth:POLarity
command/query, 927
PWIDth, :TRIGger:PWIDth:SOURce
command/query, 928
PWIDth, :TRIGger:PWIDth:TPOint
command/query, 929
PWIDth, :TRIGger:PWIDth:WIDTh
command/query, 930
Python, VISA COM example, 1103
Python, VISA example, 1154

Q

QFACtor, :MEASure:CGRade:QFACtor
command/query, 499
QUALified, :TRIGger:RUNT:QUALified
command/query, 933
QUALifier<M>,
:MEASure:QUALifier<M>:CONDition
command/query, 616
QUALifier<M>,
:MEASure:QUALifier<M>:SOURce
command/query, 617
QUALifier<M>,
:MEASure:QUALifier<M>:STATe
command/query, 618
Query, 57, 66
Query Error, 1065
Query Error, QYE Status Bit, 107
query interrupt, 82
query, headers, 66
query, interrupt, 66
query, response, 82
query, responses, formatting, 857
question mark, 66
queue, output, 66
quoted strings, 314
quotes, with embedded strings, 70
QYE bit, 255, 256

R

random jitter, specified, 637
random noise, specified, 593
RANGe, :CHANnel<N>:DISPlay:RANGe
command/query, 203
RANGe, :CHANnel<N>:RANGe
command/query, 248
RANGe, :FUNCtion<F>:HORizontal:RANGe
command/query, 350
RANGe, :FUNCtion<F>:RANGe
command/query, 361
RANGe, :FUNCtion<F>:VERTical:RANGe
command/query, 369
RANGe, :MEASure:CLOCk:VERTical:RANGe
command/query, 514
RANGe,
:MEASure:JITTer:SPECtrum:HORizontal:RA
NGe command/query, 570
RANGe,
:MEASure:JITTer:SPECtrum:VERTical:RANG
e command/query, 573
RANGe,
:MEASure:JITTer:TRENd:VERTical:RANGe
command/query, 582
RANGe,
:SPRocessing:CTLequalizer:VERTical:RANG
e command/query, 825
RANGe,
:SPRocessing:FFEQualizer:VERTical:RANGe
command/query, 855
RANGe, :TIMebase:RANGe
command/query, 873
RANGe, :TIMebase:WINDow:RANGe
command/query, 880
range, and WINDow RANGe, 880
RATe, :SPRocessing:CTLequalizer:RATe
command/query, 822
RATe, :SPRocessing:FFEequalizer:RATe
command/query, 843
ReadIEEEBlock method, 76
ReadList method, 76
ReadNumber method, 76
READout, :BUS<N>:READout
command/query, 186
ReadSTB example, 111
ReadString method, 76
real number definition, 69
real time mode, 159
real time mode, and interpolation, 158
Recall (*RCL) command, 265
Receiving Common Commands, 251
Receiving Information from the Instrument, 82
REDGe, :ACQuire:REDGe command/query, 167
REFClock, :TIMebase:REFClock
command/query, 874
REFerence, :FUNCtion<F>:FFT:REFerence
command/query, 339
REFerence, :TIMebase:REFerence
command/query, 875
register, save/recall, 265, 267
register, Standard Event Status Enable, 116
reliability of measured data, 105
remote control examples, 1071
Remote Local Code and Capability, 93
remote programming basics, 54
REPort, :MEASure:NOISe:REPort
command/query, 592
REPort, :MEASure:RJDJ:REPort
command/query, 636
representation of infinity, 144
Request Control (RQC), Status Bit, 107
Request Service (RQS), Default, 92
Request Service (RQS), status bit, 107
Reset (*RST) command, 266
RESet, :TRIGger:SEQuence:RESet:ENABle
command/query, 939
RESet, :TRIGger:SEQuence:RESet:EVENt
command, 941
RESet, :TRIGger:SEQuence:RESet:TIME
command/query, 942
RESet, :TRIGger:SEQuence:RESet:TYPE
command/query, 940
Resetting the Parser, 100
RESolution, :MEASure:HISTogram:RESolution
command/query, 561
RESolution?, :FUNCtion<F>:FFT:RESolution?
query, 340
resource session object, 77
ResourceManager object, 76
RESPonse, :ACQuire:RESPonse
command/query, 168
response, data, 85
response, generation, 146
responses, buffered, 146
result state code, and SENDvalid, 644
RESults?, :LTESt:RESults? query, 415
RESults?, :MEASure:RESults? query, 619
Returning control to system computer, 100
RFALl, :MEASure:THResholds:RFALl:ABSolute
command/query, 672
RFALl, :MEASure:THResholds:RFALl:HYSTeresis
command/query, 674
RFALl, :MEASure:THResholds:RFALl:METHod
command/query, 676
RFALl, :MEASure:THResholds:RFALl:PERCent
command/query, 678
RFALl,
:MEASure:THResholds:RFALl:TOPBase:ABS
olute command/query, 680
RFALl,
:MEASure:THResholds:RFALl:TOPBase:ME
THod command/query, 682
rise time measurement setup, 486
RISetime, :MEASure:RISetime
command/query, 622

RJ, :MEASure:RJDJ:RJ command/query, 637
RJ, :MEASure:RJDJ:SCOPe:RJ command/query, 638
RJDJ, :MEASure:RJDJ:ALL? query, 624
RJDJ, :MEASure:RJDJ:APLength? query, 626
RJDJ, :MEASure:RJDJ:BANDwidth command/query, 627
RJDJ, :MEASure:RJDJ:BER command/query, 628
RJDJ, :MEASure:RJDJ:CLOCk command/query, 630
RJDJ, :MEASure:RJDJ:EDGE command/query, 631
RJDJ, :MEASure:RJDJ:INTerpolate command/query, 632
RJDJ, :MEASure:RJDJ:METHod command/query, 633
RJDJ, :MEASure:RJDJ:MODe command/query, 634
RJDJ, :MEASure:RJDJ:PLENgth command/query, 635
RJDJ, :MEASure:RJDJ:REPort command/query, 636
RJDJ, :MEASure:RJDJ:RJ command/query, 637
RJDJ, :MEASure:RJDJ:SCOPe:RJ command/query, 638
RJDJ, :MEASure:RJDJ:SOURce command/query, 639
RJDJ, :MEASure:RJDJ:STATe command/query, 640
RJDJ, :MEASure:RJDJ:TJRJDJ? query, 641
RJDJ, :MEASure:RJDJ:UNITs command/query, 642
RLIMit, :HISTogram:WINDow:RLIMit command/query, 385
RMS voltage, and VRMS, 730
RN, :MEASure:NOISe:RN command/query, 593
RN, :MEASure:NOISe:SCOPe:RN command/query, 594
Root level commands, 747
ROW, :DISPlay: ROW command query, 316
ROW, :DISPlay:STATus:ROW command query, 320
RQC (Request Control), 107
RQC (Request Control), bit, 255, 256
RQS (Request Service), 107
RQS (Request Service), and *STB, 270
RQS (Request Service), Default, 92
RQS/MSS bit, 270
RSI, Description;Repetitive Strain Injury, Description, 133
RSI, Using the Mouse;Repetitive Strain Injury, Using the Mouse, 134
RTIMe acquisition mode, 159
rule of truncation, 140
rules of traversal, 141
RUMode, :MTESt:RUMode command/query, 466
RUMode, :MTESt:RUMode:SOFailure command/query, 467
run state, 772
RUN, and GET relationship, 100
RUNT, :ISCan:RUNT:HYSTeresis command/query, 399
RUNT, :ISCan:RUNT:LLEVel command/query, 400
RUNT, :ISCan:RUNT:SOURce command/query, 401
RUNT, :ISCan:RUNT:ULEVel command/query, 402
RUNT, :TRIGger:RUNT:POLarity command/query, 932
RUNT, :TRIGger:RUNT:QUALified command/query, 933
RUNT, :TRIGger:RUNT:SOURce command/query, 934
RUNT, :TRIGger:RUNT:TIME command/query, 935

S

SAMPlepoint, :SBUS<N>:CAN:SAMPlepoint command/query, 788
sampling mode, 159
saturation, 318
Save (*SAV) command, 267
SAVE, :DISK:SAVE:COMPosite command, 289
SAVE, :DISK:SAVE:IMAGe command, 290
SAVE, :DISK:SAVE:JITTer command, 291
SAVE, :DISK:SAVE:LISTing command, 292
SAVE, :DISK:SAVE:MEASurements command, 293
SAVE, :DISK:SAVE:PRECprobe command, 294
SAVE, :DISK:SAVE:SETup command, 295
SAVE, :DISK:SAVE:WAVeform command, 296
SAVE, :MTESt:AMASk:SAVE command, 442
SAVE, :WMEMory<N>:SAVE command, 1047
save/recall register, 265, 267
saving and loading, 281
SBUS CAN commands, 787
SBUS<N> commands, general, 784
SCALe, :CHANnel<N>:DISPlay:SCALe command/query, 204
SCALe, :CHANnel<N>:SCALe command/query, 249
SCALe, :HISTogram:SCALe:SIZE command/query, 381
SCALe, :MTESt:SCALe:BIND command/query, 468
SCALe, :MTESt:SCALe:X1 command/query, 469
SCALe, :MTESt:SCALe:XDELta command/query, 470
SCALe, :MTESt:SCALe:Y1 command/query, 471
SCALe, :MTESt:SCALe:Y2 command/query, 472
SCALe, :TIMebase:SCALe command/query, 876
SCALe, :TIMebase:WINDow:SCALe command/query, 881
SCHeme, :DISPlay:CGRade:SCHeme command/query, 303
SCOLor, :DISPlay:SCOLor command/query, 317
SCOPe, :MEASure:NOISe:SCOPe:RN command/query, 594
SCOPe, :MEASure:RJDJ:SCOPe:RJ command/query, 638
SCOPETEST, :SELFtest:SCOPETEST command/query, 811
SCPI.NET example in C#, 1181
SCPI.NET example in IronPython, 1196
SCPI.NET example in Visual Basic .NET, 1189
SCPI.NET examples, 1181
SCRatch, :MEASure:SCRatch command, 643
SCReen, HARDcopy AREA, 372
SEGHres acquisition mode, 160
SEGMented acquisition mode, 160
SEGMented, :ACQuire:SEGMented:COUNt command/query, 169
SEGMented, :ACQuire:SEGMented:INDex command/query, 170
SEGMented, :ACQuire:SEGMented:TTAGs command/query, 171
SEGMented, :DISK:SEGMented command/query, 298
SEGMented, :WAVeform:SEGMented:ALL command/query, 1022
SEGMented, :WAVeform:SEGMented:COUNt? query, 1023
SEGMented, :WAVeform:SEGMented:TTAG? query, 1024
SEGMented, :WAVeform:SEGMented:XLISt? query, 1025
SEGPdetect acquisition mode, 160
SELect, :CHANnel<N>:PROBe:HEAD:SELect command/query, 237
Selected Device Clear (SDC), 100
Selecting Multiple Subsystems, 73
self test, 273
Self-Test Commands, 809
semicolon usage, 65
sending compound queries, 103
SENDvalid, :MEASure:SENDvalid command/query, 644
separator, 58
SEQuence, :TRIGger:SEQuence:RESet:ENABle command/query, 939
SEQuence, :TRIGger:SEQuence:RESet:EVENt command, 941
SEQuence, :TRIGger:SEQuence:RESet:TIME command/query, 942
SEQuence, :TRIGger:SEQuence:RESet:TYPE command/query, 940
SEQuence, :TRIGger:SEQuence:TERM1 command/query, 937
SEQuence, :TRIGger:SEQuence:TERM2 command/query, 938
SEQuence, :TRIGger:SEQuence:WAIT:ENABle command/query, 943
SEQuence, :TRIGger:SEQuence:WAIT:TIME command/query, 944
Sequential and Overlapped Commands, 145

Serial Bus Commands, 783
Serial Data Equalization Commands, 813
serial decode bus display, 785
serial decode mode, 786
serial poll, (ReadSTB) in example, 111
Serial Poll, Disabling, 100
serial poll, of the Status Byte Register, 111
serial prefix, reading, 257
SERial, :ISCan:SERial:PATTern command/query, 403
SERial, :ISCan:SERial:SOURce command/query, 404
SERial, :MEASure:THResholds:SERial:ABSolute command/query, 684
SERial, :MEASure:THResholds:SERial:HYSTeresis command/query, 686
SERial, :MEASure:THResholds:SERial:METHod command/query, 688
SERial, :MEASure:THResholds:SERial:PERCent command/query, 690
SERial, :MEASure:THResholds:SERial:TOPBase:ABSolute command/query, 692
SERial, :MEASure:THResholds:SERial:TOPBase:METHod command/query, 694
Service Request Enabl, (*SRE) command/query, 268
Service Request Enable, Register (SRE), 112
Service Request Enable, Register Bits, 268
Service Request Enable, Register Default, 92
Service Request, Code and Capability, 93
set up oscilloscope, 47
SETGrat, :DISPlay:GRATicule:SETGrat command, 311
setting up, for programming, 74
setting up, the instrument, 78
setting, bits in the Service Request Enable Register, 112
setting, horizontal tracking, 348
setting, Standard Event Status Enable Register bits, 116
setting, time and date, 869
setting, TRG bit, 114
setting, voltage and time markers, 423
setup recall, 265
SETup, :DISK:SAVE:SETup command, 295
SETup, :RECall:SETup command, 771
SETup, :STORe:SETup command, 779
SETup, :SYSTem:SETup command/query, 867
SETuptime, :MEASure:SETuptime command/query, 645
SetupTIMe, :TRIGger:SHOLd:SetupTIMe command/query, 951
SHAPe, :MEASure:TIEFilter:SHAPe command/query, 704
SHOLd, :TRIGger:SHOLd:CSOurce command/query, 946
SHOLd, :TRIGger:SHOLd:CSOurce:EDGE command/query, 947
SHOLd, :TRIGger:SHOLd:DSOurce command/query, 948
SHOLd, :TRIGger:SHOLd:HoldTIMe (HTIMe) command/query, 949
SHOLd, :TRIGger:SHOLd:MODE command/query, 950
SHOLd, :TRIGger:SHOLd:SetupTIMe command/query, 951
Short form, 67
short-form headers, 67
short-form mnemonics, 140
SICL example in C, 1162
SICL example in Visual Basic, 1171
SICL examples, 1162
SIGNal, :SBUS<N>:CAN:SIGNal:BAUDrate command/query, 789
SIGNal, :SBUS<N>:CAN:SIGNal:DEFinition command/query, 790
simple command header, 64
SIZE, :DIGital<N>:SIZE command/query, 278
SIZE, :DISPlay:GRATicule:SIZE command/query, 312
SIZE, :HISTogram:SCALe:SIZE command/query, 381
SKEW, :CALibrate:SKEW command/query, 191
SKEW, :CHANnel<N>:DIFFerential:SKEW command/query, 199
SKEW, :CHANnel<N>:PROBe:SKEW command/query, 246
SLEWrate, :MEASure:SLEWrate command/query, 647
SLOPe, :BUS<N>:CLOCk:SLOPe command/query, 183
SLOPe, :SBUS<N>:SPI:CLOCk:SLOPe command/query, 798
SLOPe, :TRIGger:DELay:ARM:SLOPe command/query, 907
SLOPe, :TRIGger:DELay:EDELay:SLOPe command/query, 910
SLOPe, :TRIGger:DELay:TRIGger:SLOPe command/query, 914
SLOPe, :TRIGger:EDGE:SLOPe command/query, 916
SLOPe, :TRIGger:STATe:SLOPe command/query, 956
SMOoth, :FUNCtion<F>:SMOoth command, 362
SMOoth, :MEASure:JITTer:TRENd:SMOoth command/query, 578
SMOoth, :MEASure:JITTer:TRENd:SMOoth:POINts command/query, 579
SOFailure, :MTESt:RUMode:SOFailure command/query, 467
software version, reading, 257
source, 791
SOURce, :HISTogram:WINDow:SOURce command/query, 383
SOURce, :ISCan:NONMonotonic:SOURce command/query, 398
SOURce, :ISCan:RUNT:SOURce command/query, 401
SOURce, :ISCan:SERial:SOURce command/query, 404
SOURce, :ISCan:ZONE:SOURce command/query, 406
SOURce, :MEASure:QUALifier<M>:SOURce command/query, 617
SOURce, :MEASure:RJDJ:SOURce command/query, 639
SOURce, :MEASure:SOURce command/query, 648
SOURce, :MEASurement<N>:SOURce command/query, 739
SOURce, :MTESt:AMASk:SOURce command/query, 441
SOURce, :MTESt:SOURce command/query, 473
SOURce, :MTESt:TRIGger:SOURce command/query, 478
SOURce, :SBUS<N>:CAN:SOURce command/query, 791
SOURce, :SBUS<N>:IIC:SOURce:CLOCk command/query, 794
SOURce, :SBUS<N>:IIC:SOURce:DATA command/query, 795
SOURce, :SBUS<N>:SPI:SOURce:CLOCk command/query, 801
SOURce, :SBUS<N>:SPI:SOURce:DATA command/query, 802
SOURce, :SBUS<N>:SPI:SOURce:FRAMe command/query, 803
SOURce, :SBUS<N>:SPI:SOURce:MISO command/query, 804
SOURce, :SBUS<N>:SPI:SOURce:MOSI command/query, 805
SOURce, :SPRocessing:CTLequalizer:SOURce command/query, 816
SOURce, :SPRocessing:DFEQualizer:SOURce command/query, 828
SOURce, :SPRocessing:FFEQualizer:SOURce command/query, 840
SOURce, :TRIGger:AND:SOURce command/query, 887
SOURce, :TRIGger:COMM:SOURce command/query, 904
SOURce, :TRIGger:DELay:ARM:SOURce command/query, 906
SOURce, :TRIGger:DELay:EDELay:SOURce command/query, 909
SOURce, :TRIGger:DELay:TRIGger:SOURce command/query, 913
SOURce, :TRIGger:EDGE:SOURce command/query, 917
SOURce, :TRIGger:GLITch:SOURce command/query, 920
SOURce, :TRIGger:PWIDth:SOURce command/query, 928
SOURce, :TRIGger:RUNT:SOURce command/query, 934
SOURce, :TRIGger:TIMeout:SOURce command/query, 959
SOURce, :TRIGger:TRANsition:SOURce command/query, 963

SOURce, :TRIGger:TV:SOURce command/query, 970
SOURce, :TRIGger:WINDow:SOURce command/query, 979
SOURce, :WAVeform:SOURce command/query, 1026
SOURce, and measurements, 487
spaces and commas, 63
SPAN, :CHANnel<N>:ISIM:SPAN command/query, 218
specified random jitter, 637
specified random noise, 593
SPECtrum, :MEASure:JITTer:SPECtrum command/query, 567
SPECtrum, :MEASure:JITTer:SPECtrum:HORizontal command/query, 568
SPECtrum, :MEASure:JITTer:SPECtrum:HORizontal:POSition command/query, 569
SPECtrum, :MEASure:JITTer:SPECtrum:HORizontal:RANGe command/query, 570
SPECtrum, :MEASure:JITTer:SPECtrum:VERTical command/query, 571
SPECtrum, :MEASure:JITTer:SPECtrum:VERTical:OFFSet command/query, 572
SPECtrum, :MEASure:JITTer:SPECtrum:VERTical:RANGe command/query, 573
SPECtrum, :MEASure:JITTer:SPECtrum:VERTical:TYPE command/query, 574
SPECtrum, :MEASure:JITTer:SPECtrum:WINDow command/query, 575
spelling of headers, 67
SPI clock slope, 798
SPI clock source, 801
SPI clock timeout, 799
SPI decode bit order, 797
SPI decode type, 806
SPI decode word width, 807
SPI frame source, 803
SPI frame state, 800
SPI trigger commands, 796
SPI, :SBUS<N>:SPI:BITorder command/query, 797
SPI, :SBUS<N>:SPI:CLOCk:SLOPe command/query, 798
SPI, :SBUS<N>:SPI:CLOCk:TIMeout command/query, 799
SPI, :SBUS<N>:SPI:FRAMe:STATe command/query, 800
SPI, :SBUS<N>:SPI:SOURce:CLOCk command/query, 801
SPI, :SBUS<N>:SPI:SOURce:DATA command/query, 802
SPI, :SBUS<N>:SPI:SOURce:FRAMe command/query, 803
SPI, :SBUS<N>:SPI:SOURce:MISO command/query, 804
SPI, :SBUS<N>:SPI:SOURce:MOSI command/query, 805
SPI, :SBUS<N>:SPI:TYPE command/query, 806
SPI, :SBUS<N>:SPI:WIDTh command/query, 807
SQRT, :FUNCtion<F>:SQRT command, 363
Square Brackets, 61
SQUare, :FUNCtion<F>:SQUare command, 364
SRATe, :ACQuire:SRATe:ANALog command/query, 172
SRATe, :ACQuire:SRATe:ANALog:AUTO command/query, 174
SRATe, :ACQuire:SRATe:DIGital command/query, 175
SRATe, :ACQuire:SRATe:DIGital:AUTO command/query, 176
SRE (Service Request Enable Register), 112
Standard Event Status Enable Register, (SESER), 116
Standard Event Status Enable Register, Bits, 254
Standard Event Status Enable Register, Default, 92
Standard Event Status Register (ESR), 115
Standard Event Status Register, bits, 256
Standard Status Data Structure Model, 106
STANdard, :TRIGger:TV:STANdard command/query, 971
STARt, :MEASure:TIEFilter:STARt command/query, 705
STARt, :MTESt:STARt command, 474
STATe, :ACQuire:COMPlete:STATe command/query, 156
STATe, :CHANnel<N>:ISIM:STATe command/query, 219
STATe, :ISCan:ZONE<N>:STATe command/query, 409
STATe, :MEASure:NOISe:STATe command/query, 595
STATe, :MEASure:QUALifier<M>:STATe command/query, 618
STATe, :MEASure:RJDJ:STATe command/query, 640
STATe, :MEASure:TIEFilter:STATe command/query, 706
STATe, :SBUS<N>:SPI:FRAMe:STATe command/query, 800
STATe, :SPRocessing:DFEQualizer:STATe command/query, 827
STATe, :TRIGger:STATe:CLOCk command/query, 953
STATe, :TRIGger:STATe:LOGic command/query, 954
STATe, :TRIGger:STATe:LTYPe command/query, 955
STATe, :TRIGger:STATe:SLOPe command/query, 956
state, acquisition, 751
state, run, 772
STATistics, :MEASure:JITTer:STATistics command/query, 576
STATistics, :MEASure:STATistics command/query, 649
status, 87
Status Byte (*STB?) query, 270
Status Byte Register, 110
Status Byte Register, and serial polling, 111
Status Byte Register, bits, 270
Status Registers, 87, 252
Status Reporting, 105
Status Reporting Decision Chart, 128
Status Reporting, Bit Definitions, 106
Status Reporting, Data Structures, 108
STATus, :CALibrate:STATus? query, 192
STATus, :DISPlay:STATus:COL command query, 319
STATus, :DISPlay:STATus:ROW command query, 320
status, of an operation, 105
STDDev, :MEASure:HISTogram:STDDev command/query, 562
STIMe, :MTESt:STIMe command/query, 476
STOP, :MEASure:TIEFilter:STOP command/query, 707
STOP, :MTESt:STOP command, 475
STReaming, :WAVeform:STReaming command/query, 1027
string variables, 83
string variables, example, 83
STRing, :DISPlay:STRing command, 321
string, quoted, 314
strings, alphanumeric, 68
STYPe, :CHANnel<N>:PROBe:STYPe command/query, 247
SUBTract, :FUNCtion<F>:SUBTract command, 365
suffix multipliers, 69, 103
suffix units, 104
summary bits, 110
SWEep, :TRIGger:SWEep command/query, 898
syntax error, 1062
System Commands, 857
System Computer, Returning control to, 100
SYSTem:SETup and *LRN, 259

T

TAB, :DISPlay:TAB command/query, 322
Talker, Code and Capability, 93
Talker, Unaddressing, 100
TAP, :SPRocessing:DFEQualizer:TAP command/query, 830
TAP, :SPRocessing:DFEQualizer:TAP:AUTomatic command, 838
TAP, :SPRocessing:DFEQualizer:TAP:DELay command/query, 832
TAP, :SPRocessing:DFEQualizer:TAP:GAIN command/query, 835

TAP, :SPRocessing:DFEQualizer:TAP:LTARget command/query, 837
TAP, :SPRocessing:DFEQualizer:TAP:MAX command/query, 833
TAP, :SPRocessing:DFEQualizer:TAP:MIN command/query, 834
TAP, :SPRocessing:DFEQualizer:TAP:UTARget command/query, 836
TAP, :SPRocessing:DFEQualizer:TAP:WIDTh command/query, 831
TAP, :SPRocessing:FFEQualizer:TAP command/query, 844
TAP, :SPRocessing:FFEQualizer:TAP:AUTomatic command, 848
TAP, :SPRocessing:FFEQualizer:TAP:BANDwidth command/query, 849
TAP, :SPRocessing:FFEQualizer:TAP:BWMode command/query, 850
TAP, :SPRocessing:FFEQualizer:TAP:DELay command/query, 847
TAP, :SPRocessing:FFEQualizer:TAP:PLENgth command/query, 845
TAP, :SPRocessing:FFEQualizer:TAP:TDELay command/query, 851
TAP, :SPRocessing:FFEQualizer:TAP:TDMode command/query, 852
TAP, :SPRocessing:FFEQualizer:TAP:WIDTh command/query, 846
TDELay, :FUNCtion<F>:FFT:TDELay command/query, 341
TDELay, :SPRocessing:FFEQualizer:TAP:TDELay command/query, 851
TDELay, :TRIGger:DELay:TDELay:TIME command/query, 912
TDMode, :SPRocessing:FFEQualizer:TAP:TDMode command/query, 852
TEDGe, :MEASure:TEDGe command/query, 650
TEMP, :CALibrate:TEMP? query, 193
temperature and calibration, 187
temperature color grade scheme, 303
TERM1, :TRIGger:SEQuence:TERM1 command/query, 937
TERM2, :TRIGger:SEQuence:TERM2 command/query, 938
termination of message during hardcopy, 103
termination voltage for N5444A probe head, 238
Terminator, 71
Test (*TST?) query, 273
TEST, :LTESt:TEST command/query, 416
TEXT, :DISPlay:TEXT command, 323
THReshold, :DIGital<N>:THReshold command/query, 279
THReshold, :MEASure:FFT:THReshold command/query, 547
THReshold, :POD<N>:THReshold command/query, 744
THResholds, :MEASure:THResholds:ABSolute command/query, 652
THResholds, :MEASure:THResholds:GENeral:ABSolute command/query, 654
THResholds, :MEASure:THResholds:GENeral:HYSTeresis command/query, 656
THResholds, :MEASure:THResholds:GENeral:METHod command/query, 658
THResholds, :MEASure:THResholds:GENeral:PERCent command/query, 660
THResholds, :MEASure:THResholds:GENeral:TOPBase:ABSolute command/query, 662
THResholds, :MEASure:THResholds:GENeral:TOPBase:METHod command/query, 664
THResholds, :MEASure:THResholds:HYSTeresis command/query, 666
THResholds, :MEASure:THResholds:METHod command/query, 668
THResholds, :MEASure:THResholds:PERCent command/query, 670
THResholds, :MEASure:THResholds:RFALI:ABSolute command/query, 672
THResholds, :MEASure:THResholds:RFALI:HYSTeresis command/query, 674
THResholds, :MEASure:THResholds:RFALI:METHod command/query, 676
THResholds, :MEASure:THResholds:RFALI:PERCent command/query, 678
THResholds, :MEASure:THResholds:RFALI:TOPBase:ABSolute command/query, 680
THResholds, :MEASure:THResholds:RFALI:TOPBase:METHod command/query, 682
THResholds, :MEASure:THResholds:SERial:ABSolute command/query, 684
THResholds, :MEASure:THResholds:SERial:HYSTeresis command/query, 686
THResholds, :MEASure:THResholds:SERial:METHod command/query, 688
THResholds, :MEASure:THResholds:SERial:PERCent command/query, 690
THResholds, :MEASure:THResholds:SERial:TOPBase:ABSolute command/query, 692
THResholds, :MEASure:THResholds:SERial:TOPBase:METHod command/query, 694
THResholds, :MEASure:THResholds:TOPBase:ABSolute command/query, 696
THResholds, :MEASure:THResholds:TOPBase:METHod command/query, 698
TIEClock2, :MEASure:TIEClock2 command/query, 700
TIEData, :MEASure:TIEData command/query, 702
TIEFilter, :MEASure:TIEFilter:SHAPe command/query, 704
TIEFilter, :MEASure:TIEFilter:STARt command/query, 705
TIEFilter, :MEASure:TIEFilter:STATe command/query, 706
TIEFilter, :MEASure:TIEFilter:STOP command/query, 707
TIEFilter, :MEASure:TIEFilter:TYPE command/query, 708
time and date, setting, 857
Time Base Commands, 871
time buckets, and POINts?, 1016
time scale, operands and functions, 326
TIME, :SYSTem:TIME command/query, 869
TIME, :TRIGger:DELay:TDELay:TIME command/query, 912
TIME, :TRIGger:RUNT:TIME command/query, 935
TIME, :TRIGger:SEQuence:RESet:TIME command/query, 942
TIME, :TRIGger:SEQuence:WAIT:TIME command/query, 944
TIME, :TRIGger:TIMeout:TIME command/query, 960
TIME, :TRIGger:TRANsition:TIME command/query, 964
TIME, :TRIGger:WINDow:TIME command/query, 980
TIMeout, :SBUS<N>:SPI:CLOCk:TIMeout command/query, 799
TIMeout, :TRIGger:TIMeout:CONDition command/query, 958
TIMeout, :TRIGger:TIMeout:SOURce command/query, 959
TIMeout, :TRIGger:TIMeout:TIME command/query, 960
timeout, SPI clock, 799
TITLe?, :MTESt:TITLe? query, 477
TJRJDJ?, :MEASure:RJDJ:TJRJDJ? query, 641
TLIMit, :HISTogram:WINDow:TLIMit command/query, 387
TMAX, :MEASure:TMAX command/query, 709
TMIN, :MEASure:TMIN command/query, 710
TOPBase, :MEASure:THResholds:GENeral:TOPBase:ABSolute command/query, 662
TOPBase, :MEASure:THResholds:GENeral:TOPBase:METHod command/query, 664

TOPBase,
:MEASure:THResholds:RFALl:TOPBase:ABS
olute command/query, 680
TOPBase,
:MEASure:THResholds:RFALl:TOPBase:ME
THod command/query, 682
TOPBase,
:MEASure:THResholds:SERial:TOPBase:AB
Solute command/query, 692
TOPBase,
:MEASure:THResholds:SERial:TOPBase:ME
THod command/query, 694
TOPBase,
:MEASure:THResholds:TOPBase:ABSolute
command/query, 696
TOPBase,
:MEASure:THResholds:TOPBase:METHod
command/query, 698
TPOint, :TRIGger:PWIDth:TPOint
command/query, 929
TPOint, :TRIGger:WINDow:TPOint
command/query, 981
trademarks, 2
transferring waveform data, 984
TRANsition, :TRIGger:TRANsition:DIRection
command/query, 962
TRANsition, :TRIGger:TRANsition:SOURce
command/query, 963
TRANsition, :TRIGger:TRANsition:TIME
command/query, 964
TRANsition, :TRIGger:TRANsition:TYPE
command/query, 965
transmission mode, and FORMat, 1013
traversal rules, 141
Tree Traversal, Examples, 142
Tree Traversal, Rules, 141
TRENd, :MEASure:JITTer:TRENd
command/query, 577
TRENd, :MEASure:JITTer:TRENd:SMOoth
command/query, 578
TRENd,
:MEASure:JITTer:TRENd:SMOoth:POINts
command/query, 579
TRENd, :MEASure:JITTer:TRENd:VERTical
command/query, 580
TRENd,
:MEASure:JITTer:TRENd:VERTical:OFFSet
command/query, 581
TRENd,
:MEASure:JITTer:TRENd:VERTical:RANGe
command/query, 582
TRG, bit, 269, 271
TRG, bit in the status byte, 114
TRG, Event Enable Register, 107
Trigger (*TRG) command, 272
Trigger Commands, 883
Trigger Event Register (TRG), 114
TRIGger IIC commands, 792
trigger mode, 884
trigger modes, summary, 884
trigger other instruments, 190
TRIGger SPI commands, 796
TRIGger, :MTESt:TRIGger:SOURce
command/query, 478
TRIGger, :TRIGger:DELay:TRIGger:SLOPe
command/query, 914
TRIGger, :TRIGger:DELay:TRIGger:SOURce
command/query, 913
Trigger, *TRG status bit, 107
trigger, CAN sample point, 788
trigger, CAN signal baudrate, 789
trigger, CAN signal definition, 790
trigger, CAN source, 791
truncating numbers, 69
Truncation Rule, 140
TTAG?, :WAVeform:SEGMented:TTAG?
query, 1024
TTAGs, :ACQuire:SEGMented:TTAGs
command/query, 171
TV, :TRIGger:TV:LINE command/query, 967
TV, :TRIGger:TV:MODE command/query, 968
TV, :TRIGger:TV:POLarity command/query, 969
TV, :TRIGger:TV:SOURce command/query, 970
TV, :TRIGger:TV:STANdard
command/query, 971
TV, :TRIGger:TV:UDTV:ENUMber
command/query, 972
TV, :TRIGger:TV:UDTV:HSYNc
command/query, 973
TV, :TRIGger:TV:UDTV:HTIMe
command/query, 974
TV, :TRIGger:TV:UDTV:PGTHan
command/query, 975
TV, :TRIGger:TV:UDTV:POLarity
command/query, 976
TVOLt, :MEASure:TVOLt command/query, 711
TYPE, :BUS:B<N>:TYPE command/query, 178
TYPE,
:MEASure:JITTer:SPECtrum:VERTical:TYPE
command/query, 574
TYPE, :MEASure:TIEFilter:TYPE
command/query, 708
TYPE, :SBUS<N>:SPI:TYPE
command/query, 806
TYPE, :TRIGger:SEQuence:RESet:TYPE
command/query, 940
TYPE, :TRIGger:TRANsition:TYPE
command/query, 965
TYPE?, :WAVeform:TYPE? query, 1028

U

UDTV, :TRIGger:TV:UDTV:ENUMber
command/query, 972
UDTV, :TRIGger:TV:UDTV:HSYNc
command/query, 973
UDTV, :TRIGger:TV:UDTV:HTIMe
command/query, 974
UDTV, :TRIGger:TV:UDTV:PGTHan
command/query, 975
UDTV, :TRIGger:TV:UDTV:POLarity
command/query, 976
UI?, :MTESt:COUNt:UI? query, 452
UITouijitter, :MEASure:UITouijitter
command/query, 713
ULEVel, :ISCan:RUNT:ULEVel
command/query, 402
ULIMit, :ISCan:MEASurement:ULIMit
command/query, 394
ULIMit, :LTESt:ULIMit command/query, 417
Unaddressing all listeners, 100
UNITinterval, :MEASure:UNITinterval
command/query, 714
UNITs, :CHANnel<N>:PROBe:EXTernal:UNITs
command/query, 233
UNITs, :CHANnel<N>:UNITs
command/query, 250
UNITs, :MEASure:NOISe:UNITs
command/query, 596
UNITs, :MEASure:RJDJ:UNITs
command/query, 642
UNITs, :MTESt:AMASk:UNITs
command/query, 443
units, vertical, 233, 250
UNKNown vertical units, 233, 250
uppercase, 67
uppercase, headers, 67
uppercase, letters and responses, 68
URQ bit (User Request), 254
USB (Device) interface, 47
User Request (URQ) status bit, 106
User Request Bit (URQ), 254
User's Guide, 4
User-Defined Measurements, 486
Using the Digitize Command, 80
USR bit, 269, 271
UTARget,
:SPRocessing:DFEQualizer:TAP:UTARget
command/query, 836

V

VAMPlitude, :MEASure:VAMPlitude
command/query, 716
VAVerage, :MEASure:VAVerage
command/query, 717
VBA, 75, 1072
VBASe, :MEASure:VBASe
command/query, 719
version of software, reading, 257
VERSus, :FUNCtion<F>:VERSus
command, 366
vertical axis, full-scale, 248
vertical scaling, functions, 326
vertical units, 233, 250
VERTical, :AUToscale:VERTical command, 756
VERTical, :FUNCtion<F>:VERTical
command/query, 367
VERTical, :FUNCtion<F>:VERTical:OFFSet
command/query, 368
VERTical, :FUNCtion<F>:VERTical:RANGe
command/query, 369
VERTical, :MEASure:CLOCk:VERTical
command/query, 512

VERTical, :MEASure:CLOCk:VERTical:OFFSet command/query, 513
VERTical, :MEASure:CLOCk:VERTical:RANGe command/query, 514
VERTical, :MEASure:JITTer:SPECtrum:VERTical command/query, 571
VERTical, :MEASure:JITTer:SPECtrum:VERTical:OFFSet command/query, 572
VERTical, :MEASure:JITTer:SPECtrum:VERTical:RANGe command/query, 573
VERTical, :MEASure:JITTer:SPECtrum:VERTical:TYPE command/query, 574
VERTical, :MEASure:JITTer:TRENd:VERTical command/query, 580
VERTical, :MEASure:JITTer:TRENd:VERTical:OFFSet command/query, 581
VERTical, :MEASure:JITTer:TRENd:VERTical:RANGe command/query, 582
VERTical, :SPRocessing:CTLequalizer:VERTical command/query, 823
VERTical, :SPRocessing:CTLequalizer:VERTical:OFFSet command/query, 824
VERTical, :SPRocessing:CTLequalizer:VERTical:RANGe command/query, 825
VERTical, :SPRocessing:FFEQualizer:VERTical command/query, 853
VERTical, :SPRocessing:FFEQualizer:VERTical:OFFSet command/query, 854
VERTical, :SPRocessing:FFEQualizer:VERTical:RANGe command/query, 855
vertical, axis control, 196
vertical, axis offset, and YRANge, 1050
vertical, scaling, and YRANge, 1051
VIEW and BLANk, 758
VIEW, :TIMebase:VIEW command/query, 877
VIEW, :WAVeform:VIEW command/query, 1029
VISA COM example in C#, 1083
VISA COM example in Python, 1103
VISA COM example in Visual Basic, 1072
VISA COM example in Visual Basic .NET, 1093
VISA example in C, 1111
VISA example in C#, 1130
VISA example in Python, 1154
VISA example in Visual Basic, 1120
VISA example in Visual Basic .NET, 1142
VISA examples, 1072, 1111
Visual Basic .NET, SCPI.NET example, 1189
Visual Basic .NET, VISA COM example, 1093
Visual Basic .NET, VISA example, 1142
Visual Basic 6.0, 75
Visual Basic for Applications, 75, 1072
Visual Basic for Applications (VBA), 54
Visual Basic, SICL library example, 1171
Visual Basic, VISA COM example, 1072
Visual Basic, VISA example, 1120
VLOWer, :MEASure:VLOWer command/query, 720
VMAX, :MEASure:VMAX command/query, 721
VMIDdle, :MEASure:VMIDdle command/query, 723
VMIN, :MEASure:VMIN command/query, 724
voltage at center screen, 221, 232
VOLTS as vertical units, 233, 250
VOVershoot, :MEASure:VOVershoot command/query, 726
VPP, :MEASure:VPP command/query, 727
VPReshoot, :MEASure:VPReshoot command/query, 729
VRMS, :MEASure:VRMS command/query, 730
VTERm, :CHANnel<N>:PROBe:HEAD:VTERm command/query, 238
VTIMe, :MEASure:VTIMe command/query, 732
VTOP, :MEASure:VTOP command/query, 734
VUPPer, :MEASure:VUPPer command/query, 735

W

WAIT, :TRIGger:SEQuence:WAIT:ENABle command/query, 943
WAIT, :TRIGger:SEQuence:WAIT:TIME command/query, 944
Wait-to-Continue (*WAI) command, 274
WATTS as vertical units, 233, 250
Waveform Commands, 983
Waveform Memory Commands, 1043
waveform type, and COMPlete?, 988
waveform type, and COUNt?, 989
WAVeform, :DISK:SAVE:WAVeform command, 296
WAVeform, :STORe:WAVeform command, 780
waveform, data and preamble, 984
waveform, saving, 296
waveform, view parameters, 1029
WAVeforms?, :MTESt:COUNt:WAVeforms? query, 453
what's new, 29
white space (separator), 58
WIDTh, :SBUS<N>:SPI:WIDTh command/query, 807
WIDTh, :SPRocessing:DFEQualizer:TAP:WIDTh command/query, 831
WIDTh, :SPRocessing:FFEQualizer:TAP:WIDTh command/query, 846
WIDTh, :TRIGger:GLITch:WIDTh command/query, 921
WIDTh, :TRIGger:PWIDth:WIDTh command/query, 930
WINDow, :FUNCtion<F>:FFT:WINDow command/query, 342
WINDow, :HISTogram:WINDow:BLIMit command/query, 386
WINDow, :HISTogram:WINDow:DEFault command, 382
WINDow, :HISTogram:WINDow:LLIMit command/query, 384
WINDow, :HISTogram:WINDow:RLIMit command/query, 385
WINDow, :HISTogram:WINDow:SOURce command/query, 383
WINDow, :HISTogram:WINDow:TLIMit command/query, 387
WINDow, :MEASure:JITTer:SPECtrum:WINDow command/query, 575
WINDow, :MEASure:WINDow command/query, 737
WINDow, :TIMebase:WINDow:DELay command/query, 878
WINDow, :TIMebase:WINDow:POSition command/query, 879
WINDow, :TIMebase:WINDow:RANGe command/query, 880
WINDow, :TIMebase:WINDow:SCALe command/query, 881
WINDow, :TRIGger:WINDow:CONDition command/query, 978
WINDow, :TRIGger:WINDow:SOURce command/query, 979
WINDow, :TRIGger:WINDow:TIME command/query, 980
WINDow, :TRIGger:WINDow:TPOint command/query, 981
word width, SPI decode, 807
WORD, and FORMat, 1014
WORD, Understanding the format, 1008
WriteIEEEBlock method, 76
WriteList method, 76
WriteNumber method, 76
WriteString method, 76
WriteString VISA COM method, 55
writing, text to the screen, 321

X

x axis, controlling, 871
X vs Y, 366
X1, :MTESt:SCALe:X1 command/query, 469
X1Position, :MARKer:X1Position command/query, 427
X1Y1source, :MARKer:X1Y1source command/query, 429
X2Position, :MARKer:X2Position command/query, 428
X2Y2source, :MARKer:X2Y2source command/query, 430
x-axis duration, and XRANge?, 1034
x-axis, offset, and XOFFset, 1048
x-axis, range, and XRANge, 1049
x-axis, units and XUNits, 1036
XDELta, :MTESt:AMASk:XDELta command/query, 444
XDELta, :MTESt:SCALe:XDELta command/query, 470
XDELta?, :MARKer:XDELta? query, 431

XDISplay?, :WAVeform:XDISplay? query, 1031
XINCrement?, :WAVeform:XINCrement? query, 1032
XLISt?, :WAVeform:SEGMented:XLISt? query, 1025
XOFFset, :WMEMory<N>:XOFFset command/query, 1048
XORigin?, :WAVeform:XORigin? query, 1033
XRANge, :WMEMory<N>:XRANge command/query, 1049
XRANge?, :WAVeform:XRANge? query, 1034
XREFerence?, :WAVeform:XREFerence? query, 1035
XUNits?, :WAVeform:XUNits? query, 1036

Y

Y1, :MTESt:SCALe:Y1 command/query, 471
Y1Position, :MARKer:Y1Position command/query, 432
Y2, :MTESt:SCALe:Y2 command/query, 472
Y2Position, :MARKer:Y2Position command/query, 433
Y-axis control, 196
YDELta, :MTESt:AMASk:YDELta command/query, 445
YDELta?, :MARKer:YDELta? query, 434
YDISplay?, :WAVeform:YDISplay? query, 1037
YINCrement?, :WAVeform:YINCrement? query, 1038
YOFFset, :WMEMory<N>:YOFFset command/query, 1050
YORigin?, :WAVeform:YORigin? query, 1039
YRANge, :WMEMory<N>:YRANge command/query, 1051
YRANge?, :WAVeform:YRANge? query, 1040
YREFerence?, :WAVeform:YREFerence? query, 1041
YUNits?, :WAVeform:YUNits? query, 1042

Z

ZERo, :SPRocessing:CTLequalizer:ZERo command/query, 826
ZONE, :ISCan:ZONE:HIDE command/query, 405
ZONE, :ISCan:ZONE:SOURce command/query, 406
ZONE<N>, :ISCan:ZONE<N>:MODE command/query, 407
ZONE<N>, :ISCan:ZONE<N>:PLACement command/query, 408
ZONE<N>, :ISCan:ZONE<N>:STATe command/query, 409
ZSRC, :CHANnel<N>:PROBe:PRECprobe:ZSRC command, 244